Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Computing Measures for Tree-Basedness of Phylogenetic Networks

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

Arthur Mooiman

Delft, Nederland
April 18, 2018

BSc verslag TECHNISCHE WISKUNDE

"Computing Measures For tree-basedness of Phylogenetic Networks"

Arthur Mooiman

Technische Universiteit Delft

Begeleider

Leo van Iersel dr. ir.

Overige commissieleden

K.P. Hart	J. Spandaw

April 18, 2018	Delft

Contents

Abstract

Phylogenetic networks are a type of directed acyclic graph used to represent evolutionary relationships that contain events such as hybridization or horizontal gene transfer. When a network lacks such events it is a phylogenetic tree. Some phylogenetic networks that are not trees can however be represented as a tree with additional linking arcs, e.g. representing transfer of genetic materials. We have implemented an algorithm that can be used to determine whether a given network is tree-based or not. Moreover if the network is not tree-based, the algorithm shows how it can be made tree-based by adding a minimum number of additional leaves, representing possible extinct or un-sampled species. We also describe the theory behind the algorithm and apply it to several synthetic as well as biological datasets.

1 Introduction

In the last few years there has been an increasing popularity in phylogenetic networks to represent evolutionary relationships. The benefit such a network has over a phylogenetic tree is that it allows vertices to have more than one incoming arc. The convergence of arcs is an important phenomenon in evolution as it can for example describe hybridization or horizontal gene transfer. These events are also known as *reticulate evolutionary events* and these vertices are called *reticulations*. An example of hybridization can be seen in the mule. It is the offspring of a female horse and a male donkey. While a mule is not fertile, hybrids in for example plants are fertile and form new species. If a phylogenetic network has no reticulate evolutionary events it is simply a phylogenetic tree.

Phylogenetic networks that are not trees can sometimes be seen as trees with additional linking arcs between vertices of the tree. Francis and Steel [1] defined this class of phylogenetic networks as *"tree-based"* and studied them. Since then multiple studies have been carried out on tree-based networks in a number of papers ([2], [3], [4], [5]). With the study of tree-based phylogenetic networks a few ways have been described to determine if a phylogenetic network is tree-based or not. Francis, Semple and Steel [6] described several characterizations of tree-based networks. They also provided a polynomial time algorithm to find these characterizations. Each of these characterizations provides an index to see how close an arbitrary phylogenetic network $\mathcal{N}$ is to being tree-based, based on anti-chains, path partitions and matching in auxiliary bipartite graphs using previous results from [2] and [5]. Most of these characterizations have been defined for binary networks so in this thesis all networks are assumed to binary. The property of tree-basedness can be used in several different fields of research which use phylogenetic networks such as in viruses, bacteria and plants. Determining whether a network is tree-based, or how close it is to being tree-based, can be used to determine how "tree-like" the evolutionary history of the considered species has been.

The characterizations which are used to measure how close a network is to being tree-based are $l(\mathcal{N})$, $p(\mathcal{N})$ and $t(\mathcal{N})$. The first measure, $l(\mathcal{N})$, is equal to the amount of extra leaves in a rooted spanning tree of a network. The second measure, $p(\mathcal{N})$, is equal to the minimum number of disjoint paths a network can be partitioned into. The last measure is $t(\mathcal{N})$, which is equal to the number of leaves that need to be added to the network for it to be tree-based. Each of these measures can be seen as a distance from being tree-based and is equal to zero precisely when the network is tree-based.

The algorithms given by [6] can be followed by hand to calculate these distance measures of a network to determine if it is tree-based. These algorithms work quick for smaller networks but becomes a painstaking job when the network is larger. Take for example the Viola network, based on the viola genus from [7] which holds over 600 species. The viola network can be seen in Figure 1, each leaf in the network represents a large number of species. If it was to be determined if the network is tree-based or not, one could manually apply theorems or follow the algorithms which will take a while to complete.

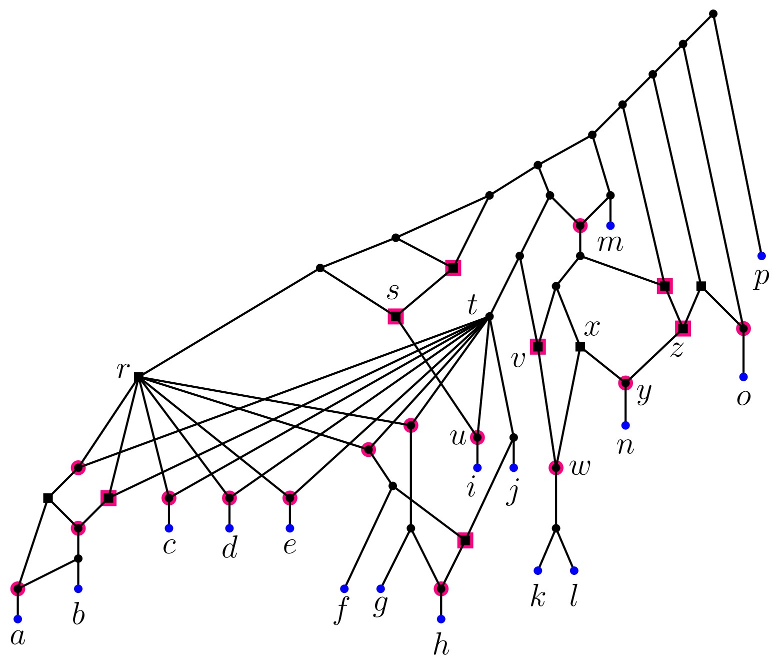


Figure 1: The viola network, an example from [2]

To avoid doing all the manual labor we programmed the algorithms for the characterizations in java. All the user has to do is provide the program with a newick string or an edge set that represents the network. In this thesis we explain the theory on which the program is based as well as give some examples on how the algorithm works and, of course, the program itself and where one can download it from.

The second chapter of this thesis is used to introduce the definition of tree-based networks and other that are used in this thesis. In Chapter 3 we explain the measures described in [6] and the algorithms used to find them. In Chapter 4 we explain how the algorithms were programmed. In Chapter 5 we will apply the algorithm to two synthetic networks used throughout the paper and to several networks from existing literature.

2 Definitions

Definition 2.1. (Phylogenetic Network)

A (rooted binary) phylogenetic network $\mathcal{N} = (V, E)$ is a rooted acyclic graph which contains the following types of vertices:

(i) a unique root vertex with out-degree two.
(ii) *tree vertices* with in-degree one and out-degree two.
(iii) vertices with in-degree two and out-degree one, called *reticulations*.
(iv) vertices with in-degree one and out-degree zero, called *leaves*. In addition, the set of leaves will be a non-empty finite set X.

In Figure 2 a small example is given of a phylogenetic network with each of these vertices. Notice that each edge is not drawn as a directed edge, that is because all edges are directed downwards. This will be the case for all phylogenetic networks in this thesis. The leaves of a network are often present-day species.

