Triplet Encoding of Stemmata

by

T.M.A. Levert

to obtain the degree of Master of Science,
as part of the master's program Applied Mathematics
specialising in Discrete Mathematics and Optimisation
at the faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS),
at Delft University of Technology,
to be publicly defended on Monday, July 21, 2025, at 13:00

Student number:	5086191	
Thesis committee:	Dr. Y. Murakami (Supervisor)	TU Delft
	Dr.ir. L.J.J. van Iersel (Supervisor)	TU Delft
	Dr.ir. J. Bierkens	TU Delft

Delft, July 3, 2025

Abstract

Stemmatology is the study and reconstruction of textual genealogy and has several similarities to phylogenetics, the study of evolutionary histories of species. Current methods in computational stemmatology often borrow tools from phylogenetics, yet classical phylogenetic models are not well suited to the structural and labelling requirements of manuscript traditions. In particular, phylogenetics typically assumes leaf-labelled trees or networks and lacks the means to accommodate internal labels—a feature which is crucial in stemmatology. Nonetheless, these models are frequently used as there are few formal alternatives.

This thesis addresses that gap by studying the encoding and reconstruction of internally labelled trees and networks from rooted triplets. Specifically, we consider three classes of graphs: multifurcating rooted trees, general rooted trees (allowing nodes with out-degree one), and level-1 networks. Each graph is assumed to have labels on a subset of its vertices, including all leaves and some internal nodes. We prove that, under limited assumptions, a complete set of rooted triplets uniquely determines the structure and labelling of each of these graph classes up to isomorphism. This generalises earlier results for leaf-labelled binary trees and extends triplet-based encoding to structures with internal labelling and limited reticulation.

Building on these encoding theorems, we develop polynomial-time algorithms to reconstruct each graph class from its full triplet set. For trees, our methods generalise previously proposed algorithms by allowing multifurcations, nodes with out-degree one, and labels at internal vertices. For level-1 networks, we design a reconstruction algorithm that correctly identifies cycle structures and label placement, adapting earlier techniques for dense triplet sets. All reconstruction algorithms are proven correct and theoretically efficient under the given assumptions. The algorithms have been tested on many instances and run quickly.

To compare internally labelled graphs, we introduce an extension of the classical triplet distance. This adapted metric counts differences in triplet sets between two graphs on the same label set. We evaluate the metric and reconstruction algorithms on synthetic and real-world data, demonstrating their ability to capture meaningful structural differences and to recover known graphs from complete or near-complete triplet information.

These results show that rooted triplets form a robust foundation for reasoning about internally labelled structures in both tree-like and mildly reticulate settings. The theory and algorithms presented in this thesis provide new tools for computational phylogenetics and stemmatology, enabling the reconstruction and comparison of complex transmission histories from local relational constraints.

Acknowledgments

Writing this thesis has been both a challenging and rewarding journey, and I owe a great deal of gratitude to the people who supported me along the way.

First and foremost, I want to thank my supervisor, Yuki Murakami, for all the time and effort he put into our weekly meetings. Thanks to his patience and guidance, I was able to keep going even after trying dozens of ways to make some of the proofs work. His sharp insights and careful attention to both detail and structure helped shape this thesis in ways I could not have done alone. Our meetings consistently took way longer than the planned hour, but therefore allowed us to come up with possible new approaches to the many obstacles that I encountered along the way. Thank you for our fun times and good conversations.

I am also very grateful to Leo van Iersel and Joris Bierkens for joining my thesis committee, assessing this thesis, and attending my defence.

To my fellow students and friends from EWI (and even TN), thank you for the wonderful times and the much-needed distractions. Throughout the many exam weeks, late nights, and long lectures, you made it so much more fun.

A special thank you to all my "clubgenoten" with whom I have grown very close in these last five years together. Without all of you, I cannot imagine having had such a great time as I did now. I am looking forward to our big trip together to Guatemala and Belize!

I am especially thankful to my family, who were always there to listen to my struggles, nod even though they had no clue what I was talking about, and assure me everything would work out in the end. Your patience, love, support, and willingness to listen meant more to me than I can say.

To everyone who helped me along the way, thank you!

Contents

Abstract **ii**

Acknowledgments **iii**

1 Introduction **1**
1.1 Previous work 3
1.2 Our contributions 4

2 Preliminaries **5**
2.1 Graphs 5
2.2 Triplets 6
2.3 Trees 6
2.4 Networks 7
2.5 Metric 8

3 Trees **9**
3.1 Multifurcating tree with unlabelled root 9
3.1.1 Unique triplet encoding 11
3.1.2 Algorithm 13
3.2 General trees 17
3.2.1 Unique triplet encoding 19
3.2.2 Algorithm 21

4 Level-1 Networks **26**
4.1 Unique triplet encoding 27
4.2 Algorithm 38

5 Computational Performance **42**
5.1 Dataset Creation 42
5.2 Multifurcating Tree Reconstruction 44
5.3 General Tree Reconstruction 46
5.4 Level-1 Network Reconstruction 48