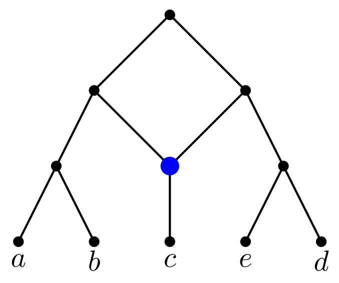


Figure 2: Example of a phylogenetic network. The reticulation is shown by a blue node.

Take $(a, b) = e \in E$, an edge in a phylogenetic network $\mathcal{N} = (V, E)$. Then e is an *outgoing* edge of a and an *incoming* edge of b. In addition, a is a *parent* of b and b is a *child* of a. It is possible for a vertex to have up to two children and it is also possible to have up to two parents, however, it is not allowed to have both two children and two parents. The incoming edges of a reticulation are called *reticulation edges.*

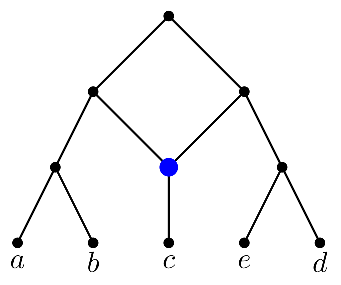
A tree is called a *rooted spanning tree* of $\mathcal{N}$ if it is a rooted subtree of $\mathcal{N}$ containing all vertices.

Now that the basics of phylogenetic networks have been established we can define the property which the algorithm is about: The tree-basedness of a phylogenetic network. Here follows a definition of tree-based networks.

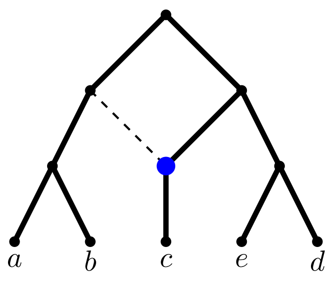
Definition 2.2. A phylogenetic network $\mathcal{N} = (V, E)$ on X is a *tree-based* network if there exists an $E' \subseteq E$ such that $\mathcal{N}' = (V, E')$ is a rooted spanning tree of $\mathcal{N}$ that has all its leaves in X.

Definition 2.3. If $\mathcal{N}$ is tree-based we call $\mathcal{N}'$ a base tree of $\mathcal{N}$.

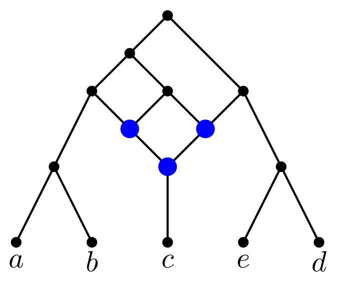
In Figure 3 two examples are given, the first example is a tree-based phylogenetic network and the other is an example of a non-tree-based phylogenetic network.



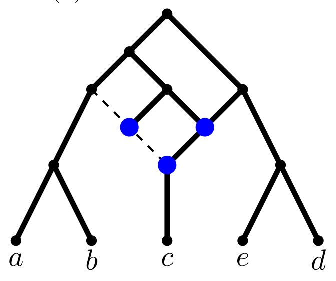
(a) Example of a tree-based network



(b) A rooted spanning tree of the network in (a)



(c) Example of a non-tree-based network



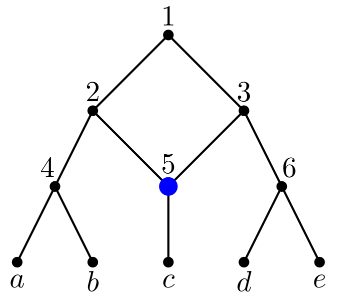
(d) A rooted spanning tree of the network in (c)

Figure 3: Example of a tree-based network and a non-tree-based network. The non-tree-based network in (c) has an extra leaf in the rooted spanning tree displayed in (d) (and it has at least one extra leaf in every other rooted spanning tree).

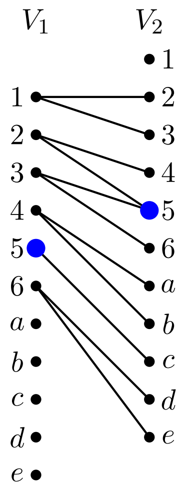
The algorithms that we implemented make use of auxiliary bipartite graphs of rooted binary phyologenetic networks. The first that is used is $\mathcal{G}_{\mathcal{N}}$ as defined by Francis, Semple and Steel [6]. The other one is $\mathcal{Z}_{\mathcal{N}}$ defined by Zhang [5]. First the definition of $\mathcal{G}_{\mathcal{N}}$ is given.

Definition 2.4. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Let V_1 and V_2 be exact copies of V. Then $\mathcal{G}_{\mathcal{N}}$ is the auxiliary bipartite graph with vertex bipartition $\{V_1, V_2\}$ and an edge between $u \in V_1$ and $v \in V_2$ when (u, v) is an edge in $\mathcal{N}$.

Figure 4 shows how $\mathcal{G}_{\mathcal{N}}$ is created from a phylogenetic network.



(a) Example of a tree-based network



(b) $\mathcal{G}_\mathcal{N}$ for network $\mathcal{N}$ in (a)

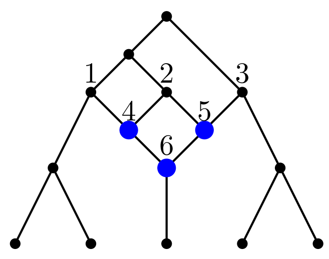
Figure 4: Example of a phylogenetic network and its auxiliary bipartite $\mathcal{G}_\mathcal{N}$ representation.

Bipartite graph $\mathcal{G}_\mathcal{N}$ as defined by Definition 2.4 provides us with a way to check the tree-basedness of a network. In Chapter 3 a characterization of tree-based networks will be given and proven based on $\mathcal{G}_\mathcal{N}$. The characterization was first proven in [6] and complements two previous characterizations of tree-based networks via matching which were proven in [2] and [5].

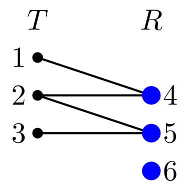
The following definition defines $\mathcal{Z}_\mathcal{N}$. This particular bipartite graph is used to calculate one of the characterizations in Chapter 3.

Definition 2.5. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Let R be the set of reticulations in $\mathcal{N}$ and T the set of tree vertices in $\mathcal{N}$ that are parents of reticulations. Let $\mathcal{Z}_\mathcal{N}$ be the auxiliary bipartite graph with vertex set $T \cup R$ and an edge between $t \in T$ and $r \in R$ when (t, r) is an edge in $\mathcal{N}$.

Figure 5 shows how an auxiliary bipartite graph $\mathcal{Z}_\mathcal{N}$ is created from the phylogenetic network in Figure 3c.



(a)



(b)

Figure 5: In (a) an example of a non-tree-based phylogenetic network is shown and its auxiliary bipartite graph $\mathcal{Z}_\mathcal{N}$ representation in (b). $T = \{1, 2, 3\}$ holds all the parents of reticulations that are tree vertices. $R = \{4, 5, 6\}$ holds all the reticulations.

A few characterizations of tree-based networks based on $\mathcal{Z}_\mathcal{N}$ have been defined by Zhang [5]. These definitions use a matching and a maximal path in $\mathcal{Z}_\mathcal{N}$. A matching in a graph is a set of edges without overlapping vertices. A maximal path is a path that cannot be extended to a longer path.

Theorem 2.1. Let $\mathcal{N}$ be a phylogenetic network. Then the following statements are equivalent:

i) $\mathcal{N}$ is tree-based.
ii) $\mathcal{Z}_{\mathcal{N}}$ has a matching such that each reticulation is matched.
iii) $\mathcal{Z}_{\mathcal{N}}$ has no maximal path that starts and ends with a reticulation.

In Figure 5 you can use Theorem 2.1 to check that the phylogenetic network is not tree-based, as $\mathcal{Z}_{\mathcal{N}}$ does not have a matching such that each reticulation is matched.

You can also conclude that $\mathcal{N}$ is not tree-based from the fact that vertex 6 by itself forms a maximal path that starts and ends at a reticulation.

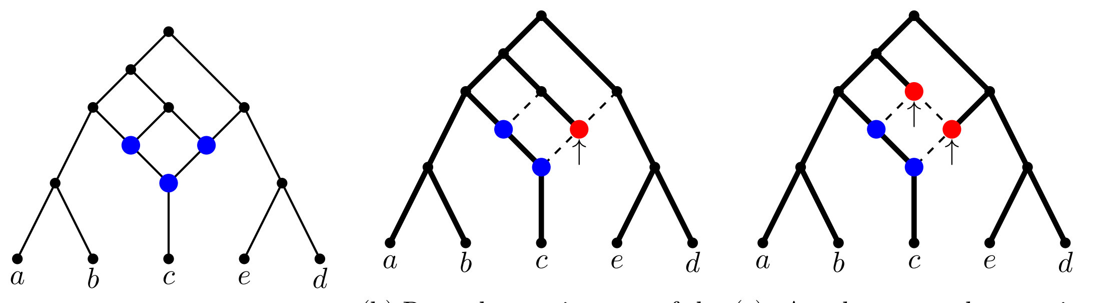
3 Measures for tree-basedness

A phylogenetic network $\mathcal{N}$ is either tree-based or it is not. This allows for measures to be zero if and only if $\mathcal{N}$ is tree-based. If these measures are not zero they show how close $\mathcal{N}$ is to being tree-based. In this section, all of the measures which are found using the algorithms are defined and a small example is given to show how to calculate them. Francis, Semple and Steel established three measures to check how close a phylogenetic network is to being tree-based. These are named $l(\mathcal{N})$, $p(\mathcal{N})$ and $t(\mathcal{N})$. The correctness of these measures will be proven later in this chapter.

First of all we define $l(\mathcal{N})$, which looks at the extra leaves of a rooted spanning tree of $\mathcal{N}$:

Definition 3.1. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Then $l(\mathcal{N})$ is equal to the minimum number of leaves in $V - X$ that must be present as leaves in a rooted spanning tree of $\mathcal{N}$.

Figure 6 shows an example of what $l(\mathcal{N})$ represents and how it is measured.



(a) Network

(b) Rooted spanning tree of the network in (a)

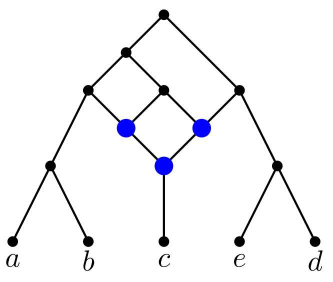
(c) Another rooted spanning tree of the network in (a)

Figure 6: An example of $l(\mathcal{N})$. In (b) a rooted spanning tree of the network in (a) is shown together with a leaf in $V - X$ marked by red. Another rooted spanning tree in (c) shows two leaves in $V - X$. The minimum number of leaves that must be present in $V - X$ is one. Thus $l(\mathcal{N}) = 1$ which indicates that the network is not tree-based.

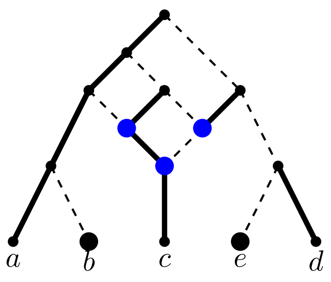
The second measure that is defined is $p(\mathcal{N})$. This measure is calculated by partitioning the network into vertex disjoint paths and using the number of paths. The definition of $p(\mathcal{N})$ is as follows:

Definition 3.2. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Then $p(\mathcal{N})$ is equal to the minimum number of vertex disjoint directed paths that partition the vertices of $\mathcal{N}$, minus $|X|$.

Figure 7 shows an example of what $p(\mathcal{N})$ represents and how it is measured.



(a) Network

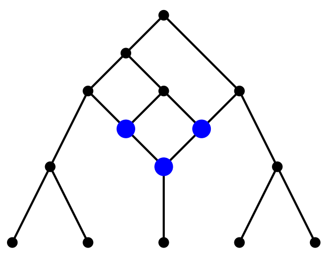


(b) Partition by vertex disjoint paths

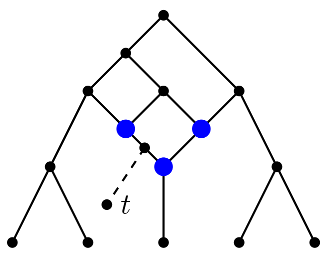
Figure 7: An example of $p(\mathcal{N})$. In (b) can be seen that the phylogenetic network of (a) can be partitioned by a minimum of six vertex disjoint paths, drawn in thicker black. The number of leaves that are present in the network is $|X| = 5$. So the end result $p(\mathcal{N}) = 6 - 5 = 1$ which indicates that the network is not tree-based.

The last measure to be defined is $t(\mathcal{N})$ which is determined by the number of leaves that need to be added to a network for it to be tree-based. Adding a leaf to a network is done by splitting an edge with a node and then attaching a leaf to that node. This will make sure the network stays binary.

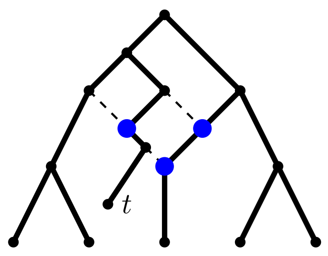
Definition 3.3. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Then $t(\mathcal{N})$ is equal to the minimum number of leaves that need to be added to $\mathcal{N}$ so the resulting network is tree-based.



(a) Network



(b) Added a single leaf node t



(c) A base tree of the network in (b)

Figure 8: An example of $t(\mathcal{N})$. From the examples in Figure 6 and 7 we know that the network of (a) is not tree-based. To determine the measure $t(\mathcal{N})$ leaves need to be added to the network until it is tree-based. To start off we add a node to a reticulation edge, then a single leaf t is added to the node. The network is now tree-based. This can be checked with the other measures or with the definition. This means $t(\mathcal{N}) = 1$.