6 Triplet Distance Metric **51**
6.1 SPR Moves 51
6.2 Triplet Distance Comparison on Real Datasets 54

7 Conclusion **59**
7.1 Summary of results 59
7.2 Discussion 60
7.3 Future work 60

A Appendix **64**
A.1 Trees 64
A.1.1 Multifurcating trees 64
A.1.2 General trees 67
A.2 Networks 71

1 Introduction

Stemmatology is the discipline concerned with reconstructing the historical relationships between different versions of a text. At its core lies the question of how a set of manuscript witnesses[1], each containing a version of a text, relates to one another in terms of copying, corruption, and shared ancestry. These relationships are traditionally modelled using tree-like structures, known as stemmata, which encode the inferred paths of transmission. Such a stemma can be seen in Figure 1.1. It relies on the idea that if two texts share a distinctive error, it suggests that they descend from a common ancestor. Over the past decades, researchers have increasingly drawn connections between stemmatology and computational phylogenetics, a subfield of bioinformatics concerned with reconstructing evolutionary trees from molecular data. The analogy is compelling: just as species evolve thanks to occasional mutations, so too do manuscripts develop via chains of copying, where errors, emendations, and variants accumulate and propagate (Finlay, 2023).

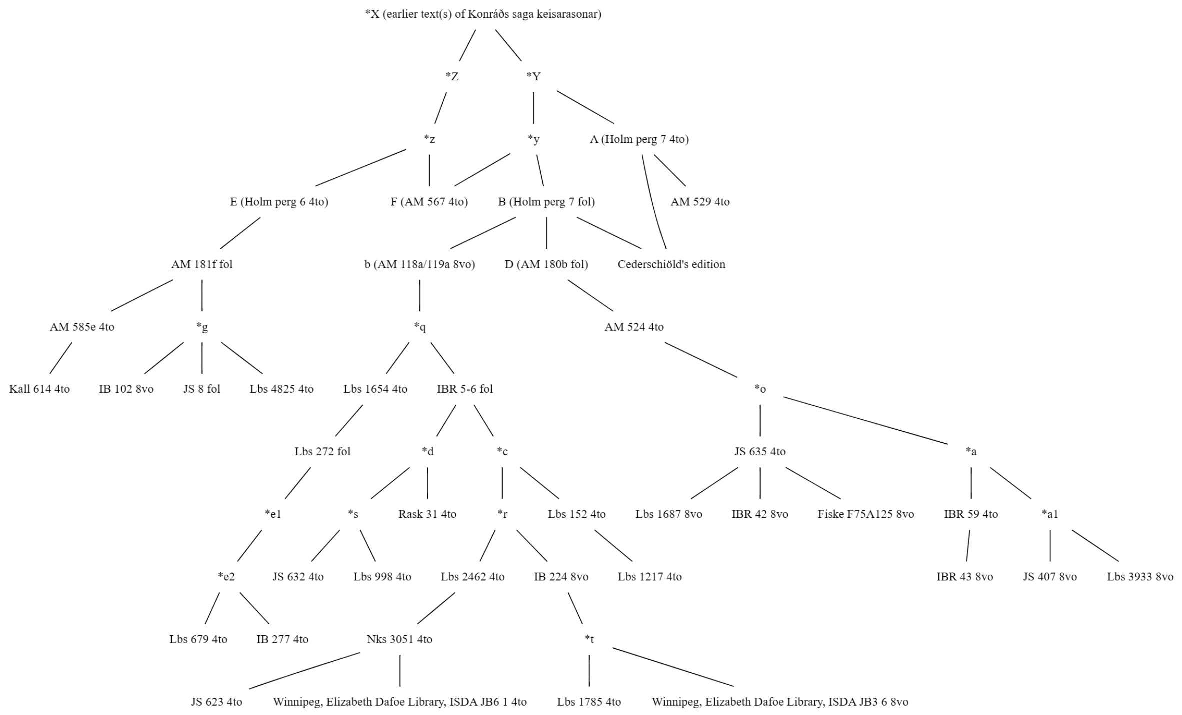


Figure 1.1: A stemma for "Konráðs saga keisarasonar" proposed by Hall and Parsons (2013). Several children and their descendants of "E (Holm perg 6 4to)" have been removed for compactness. $*X$ represents the unknown original manuscript (archetype). Extant manuscripts are shown in upper-case letters or names, and non-extant manuscripts are shown as an asterisk followed by a lower-case letter.

[1]A witness is an existing instance of a text. They are represented by labelled nodes in a graph.

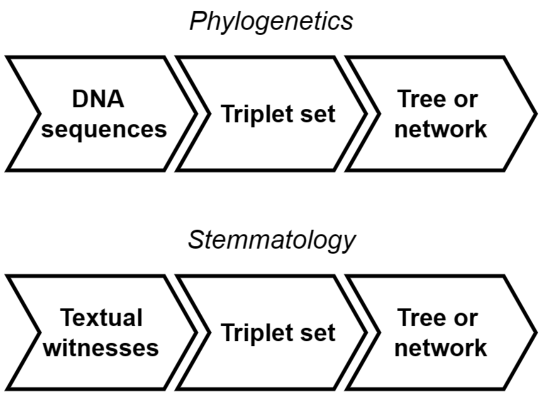


Figure 1.2: A process of creating phylogenetic trees or networks and stemmata based on triplets. In phylogenetics, the triplet sets are based on DNA sequences, while in stemmatology, they are based on textual witnesses.

When looking at the processes of creating the graphs for both phylogenetics and stemmatology based on the triplet reconstruction technique, the similarity is even more striking. Figure 1.2 shows these heavily simplified processes. While in phylogenetics the triplets are based on DNA sequences, in stemmatology they are based on extant textual witnesses. In both fields, these triplets can then be used to produce the networks or trees. The assumptions made to go from triplet sets to networks, however, differ between phylogenetics and stemmatology.

The similarities that do exist in obtaining networks from triplets have enabled scholars to apply techniques originally developed for modelling biological evolution to the analysis of textual traditions. For example, phylogenetic algorithms have been used to infer the genealogy of manuscripts in works such as Chaucer's Canterbury Tales and biblical texts, as shown by studies like (Barbrook et al., 1998) and more recently (Zammit, 2024). Such applications typically represent manuscripts as leaves in a tree, with internal nodes representing hypothetical ancestors or evolutionary events such as speciation. The relationships are inferred based on patterns of shared variants, similar to mutations in a DNA sequence, that suggest common ancestry. As the field has evolved, so too has the recognition that a purely tree-like model may not be sufficient. Just as horizontal gene transfer and hybridisation events motivate the use of phylogenetic networks in biology, textual contamination — where a scribe consults multiple sources — requires more flexible models that include reticulations. Reticulations are nodes within a network that have more than one parent. Even in other fields of research, such as historical linguistics, scholars are looking to use networks to model language evolution due to borrowing and hybrid languages (Francois, 2015).

In fact, applying phylogenetic methods to stemmatology enables quicker and more streamlined reconstructions of textual histories as opposed to traditional manual reconstructions. However, stemmatology differs from phylogenetics in several key aspects. Specifically, the assumptions made about the structure of the networks differ and thus also the process of For example, phylogenetics looks at changes over millions of years, while stemmatology can look at different versions of a text over several decades or even shorter. Therefore, the underlying models that explain changes differ considerably (Roelli, 2020, Chapter 5). Moreover, in phylogenetics, generally, only extant species are considered and are thus placed as leaf nodes. An example of a phylogenetic network is shown in Figure 1.3. Indeed, we can observe that only extant species are shown, and are all placed as leaf nodes. On the other hand, in stemmatology, older texts are also included that might have newer versions based on them and thus should not be represented by leaf nodes. In reality, internal nodes are often part of stemmata (Roelli, 2020, Section 5.5.7). Nevertheless, as a result of using phylogenetic algorithms as an "out-of-the-box" tool, most, if not all, texts are treated as leaf nodes when applying them to stemmatology instances (Heikkilä, 2022). This thesis will show how triplets—rooted binary trees on three labels—can be used to encode and recover internally labelled stemmata, allowing for more widely applicable algorithms in several fields of research.

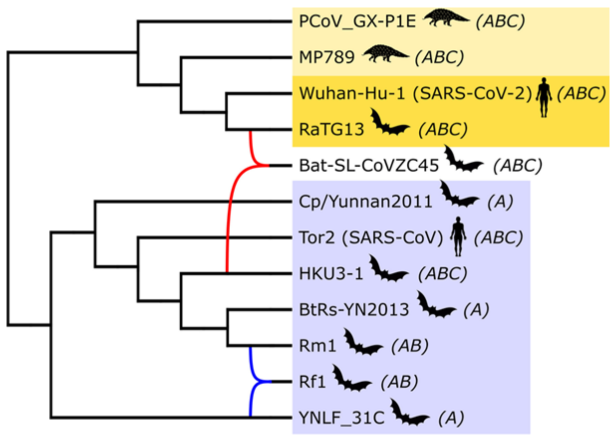


Figure 1.3: A phylogenetic network on coronaviruses produced by Wallin et al., 2021. The leaf nodes represent the genomes of specific variants and the curved edges signify reticulations.

1.1. Previous work

The study of triplets has become a central tool in computational phylogenetics and has gradually been used more often in stemmatology. A rooted triplet encodes the relative relationships among three elements, specifying which pair shares a more recent common ancestor. Collections of such triplets have been shown to encode entire trees under certain conditions, as originally demonstrated by Aho et al. (1981). Their efficient `BUILD` algorithm reconstructs a tree from a set of compatible triplets in polynomial time, forming the basis of many modern methods. The limitation of this method is that it only supports binary trees as input and leaf-labelled nodes.

Ng and Wormald (1996) extended this result to multifurcating trees by incorporating fanned triplets, allowing for nodes with out-degree three or higher. They also created an algorithm capable of finding all possible trees compatible with the given triplet set. However, like Aho et al. (1981), it still only considers tree-like structures, thus not allowing for any reticulations or nodes with out-degree one.