The first theorem will prove that $p(\mathcal{N})$ is a correct measure to use when determining the tree-basedness of a network. The proof is also the algorithm for computing the measure. Before we prove the theorem we define $u(\mathcal{G}_\mathcal{N})$, a measure of $\mathcal{G}_\mathcal{N}$ as defined in Definition 2.4 and prove Lemma 3.1. Recall that, for a phylogenetic network $\mathcal{N} = (V, E)$ on X, the auxiliary bipartite graph $\mathcal{G}_\mathcal{N}$ has vertex bipartition $\{V_1, V_2\}$ and an edge between $u \in V_1$ and $v \in V_2$ when (u, v) is an edge in $\mathcal{N}$. $u(\mathcal{G}_\mathcal{N})$ equals the number of unmatched vertices of V_1 in a maximum matching in $\mathcal{G}_\mathcal{N}$.

Lemma 3.1. Let $\mathcal{N}$ be a phylogenetic network on X. Then

$$p(\mathcal{N}) = u(\mathcal{G}_\mathcal{N}) - |X|$$

Proof. First we show that $p(\mathcal{N}) \leq u(\mathcal{G}_{\mathcal{N}}) - |X|$. Let M be a matching in $\mathcal{G}_{\mathcal{N}}$ and U_2 the set of unmatched vertices in V_2. For each vertex $u \in U_2$, we recursively construct a directed path P_u in $\mathcal{N}$ as follows. Set $u = u_0$ and initially set $P_u = u_0$. If u_0 is unmatched in V_1, then stop the process and set $P_u = u_0$; else set $P_u = u_0u_1$, where $(u_0, u_1) \in M$. Continue this process with u_1. If u_1 is unmatched in V_1, then stop the process and set $P_u = u_0u_1$; else, when u_1 is matched, set $P_u = u_0u_1u_2$, where $(u_1, u_2) \in M$. Because $\mathcal{N}$ is acyclic, this process will eventually stop at the last vertex u_k which will be added to P_u as it is unmatched in V_1. We repeat this construction for all vertices in U_2. Creating a collection $\mathcal{P} = \{P_u : u \in U_2\}$ of directed paths in $\mathcal{N}$. Since M is a matching, the paths in $\mathcal{P}$ are disjoint. In addition, each vertex of $\mathcal{N}$ can be found in a path from $\mathcal{P}$. If not, suppose there is a vertex $v \in V$ that is not on such a path. Clearly, v is matched in V_2. But by reversing the construction above starting at v in V_2 we can see that v is on a path in $\mathcal{P}$. Since each vertex in X is unmatched in V_1 (no outgoing edges), and because the number of paths in $\mathcal{P}$ is equal to the number of unmatched vertices in V_1 and in V_2, it follows by choosing the maximum matching M that

$$p(\mathcal{N}) \leq |\mathcal{P}| - |X| = u(\mathcal{G}_{\mathcal{N}}) - |X|$$

Now to prove that $p(\mathcal{N}) \geq u(\mathcal{G}_{\mathcal{N}}) - |X|$. Let $\mathcal{P}$ be a set of disjoint paths that partition the vertices of $\mathcal{N}$ and let M be a matching of $\mathcal{G}_{\mathcal{N}}$ obtained in the following way: $M = \{(u, v) : u$ and v are consecutive vertices on a path in $\mathcal{P}\}$. The paths in $\mathcal{P}$ are disjoint so M is indeed a matching. Because every vertex of $\mathcal{N}$ is on a path in $\mathcal{P}$, the number u_1 of unmatched vertices in V_1 is equal to the number of paths in $\mathcal{P}$. Thus by choosing $\mathcal{P}$ to be of minimum size

$$p(\mathcal{N}) = |\mathcal{P}| - |X| = u_1 - |X| \geq \mathcal{G}_{\mathcal{N}} - |X|$$

□

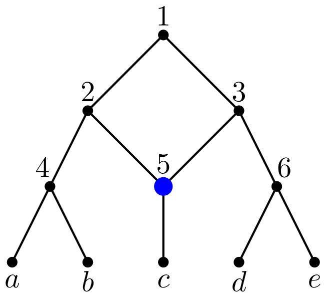
The next theorem, which shows that $p(\mathcal{N}) = 0$ is a correct measure for tree-basedness, uses Lemma 3.1 in its proof.

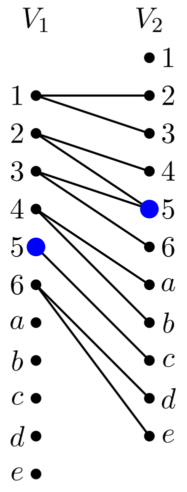
Theorem 3.1. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Then the following statements are equivalent:

i) $\mathcal{N}$ is tree-based.
ii) $\mathcal{G}_{\mathcal{N}}$ has a matching of size $|V| - |X|$.

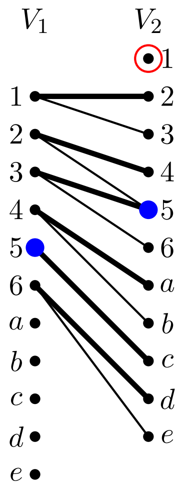
Proof. Each of the leaves in the network are unmatched in V_1, because of this it follows that $\mathcal{G}_{\mathcal{N}}$ has a matching of size $|V| - |X|$ if and only if $\mathcal{G}_{\mathcal{N}}$ has a maximum-sized matching of this size. With use of Lemma 3.1 we can show that $\mathcal{G}_{\mathcal{N}}$ has a maximum-sized matching of $|V| - |X|$ if and only if $p(\mathcal{N}) = u(\mathcal{G}_{\mathcal{N}}) - |X| = 0$. The result follows from $p(\mathcal{N}) = 0$ if and only if $\mathcal{N}$ is tree-based. □

In the following set of figures the algorithm from the proof is used step by step to show how it works.

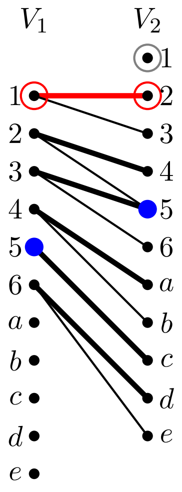

(a) Example of a tree-based network



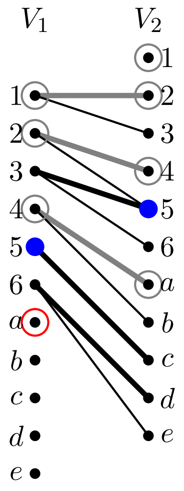
(b) $\mathcal{G}_{\mathcal{N}}$ for network $\mathcal{N}$ in (a)



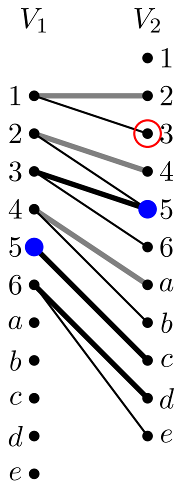
(c) Start at the first unmatched vertex in V_2, Vertex 1. The matching is displayed with a thicker line.