In the context of networks, Jansson and Sung (2006) showed that a dense set of triplets (at least one triplet for every label) can be used to reconstruct level-1 networks, although such reconstructions are not unique. This result was further refined in later work on trinets and more general network classes by van Iersel et al. (2022) and To and Habib (2009). Their algorithms all ran in polynomial time, which allowed for practical use cases where limited reticulations were now possible.

Recent research has also investigated how triplet-based distances can serve as similarity measures for comparing tree structures. Of particular relevance is the work of Ciccolella et al. (2021), who introduced a triplet-based similarity score for fully multi-labelled trees (each node has at least one label) with poly-occurring labels (any label can occur at several nodes). In biological contexts, such labels arise in cancer phylogenies, where the same mutation may appear in multiple branches of a tree, and the DNA of internal nodes is often known. In stemmatology, labelled internal nodes occur when older manuscripts have survived to the present day and can be used to determine the texts' ancestry. The work of Ciccolella et al. demonstrates that triplets can still offer meaningful comparisons in such settings, provided the structure of label occurrence is taken into account. However, they did not prove that these triplets encode such trees, and thus their measure lacks the theoretical basis to be used as a metric.

Despite these advances, several important gaps remain. Most algorithms and distance measures in phylogenetics and stemmatology are designed for leaf-labelled trees or networks. Internal labels — while present in many real-world cases of stemmata — are usually ignored, or have to be manually added afterwards. As a result, there is a lack of formal theory surrounding the use of internal labels in graph encoding and comparison. Even in recent work on fully-labelled trees, such as in Ciccolella et al., 2021, the theoretical background for using triplets as a measure is lacking. There is therefore a clear need for a rigorous investigation into whether and how rooted triplets can be used to encode and reconstruct graphs that allow for both internal labels and reticulations.

1.2. Our contributions

This thesis addresses this research gap by developing a general framework for encoding and reconstructing internally labelled stemmata using rooted triplets. The core contribution is a series of theoretical results demonstrating that under suitable conditions, the complete set of rooted triplets of a graph uniquely determines its structure, even in the presence of internal labels and cycles. We first consider multifurcating trees with internal labels, establishing that triplets suffice to encode such structures. We then extend this result to general rooted trees, allowing for nodes with out-degree one and more complex configurations. Finally, we prove that level-1 networks, which permit limited reticulation, can also be uniquely determined by their triplets if certain labelling conditions are met.

In addition to these encoding theorems, we present reconstruction algorithms that recover each graph class from its triplet set. These algorithms generalise existing methods such as `BUILD` by incorporating procedures for handling internal labels and cycles. We also investigate triplet-based distance measures, adapting the classic triplet distance to account for internal labelling and testing its behaviour on both synthetic and real-world data. Our experiments suggest that triplet distances can meaningfully compare internally labelled graphs, and that some of our reconstruction methods can cope with incomplete triplet sets.

The structure of the thesis is as follows. Chapter 2 reviews key definitions and formal background. Section 3.1 presents the encoding theorem and reconstruction algorithm for multifurcating trees with internal labels. Section 3.2 extends this framework to general stemmatic trees, while Chapter 4 turns to level-1 networks, developing the necessary theory and algorithmic tools. Chapter 5 describes the computational performance of our algorithms. In Chapter 6 we apply the triplet metric, as proposed in Chapter 2, to artificial and real-world data. Finally, Chapter 7 offers a critical reflection on our findings and outlines promising directions for future research.

Through this work, we provide a theoretical foundation for triplet-based encoding and comparison of internally labelled stemmata, with potential applications not only in textual criticism but also in other fields such as cancer phylogenies and linguistics.

2 Preliminaries

2.1. Graphs

A *graph* G is a set of vertices V with a set of edges E connecting those vertices. It is written as $G = (V,E)$. In the context of this thesis, they are assumed to be directed acyclic graphs. A vertex $c \in V$ is a *child* of another vertex $v \in V$ if $vc \in E$ exists. Here v is the *parent* of c. A *descendant* of a vertex $v \in V$ is a vertex $d \in V \setminus \{v\}$ such that a path exists from v to d in G. Likewise, an *ancestor* of a vertex $v \in V$ is a vertex $a \in V \setminus \{v\}$ such that a path exists from a to v in G. The *subgraph* rooted at a vertex $v \in V$ is the induced subgraph of G by v and its descendants and has v as its root.

A *(partially-)labelled graph* is a graph that contains a set of labels X, where each label is placed at a different vertex. However, not all vertices have to contain a label. The function $l : X \to V$ is an injective function mapping the labels to their respective vertices. Such a labelled graph is denoted as $G = (V,E,l)$. Figure 2.1 shows an example of such a graph.

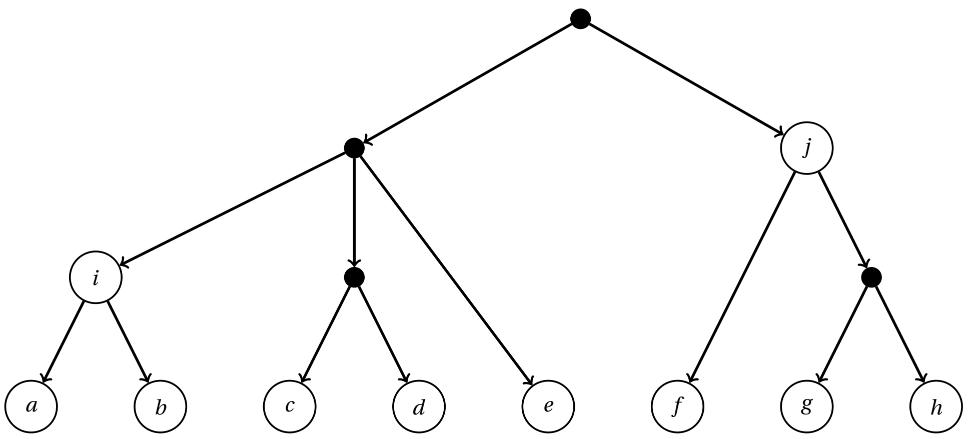


Figure 2.1: A graph with $X = \{a,b,c,d,e,f,g,h,i,j\}$ where all leaves and some internal nodes are labelled. Unlabelled nodes are depicted as black dots.

Definition 2.1. Let $G = (V,E,l)$ be a graph with labels X and $v_1v_2 \in E$ an edge such that at most one of $v_1 \in V$ and $v_2 \in V$ is labelled and v_1 has out-degree one. Then *contracting* v_1v_2 is the action of deleting this edge from E, and merging the two nodes. This results in a graph $\tilde{G} = (\tilde{V},\tilde{E},\tilde{l})$ with labels X. Here $\tilde{V} = V \setminus \{v_1\}$, $\tilde{E} = (E \setminus \{v_1v_2,\ p_1v_1,\ldots,p_kv_1\}) \cup \{p_1v_2,\ldots,p_kv_2\}$ where $p_1,\ldots,p_k \in V$ are the parents of v_1 if they exist. Moreover, if $x \in X$ such that $l(x) \in \{v_1,v_2\}$, then $\tilde{l}(x) = v_2$ and $\tilde{l}(y) = l(y)$ for all other $y \in X$.

Definition 2.2. Let $G = (V,E,l)$ be a graph with labels X, and let $x \in X$. Then G *without* x, or $G \setminus x$, is the subgraph of G formed by removing x from X and making its vertex $l(x) \in V$ unlabelled, taking $l : X \setminus \{x\} \to V$ the same, and applying the following rules until none apply:

1. Removing any unlabelled leaf nodes and their edges
2. Contracting any edge $v_1 v_2 \in E$ such that at most one of $v_1 \in V$ and $v_2 \in V$ is labelled and v_1 has out-degree one according to Definition 2.1

Definition 2.3. Take $G = (V, E)$ to be a graph and $u, v \in V$. The *least common ancestor* of u and v in G is the vertex $w \in V$, such that w is an ancestor of both u and v and no descendant of w is also an ancestor of both u and v. We write $\text{LCA}(u, v)$ to refer to the least common ancestor.

When looking at Figure 2.1, it can be observed that $\text{LCA}(f, h) = j$ and $\text{LCA}(a, e)$ is the unlabelled child of the root on the left. Note that we, at times, use $\text{LCA}(u, v)$ with $u, v \in X$ rather than $u, v \in V$. By this, we imply $\text{LCA}(l(u), l(v))$ instead.

Definition 2.4. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two unlabelled graphs. Then G_1 is *isomorphic* to G_2, say $G_1 \cong G_2$, if and only if there exists a bijection $f : V_1 \to V_2$ such that $u, v \in V_1$ are adjacent in G_1 if and only if $f(u), f(v) \in V_2$ are adjacent in G_2.

Murakami (2021) defined when leaf-labelled graphs are isomorphic. Since we also deal with internally labelled vertices, we alter the definition slightly.

Definition 2.5. Let $G_1 = (V_1, E_1, l_1)$ and $G_2 = (V_2, E_2, l_2)$ be two (partially-)labelled graphs with the same set of labels X. And take $G_1' = (V_1, E_1)$ and $G_2' = (V_2, E_2)$ to be the same graphs as G_1 and G_2, respectively, but without the labels. Then G_1 is *isomorphic* with G_2 if and only if there exists a bijection $f : V_1 \to V_2$ such that:

(i) f satisfies Definition 2.4

(ii) and $f(l_1(x)) = l_2(x)$ for all $x \in X$.

As a result of this, G_1 and G_2 with the same set of labels X are isomorphic if and only if G_1 can be made the same as G_2 by only reordering the children of any number of parent nodes.

If a graph $G = (V, E, l)$ is directed and connected, we call an edge $e \in E$ a *cut-arc* if removing e from G makes the resulting graph disconnected. Moreover, we call a cut-arc $v_1 v_2 \in E$ a *highest cut-arc* if there is no other cut-arc $v_1' v_2' \in E$ such that v_1 is a descendant of v_1'.