(d) Vertex 1 is unmatched in V_1 and connected to vertex 2. Set $P_1 = (1\ 2)$

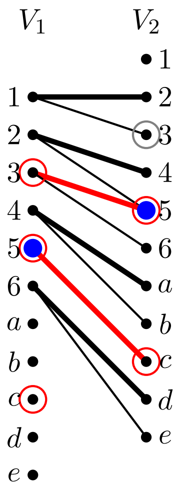


(e) Repeat the process for vertex 2 and 4. Set $P_1 = (1\ 2\ 4\ a)$. a is unmatched in V_1 so the process is terminated.

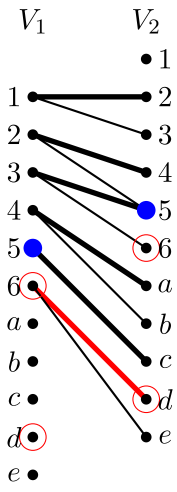


(f) The next unmatched vertex is vertex 3.

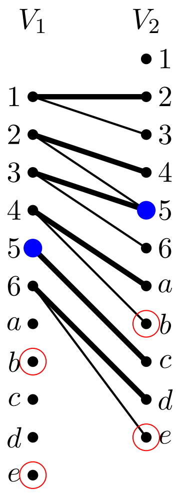
Figure 9: Example of a phylogenetic network and its auxiliary bipartite $\mathcal{G}_{\mathcal{N}}$ representation.



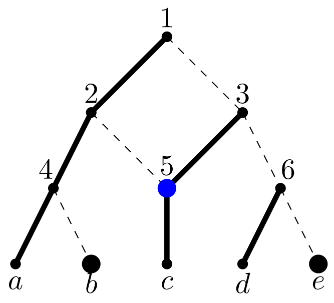
(a) The next unmatched vertex is vertex 3. Repeat the previous steps. 3 is matched to 5. 5 is matched to c and c is unmatched. $P_3 = (3\ 5\ c)$.



(b) The next unmatched vertex is vertex 6. 6 is matched to d and d is unmatched. $P_6 = (6\ d)$.



(c) The next unmatched vertices are b and e. However they are also unmatched in V_1 resulting in $P_b = (b)$ and $P_e = (e)$



(d) Showing disjoint paths in $\mathcal{P} = \{P_1, P_3, P_6, P_b, P_e\}$. The number of disjoint paths is 5, the number of leaves is also 5. So we see that $p(\mathcal{N}) = 5 - 5 = 0$.

Figure 10: Continuation of the example in Figure 9

The following measure $l(\mathcal{N})$ is also well defined as it is identical to $p(\mathcal{N})$.

Theorem 3.2. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Then

$$l(\mathcal{N}) = p(\mathcal{N})$$

To show that the last measure $t(\mathcal{N})$ is also well defined we use the following theorem.

Theorem 3.3. Let $\mathcal{N} = (V, E)$ be a phylogenetic network on X. Let M be a maximum sized matching on $\mathcal{Z}_{\mathcal{N}}$. Then $t(\mathcal{N})$ is the number of unmatched reticulations in $\mathcal{Z}_{\mathcal{N}}$.

These two theorems have been proven in [6].

4 Algorithms

In this chapter a pseudocode will be given which shows how the algorithm has been programmed. The measure that will be computed first is $p(\mathcal{N})$ in Section 4.1. The second measure $l(\mathcal{N})$ will be computed in 4.2 and will also deliver a rooted spanning tree of the network which uses the vertex disjoint paths created by the algorithm in Section 4.1. Measure $t(\mathcal{N})$ is computed in section 4.3. . The programming language that is used is Java. A separate piece of pseudocode will be given and explained for each of the functions that are used.

The first piece of the algorithm is the main function. The input for the main function can be a text file containing a newick string or an edge set. The main function is used to read the text file and call the algorithms used for the other sections. To start of the algorithm a valid text file needs to be provided. This text file will be converted to an iterable data structure. The algorithm to convert a given newick string to an iterable data structure has been written by Leo van Iersel. The algorithm that converts a given edge set has been written by Arthur Mooiman. With the conversion algorithms we will end up with a single root node which holds information about its children and several other properties. These children once again have information about their children and their parents.

After the conversion the work is split up into three other functions. First will be vertexDisjointPaths() to calculate the maximum number of vertex disjoint paths in the network and thus also $p(\mathcal{N})$. This function can be found in section 4.2. The second function is rootedSpanningTree(). This function creates a rooted spanning tree $\mathcal{T}$ of the network with a minimum number of dummy leaves, see Section 4.2. The last function treeBasedNetwork() constructs a tree-based network $\mathcal{N}'$ by adding $t(\mathcal{N})$ leaves to the network $\mathcal{N}$ if necessary.

When each of the different functions is completed the created networks are converted to DOT format. The original network will be in the text file “DotNetwork.txt”. The rooted spanning tree will be in the text file “DotSpanningTree”. The tree-based version will be in the text file “DotTreeBasedVersion”.

Algorithm 1 Main function

```
Require: newick string or edge set in a text file
  $n \leftarrow$ Read the text file.
  Root node $N \leftarrow$ An iterable data structure of the network.
  Convert the network to DOT format.
  $p(\mathcal{N}) \leftarrow$ The number of vertex disjoint paths in the network. (vertexDisjointPaths())
  print $p(\mathcal{N})$
  Create a rooted spanning tree of the given network. (rootedSpanningTree())
  Convert the created rooted spanning tree to DOT format.
  Create a tree-based network of the given network. (treeBasedNetwork())
  $t(\mathcal{N}) \leftarrow$ The number of leaves added by treeBasedNetwork().
  Convert the created tree-based network to DOT format.
```

4.1 Vertex disjoint paths

The function which calculates the measure $p(\mathcal{N})$ will be *vertexDisjointPaths*. The function follows the proof of Lemma 3.1. Following the proof a few things need to be done:

- Construct bipartite graph $\mathcal{G}_{\mathcal{N}}$ by *network2BipartiteGn*
- Find a maximum matching on $\mathcal{G}_{\mathcal{N}}$ by *findMaxMatching*
- Find all unmatched vertices of V_2 by *findUnmatchedV2*
- Find all disjoint paths starting at an unmatched V_2 node by *findUniqueMaxSequences*.

After these steps we can calculate $p(\mathcal{N})$ by substracting the number of leaves from the number of disjoint paths.