2.2. Triplets

Definition 2.6. Let $G = (V, E, l)$ be a (partially-) labelled graph with label set X. Take $t = (V_t, E_t, l_t)$ to be a rooted connected graph, with three labels u, v, and w of X, such that its underlying undirected graph is acyclic. Then t is called *consistent* with G if t contains no unlabelled vertices with out-degree one and there is a mapping, $\phi : V_t \to V$, from the vertices of this graph to the vertices of the graph G, such that the following holds:

1. For every $xy \in E_t$, there must exist a path connecting $\phi(x)$ to $\phi(y)$ in G. All these paths are edge-disjoint and vertex-disjoint except for the end points.
2. $\phi(l_t(u)) = l(u), \phi(l_t(v)) = l(v)$, and $\phi(l_t(w)) = l(w)$

Such a rooted tree is called a *triplet* of G, and the set of all triplets on u, v, and w is denoted by $G|_{u,v,w}$.

The set of all triplets of a graph G is denoted as $t(G)$ and contains all possible trees with three labels from X consistent with G.

2.3. Trees

Trees differ from graphs in the sense that their underlying undirected graph is acyclic and connected. They can represent vertical descent between species, texts or other objects.

Definition 2.7. Let X be a set of labels. A *(partially-)labelled tree* is a tree with the following properties:

1. A single root exists with in-degree zero
2. The leaves are the vertices with out-degree zero

3. The internal vertices have in-degree one and out-degree larger than or equal to one
4. All the leaves of the tree are labelled

A (partially-)labelled tree is denoted as $T = (V, E, l)$, where V represents the set of vertices, E is the set of directed edges, and $l : X \to V$ is an injective function that maps each label to its corresponding vertex.

Note that the graph in Figure 2.1 is in fact a partially-labelled tree.

In order to talk about the structure of trees, it is useful to isolate substructures that represent shared ancestry among groups of texts. Such a substructure, called a branch, represents a child and its descendants of the root. Intuitively, we can expect all the texts of a branch to have some shared similarity, different from the other branches. Formally, we define a branch as follows.

Definition 2.8. Let T be a tree on X (with a labelled root $\rho \in X$) and $c_1, \dots, c_k$ the children of the root. The *branches*, $B_1 \dots B_k$, of T are defined as the partition of the label set $X(\setminus\{\rho\}) = B_1 \cup \dots \cup B_k$ such that B_i contains all labels that appear in the subtree rooted at the i-th child of the root. Then each B_i is a *branch* of T.

2.4. Networks

While trees model hierarchical relationships, they cannot capture structures where nodes have multiple direct predecessors. In such cases, we turn to networks, which generalise trees by allowing cycles and more complex connectivity.

Definition 2.9. Let X be a set of labels. A *(partially-)labelled network* is a directed acyclic graph with the following properties:

1. A single root exists with in-degree zero
2. The leaves are the vertices with out-degree zero
3. The in- and out-degree of internal vertices is larger than or equal to one
4. All the leaves of the network are labelled

A (partially-)labelled network is denoted as $N = (V, E, l)$. Here, V represents the set of vertices, E is the set of directed edges, and $l : X \to V$ is an injective function that maps each label to its corresponding vertex.

The underlying undirected graph of a network $N = (V, E, l)$ can contain cycles. If $C \subseteq V$ forms a cycle in the underlying undirected graph, then the vertex $s \in C$ such that all vertices in $C \setminus \{s\}$ are descendants of s in N is called the *source* of C. A vertex $r \in C$ is called the *sink* of C if it is a descendant of all vertices in $C \setminus \{r\}$. A *highest sink* is a sink, $r \in C$, such that it is not a descendant of another sink $r' \in C'$.

A *biconnected component* of a network N is a connected subgraph induced by a subset of vertices such that removing any single vertex and its edges from this subgraph keeps the subgraph connected.

Note that branches in networks, as defined in Definition 2.8, are often not possible since such a partition does not always exist. However, by looking at the blobgraph of a network, the concept of branches can be used again. The blob tree, or blob graph, of a network is defined as the tree obtained by collapsing every biconnected component of the network into a single node and removing any leaves. This was first proposed by Gusfield and Bansal (2005). We will use a slightly different definition; the blob graph of a network is the tree obtained by performing the following steps for every (inclusion-wise maximal) cycle in the underlying undirected graph:

1. Remove all edges within the cycle
2. Add a single node labelled by all the labels present in the cycle
3. Make all the vertices in the cycle unlabelled
4. Add edges from this newly added node to all nodes in the cycle except the source
5. Add an edge from the source of the cycle to the newly added node
6. "Clean up" the tree according to the rules as described in Definition 2.2.

The blob graph is therefore always a tree, and thus the concept of branches once again works. An example of a network and its blob graph is shown in Figure 2.2. Note that we use B_i^C to denote the i-th branch of the collapsed cycle C in the blob graph, together with the possible label in C that is the direct parent of this branch. For example, $\{h, d, e\}$ and $\{f, g\}$ are such branches.