Algorithm 2 vertexDisjointPaths

```
Require: Network $N$
  Vertices ← All vertices of the network.
  Edges ← All edges of the network.
  Gn ← The bipartite graph $\mathcal{G}_{\mathcal{N}}$ of your network. (network2BipartiteGn)
  Find the maximum matching on $\mathcal{G}_{\mathcal{N}}$. (findMaxMatching)
  unmatchedV2 ← Unmatched vertices of Gn.
  disjointPaths ← All disjoint paths in Gn starting at a node in unmatchedV2. (findUnique-
  MaxSequences)
  numOfDisjointPaths ← The number of elements in disjointPaths.
  numOfLeaves ← The number of leaves in the network.
  $p(\mathcal{N})$ ← numOfDisjointPaths - numOfLeaves
  return $p(\mathcal{N})$
```

The following algorithm, *network2BipartiteGn*, constructs a bipartite graph $\mathcal{G}_{\mathcal{N}}$ simply by going over each edge in the network. This function requires the value $num[0]$, the highest id number of a vertex. For each edge it puts the first vertex in V_1 and the second vertex in V_2, where V_1 and V_2 are part of $\mathcal{G}_{\mathcal{N}}$. It tracks if a vertex has been visited already to prevent duplicating vertices. This algorithm also creates an initial matching in $\mathcal{G}_{\mathcal{N}}$ while it is constructing which will be used in a later algorithm. This matching is not necessarily a maximum matching. At the end of the algorithm you are left with bipartite graph $\mathcal{G}_{\mathcal{N}}$ and a matching.

Algorithm 3 network2BipartiteGn

```
Require: *edges*, Boolean vector to track visited vertices
  Create Gn with $V1$ and $V2$.
  for each edge in *edges* do
    startNode ← The first node in edge.
    endNode ← The last node in edge.
    if startNode has not been visited then
      startNode is visited
      Add startNode to both V1 and V2
    else if startNode has been visited then
      startNode ← startNode from V1
    end if
    if endNode has not been visited then
      endNode is visited
      Add endNode to both V1 and V2
    else if endNode has been visited then
      endNode ← endNode from V2
    end if
    Make startNode a neighbour of EndNode and vice versa.
    if Both startNode and endNode are unmatched then
      Match startNode to endNode
    end if
  end for
  return Gn, matching M
```

To continue the algorithm we have the function *findMaxMatching*. This function finds the maximum matching in a bipartite graph by using Berge's lemma [8]. Berge's lemma states that a matching M in a graph is maximum if and only if there is no augmenting path with M. An augmenting path is a path where the start and end vertices are unmatched in M and the path is alternating edges between in and not in M. The function has to find an augmenting path and change the matching when it finds one. It will repeat this process until it can no longer find an augmenting path, thus leaving a maximum matching.

Algorithm 4 findMaxMatching

```
Require: Gn, matching M
  while There is an augmenting path in Gn (hasAugmentingPath) do
    for Each edge in the augmenting path do
      if Edge in matching then
        Remove edge from matching
      else if Edge not in matching then
        Add edge to matching
      end if
    end for
  end while
```

To check whether there is an augmenting path present the following function will return either true or false. It will also set up the path for *findMaxMatching* to continue.

Algorithm 5 hasAugmentingPath

```
Require: Gn, matching M
  edgeTo ← Vector to track paths.
  unmatchedV1 ← All unmatched vertices in $V_1$
  while unmatchedV1 is not empty do
    s ← Remove a vertex from unmatchedV1
    for Each neighbour t of s do
      if Edge (s,t) is forward edge not in matching or backward edge in matching AND t has
      not been visited then
        t is visited
        if t is not matched then
          return True and return the path
        end if
      end if
    end for
  end while
  return False
```

With the maximum matching found we can continue following the proof of Theorem 3.1. Let *unmatchedV2* be the set of unmatched vertices in V_2.

The final part of the algorithm is to find the disjoint vertex paths using *findUniqueMaxSequences*. A path is created for each vertex in *unmatchedV2*. Then we take the same vertex but from V1 and check if it is matched. If it is matched then add that vertex to the current path and repeat the process. Eventually it will stop at a vertex which is unmatched in V1. Because we use a maximum matching all of the found paths will be vertex disjoint. After this function we have a list full of vertex disjoint paths.

Algorithm 6 findUniqueMaxSequences

```
Require: Gn, unmatchedV2
  paths ← Empty list of lists
  for Each vertex nv2 in unmatchedV2 do
    p ← empty list
    Add nv2 to p
    nv1 ← nv2 vertex from V1
    while nv1 is matched do
      Add the matched vertex to p
      nv1 ← matched vertex from V1
    end while
    Add p to paths
  end for
  return paths
```

With the last part of the algorithm completed the only thing left to do is to calculate $p(\mathcal{N})$ which happens in the first function.

4.2 Rooted Spanning Tree

The function to construct the rooted spanning tree $\mathcal{N}'$ of the network $\mathcal{N}$ will be *rootedSpanningTree*. This function uses the vertex disjoint paths found by the function in Section 4.1. Each of the paths in vertex disjoint paths except the path that starts at the root will be extended. A path in vertex disjoint paths is of the form $\pi = v_0 v_1 \ldots v_n$. This path will be extended to $\pi' = w v_0 v_1 \ldots v_n$, where (w, v_0) is an edge in $\mathcal{N}$. After each path is extended they will be merged together to form the rooted spanning tree $\mathcal{T}$.

The *rootedSpanningTree* function will split the work up into three parts:

- Find the path in vertex disjoint paths which traverses the root.
- Extend the other paths in vertex disjoint paths.
- Combine the root path and the extended paths to create a rooted spanning tree.

Algorithm 7 rootedSpanningTree

```
Require: disjointPaths
  rootpath ← The path in disjointPaths which traverses the root.
  Extend the remaining paths in disjointPaths.
  spanningTree ← The combined rootpath and extended paths. (createSpanningTree())
```

To merge the rootpath and the extended paths we use *createSpanningTree*. In this function we create a new vertex for each vertex in rootpath and give them children and parents as they are ordered. For the extended paths we need to check if a vertex has been created already before adding new ones. Give each of these vertices children and parents as they are ordered. After every extended path has been added, the leftover network will be a rooted spanning tree of $\mathcal{N}$. The rooted spanning tree will be returned in DOT format in a text file named “DotSpanningTree.txt” .

Algorithm 8 createSpanningTree

```
Require: rootPath, extended disjointPaths.
  allST ← The vector containing all created vertices.
  for each vertex in rootpath do
    Create a new vertex and add it to allST.
    Give each vertex children and parents as they are ordered.
    Label the root vertex as root.
  end for
  for each path in extended disjointPaths do
    Create a new vertex for each vertex in path unless that vertex already exists.
    Add children and parents to those vertices as they are ordered.
  end for
  return The root of the rooted spanning tree.
```

4.3 Tree-Based Network

In this section the function *treeBasedNetwork* is explained. The steps taken by this function coincide with theorem 3.3. The function will provide a tree-based version of the given network that is closest to the original by adding $t(\mathcal{N})$ leaves to the network.

To construct the tree-based network the following things need to be done:

- Construct $\mathcal{Z}_{\mathcal{N}}$ as defined in definition 2.5.
- Find the maximum matching in $\mathcal{Z}_{\mathcal{N}}$.
- Find all unmatching reticulations in $\mathcal{Z}_{\mathcal{N}}$.
- For each of those reticulations we remove one of their parents in the original network and replace that parent with a dummy vertex. This dummy vertex has two children, the reticulation and a new vertex as a leaf. The dummy vertex has the original parent as a parent.