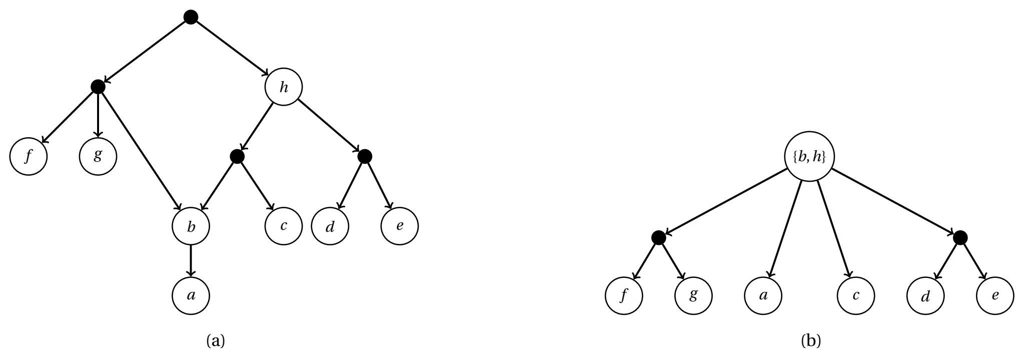


Figure 2.2: A network (a) and its blob graph (b).

2.5. Metric

A metric is a function used to measure the distance between two points in a metric space. More formally, a metric space is an ordered pair (M, d) where M is a set and $d : M \times M \to \mathbb{R}$ is a metric on M where d satisfies the following properties for any $x, y, z \in M$:

1. $d(x, x) = 0$
2. If $x \neq y$, then $d(x, y) > 0$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

Dobson (1975) first introduced the triplet distance between two trees and mentioned it satisfies properties 1 and 3. Dobson defined the triplet distance as a similarity score by counting the number of triplets that two trees share. However, by taking the cardinality of the symmetric difference between the two triplet sets, one gets a distance that satisfies properties 1 and 3 as they are stated.

The triplet distance is thus defined formally as:

$$d(N_1, N_2) = |t(N_1) \Delta t(N_2)| \tag{2.1}$$

Where $A \Delta B$ defines the symmetric difference between two sets A and B.

One of the properties of the symmetric differences is that $A\Delta C = (A\Delta B)\Delta(B\Delta C)$. It thus follows immediately that $|A\Delta C| = |(A\Delta B)\Delta(B\Delta C)| \leq |A\Delta B| + |B\Delta C|$. Therefore, the triplet distance also satisfies property 4. To ensure the triplet distance is a metric, it is left to prove that $d(N_1, N_2) > 0$ holds when N_1 is non-isomorphic with N_2. Or more precisely, that $t(N_1) = t(N_2)$ is equivalent to N_1 and N_2 being isomorphic. In other words, the triplet set encodes a network. This will be proven in Sections 3.1.1, 3.2.1 and 4.1.

It is also possible to normalise the triplet distance by dividing Equation (2.1) by the cardinality of the union of the triplet sets. This gives the following equation for the normalised triplet distance:

$$d_n(N_1, N_2) = \frac{|t(N_1) \Delta t(N_2)|}{|t(N_1) \cup t(N_2)|} \tag{2.2}$$

By Yianilos (2002), Equation (2.2) is a metric, given that triplet sets encode networks, and indeed normalised.

3 Trees

In this chapter, we aim to prove that triplets encode trees and to develop an algorithm that reconstructs a tree based on triplets. We will first prove the encoding and algorithm for specific trees, called *multifurcating trees*, where each non-leaf node has at least out-degree 2. These results are presented in Section 3.1. To expand upon this, in Section 3.2, the triplet encoding is proven for a broader class of trees. We refer to these trees as *general trees*. For these general trees, an algorithm is also presented.

3.1. Multifurcating tree with unlabelled root

In this section, we assume T to be a tree according to Definition 3.1.

Definition 3.1. We call $T = (V, E, l)$ a *multifurcating (partially-)labelled stemmatic tree* with an unlabelled root if the following holds:

1. T is a tree according to Definition 2.7
2. All internal nodes, as well as the root, have out-degree larger than or equal to two
3. The root is unlabelled

Multifurcating trees are slightly limited in the information they can convey when considering internal labels. Namely, every internal label must have at least two children. So whenever a text has been used to only directly produce one new text, these trees are not able to properly visualise the stemma. However, when removing any internal labels, the tree is a multifurcating phylogenetic tree. Therefore, these trees are still interesting to consider as they are the most obvious extension to the phylogenetic trees.

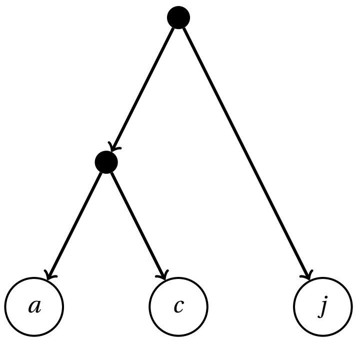
Resolved triplets, denoted as $uv|w$, and fanned triplets, denoted as $u|v|w$, are the only triplets present in multifurcating phylogenetic trees, as only the leaves are labelled. These triplets are therefore used when looking at triplet distances or triplet reconstruction algorithms for multifurcating phylogenetic trees. For multifurcating stemmatic trees, we will argue that only using fanned and resolved triplets is sufficient to encode the tree. Therefore, for multifurcating trees, $t(T)$ denotes the set of all triplets according to Definition 2.6 such that none of the labels are ancestors of another.