After all these steps the network is still binary and is tree-based. The function *treeBased-Network* breaks the work up into these four steps.

Algorithm 9 treeBasedNetwork

```
Require: Root vertex N.
  rets ← All reticulations of $\mathcal{N}$.
  Zn ← Bipartite graph $\mathcal{Z}_{\mathcal{N}}$. (network2BipartiteZN())
  Find the maximum matching in $\mathcal{Z}_{\mathcal{N}}$. (findMaxMatching(Zn))
  unmatchedRets ← All unmatched rets in $\mathcal{Z}_{\mathcal{N}}$.
  Attach leaves to each unmatched reticulation. (attachLeaves())
```

First of all it finds all reticulations that are present in the network. When those are found it moves on to *network2BipartiteZN* to create the bipartite graph $\mathcal{Z}_{\mathcal{N}}$. Each of the reticulations of the network are put into V_1. The parents of these reticulations are put into V_2 if its a tree vertex. While doing this, create an edge between the reticulation and the parent. If both of these are unmatched in $\mathcal{Z}_{\mathcal{N}}$, match them to each other. In the end you will end up with $\mathcal{Z}_{\mathcal{N}}$ and a matching.

Algorithm 10 network2BipartiteZN

```
Require: Each reticulation in the network.
  for each reticulation do
    Put the reticulation in $V_1$.
    Put its parents in $V_2$ if it is a tree vertex.
    Create an edge between the reticulation in $V_1$ and its parents in $V_2$.
    if both are unmatched then
      match both to each other.
    end if
  end for
  return $\mathcal{Z}_{\mathcal{N}}$.
```

For the next step it is required to find the maximum matching in $\mathcal{Z}_{\mathcal{N}}$. To do this we reuse the function4.

With a maximum matching the following step is to find the unmatched reticulations in $\mathcal{Z}_{\mathcal{N}}$. For each of these reticulations we need to add some dummy vertices to the network to make it tree-based. The function to do this will be *attachLeaves*, which will also return $t(\mathcal{N})$, the number of added leaves, and the tree-based network.

Algorithm 11 attachLeaves()

```
Require: Root vertex N, unmatched reticulations in $\mathcal{Z}_{\mathcal{N}}$.
  $t \leftarrow 0$.
  for Each unmatched reticulation do
    $t = t + 1$.
    Get one of the parents from this reticulation.
    Change the child of this parent and the parent of the reticulation to a new vertex.
    Add to the new vertex another new vertex as a child and leaf.
  end for
  return $t$
```

When the function is finished you are left with $t(\mathcal{N})$ and a tree-based version of your network.

5 Examples

This section holds different examples to show what is needed for the algorithm to work and the various outputs it produces. Each example will start with a newick string and will output the measures $p(\mathcal{N})$, $l(\mathcal{N})$ and $t(\mathcal{N})$. Alongside these measures there will be three networks in DOT format: The original network, the rooted spanning tree of the network and a tree-based network obtained by adding a minimum number of leaves to the original network.

5.1 Example 1

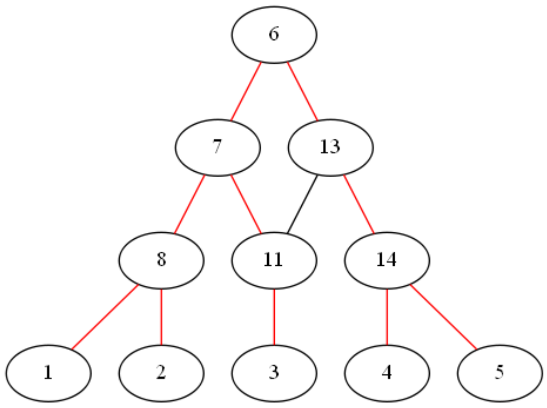
The first example will be the tree-based network used throughout the thesis. The newick string for this network is:

```
(((a,b),((c)#H1)),((d,e),#H1));
```

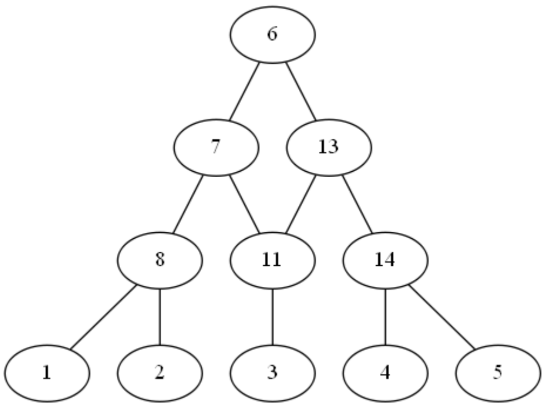
When the algorithm is run on this newick string it will output the following:

- $p(\mathcal{N}) = 0$

- $l(\mathcal{N}) = 0$
- $t(\mathcal{N}) = 0$



(a) Network with rooted spanning tree in red.



(b) Tree-based version.

As expected the results for all three measures is 0.

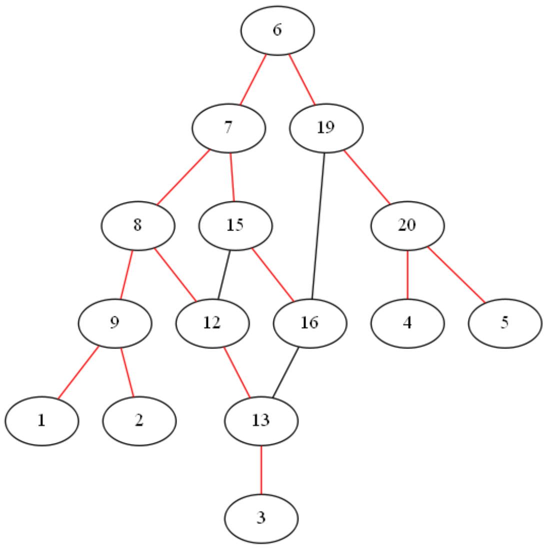
5.2 Example 2

The second example will be the non-tree-based network used throughout the thesis. The newick string for this network is:

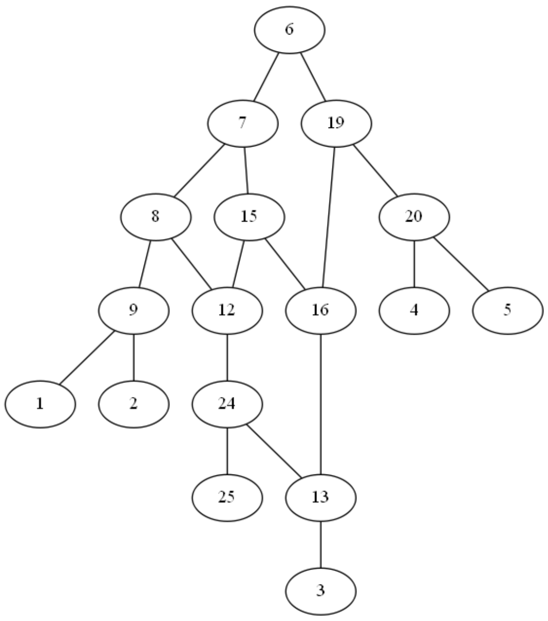
```
((((a,b),(((c)#H1)#H2)),(((#H1)#H3),#H2)),((d,e),#H3));
```

When the algorithm is run on this newick string it will output the following:

- $p(\mathcal{N}) = 1$
- $l(\mathcal{N}) = 1$
- $t(\mathcal{N}) = 1$



(a) Network with rooted spanning tree in red.



(b) Tree-based version.

5.3 Example 3

The following example is of a network with around 50 leaves and 104 reticulations. The network can be found on `http://phylnet.univ-mlv.fr/recophync/networkDraw.php` and is the network already filled in in the "Draw this network" window.

When the algorithm is run on this newick string it will output the following:

- $p(\mathcal{N}) = 11$
- $l(\mathcal{N}) = 11$
- $t(\mathcal{N}) = 11$

5.4 Example 4

The following examples are from existing literature and can all be found on `http://phylnet.univ-mlv.fr/r`

The first of these examples is from [9] figure 2. This network is an example ancestral recombination graph for variation at the yeast gene encoding and flanking sequence. It has two reticulations and 10 leaves. Each reticulation represents a recombination event. The newick string of this network is :

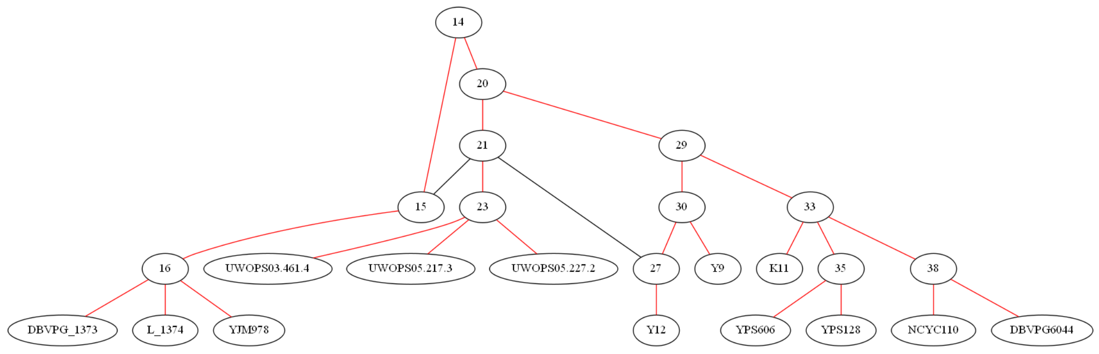
```
((DBVPG\_1373,L\_1374,YJM978)\#H1,((\#H1,(UWOPS03.461.4,UWOPS05.217.3,UWOPS05.227.2),
(Y12)\#H2),((\#H2,Y9),((K11,(YPS606,YPS128),(NCYC110,DBVPG6044))))))
```

One can also use an edge set to get the same results. Below are the first few edges of the network:

```
r 94864
r b
b c
c 94864
```

When the algorithm is run on this network it will output the following:

- $p(\mathcal{N}) = 0$
- $l(\mathcal{N}) = 0$
- $t(\mathcal{N}) = 0$



(a) Network with rooted spanning tree in red.

5.5 Example 5

A larger example which also produces measures which are greater than one is figure 3 from [10]. This example has 31 reticulations and 7 leaves. The newick string or the edge set are simply too big to place in this thesis. The same holds true for the figure produced by the algorithm.

When the algorithm is run on this network it will output the following:

- $p(\mathcal{N}) = 3$
- $l(\mathcal{N}) = 3$
- $t(\mathcal{N}) = 3$

5.6 Example 6

The following example is figure 7 of [11]. This network has 5 reticulations and 29 leaves. The network represents the five largest highways in fungi. Each reticulations represents gene transfer between the fungi.

When the algorithm is run on this network it will output the following:

- $p(\mathcal{N}) = 0$
- $l(\mathcal{N}) = 0$
- $t(\mathcal{N}) = 0$

6 Discussion

In this thesis the algorithms have been programmed to determine the tree-basedness of a binary phylogenetic network. For these algorithms we used several theorems which characterize when a network is tree-based or not. Each of these theorems are proven specifically for binary networks. And thus the code written is only proven correct for binary phylogenetic networks.

To expand further one could define each of the measures for non-binary phylogenetic networks. This would allow vertices to have an in-degree or out-degree greater than two. The code could then be expanded to cover more diverse networks.

Another item that can be expanded upon is the base tree that the algorithm produces. The algorithm provides only one base tree for a given network while a network could have several other base trees. One could count all the different base trees of a phylogenetic network.

To expand further on the topic of base trees one could provide a network $\mathcal{N}$ and a tree T in $\mathcal{N}$ and calculate how close T is to being a base tree of $\mathcal{N}$.

References

[1] Andrew R. Francis and Mike Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic biology 64.5*, pages 768–777, 2015.

[2] Laura Jetten and Leo van Iersel. Nonbinary tree-based phylogenetic networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 2016. DOI: 10.1109/TCBB.2016.2615918.

[3] Maria Anaya, Olga Anipchenko-Ulaj, Aisha Ashfaq, Joyce Chiu, Mahedi Kaiser, Max Shoji Ohsawa, Megan Owen, Ella Pavlechko, Katherine St John, Shivam Suleria, et al. On determining if tree-based networks contain fixed trees. *Bulletin of mathematical biology*, 78(5):961–969, 2016.

[4] Charles Semple. Phylogenetic networks with every embedded phylogenetic tree a base tree. *Bulletin of mathematical biology*, 78(1):132–137, 2016.

[5] Louxin Zhang. On tree-based phylogenetic networks. *Journal of Computational Biology*, 23(7):553–565, 2016.

[6] Andrew Francis, Charles Semple, and Mike Steel. New characterisations of tree-based networks and proximity measures. *arXiv preprint arXiv:1611.04225*, 2016.

[7] Thomas Marcussen, Lise Heier, Anne K Brysting, Bengt Oxelman, and Kjetill S Jakobsen. From gene trees to a dated allopolyploid network: insights from the angiosperm genus viola (violaceae). *Systematic biology*, 64(1):84–101, 2014.

[8] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957. National Acad Sciences.

[9] Paul A Jenkins, Yun S Song, and R. B. Brem. Genealogy-based methods for inference of historical recombination and gene flow and their application in saccharomyces cerevisiae. *PLOS ONE*, 7(11):e46947, 2012. Public Library of Science.

[10] Nikki D Charlton, Ignazio Carbone, Stellos M Tavantzis, and M. A. Cubeta. Phylogenetic relatedness of the m2 double-stranded rna in rhizoctonia fungi. *Mycologia*, 100(4):555–564, 2008. Taylor & Francis.

[11] Gergely J Szöllősi, Adrián Arellano Davín, Eric Tannier, Vincent Daubin, and Bastien Boussau. szetal. *Phil. Trans. R. Soc. B*, 370(1678):20140335, 2015. The Royal Society.