Some examples of triplets of the tree in Figure 2.1 are shown in Figure 3.1. Figure 3.1a shows a resolved triplet of the form $ac|j$, while Figure 3.1b shows a fanned triplet of the form $b|d|e$.

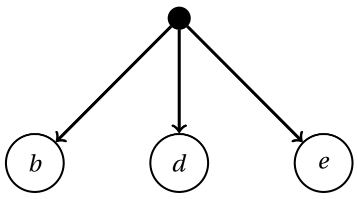
Brynt (1997) has already shown that the triplets encode any phylogenetic multifurcating tree where only the leaf nodes are labelled.

Theorem 3.1 (Brynt, 1997, Theorem 2.1)**.** Given a multifurcating phylogenetic tree T where only the leaves are labelled by the set X, then $t(T)$ encodes T.

To prove the encoding and algorithm, for any given label u, the set D_u is introduced. These sets contain information on which other labels share a triplet with the label u.



(a) The resolved triplet $T|_{a,c,j}$.



(b) The fanned triplet $T|_{b,d,e}$.

Figure 3.1: Two examples of triplets obtained from the tree in Figure 2.1.

Definition 3.2. Let T be a (partially-)labelled tree and take $u \in X$. Then the set of all labels in X that do not form any triplet with u is denoted as $D_u^T = \{x \in X | x \neq u, \forall t_i \in t(T) : \{x, u\} \not\subseteq t_i\}$.

If it is clear from the context which tree this set belongs to, D_x is written rather than D_x^T. Note that for the tree in Figure 2.1, $D_a = \{i\}$ and $D_j = \{f, g, h\}$.

Using this D_u set, we know which labels are u's ancestors and descendants as shown in Lemma 3.2.

Lemma 3.2. Let T be a multifurcating tree with label set X, with $|X| \geq 3$. Let $u \in X$ be a labelled node. Then $v \in D_u$ if and only if v satisfies one of the following three statements:

- v is a descendant of u
- v is an ancestor of u
- v is a sibling of u if u and v share the root as their direct parent and the root has out-degree 2.

Proof.
Suppose v is a descendant of u. Then no triplet can exist containing both u and v as this would require both to be leaves in the triplet, which is impossible as v is a descendant of u.

Suppose v is an ancestor of u. Then, for the same reasoning as above, no triplet can exist containing both u and v.

Suppose v is a sibling of u such that u and v share the root as their direct parent, and the root has out-degree two. Then all $x \in X$ different from both u and v is a descendant of either u or v and thus cannot form a triplet with u and v by the reasoning above.

Now, assume v does not satisfy any of the three statements with respect to u, then we must show $v \not\in D_u$. In this case, either (i) u and/or v are not children of the root, or (ii) they are both children of the root but the root has an out-degree larger than two.

(i) Either (a) u and v are in the same branch of the root or (b) they are in different branches of the root.

   (a) Since for u and v neither is a descendant of the other, we can take any $x \in X$ such that x is in another branch of the root to form a triplet of the form $uv|x$.

   (b) Since u and v do not both have the root as their parent, either one of them must have a sibling that is not a descendant or ancestor of u and v. Thus u, v, and that sibling (or one of its labelled descendants) $x \in X$ can form a triplet of the form $ux|v$ or $u|xv$.

(ii) Since the root has out-degree larger than two, it has at least three children, say u, v, and w. Then u, v, and w (or one of the labelled descendants of w) can form a fanned triplet $u|v|w$.

Thus indeed $v \not\in D_u$. □

3.1.1. Unique triplet encoding

To prove that a multifurcating tree is encoded by its triplet set, we will use induction on the number of labels. This only works if all triplets of a subtree are also triplets of the tree itself. Lemma 3.3 proves this formally.

Lemma 3.3. Let T be a tree on X, and $\tilde{T}$ be a subtree of T obtained by removing any number of labels as described by Definition 2.2. R is the set of labels in T that are not present in $\tilde{T}$. Then $t(\tilde{T}) \subseteq t(T)$ and for all $t \in t(T) \setminus t(\tilde{T})$ we have $t \cap R \neq \emptyset$.

Proof.
For the first statement, suppose T is a tree with no out-degree one vertices and an unlabelled root, and $\tilde{T}$ is a subtree of T. For a contradiction, assume $t(\tilde{T}) \not\subseteq t(T)$, then there exists a $t \in t(\tilde{T})$ such that $t \notin t(T)$. Without loss of generality, assume $u, v, w \in X$ form this triplet t. Then in $\tilde{T}$, none of u, v, w is a descendant of another by Lemma 3.2. But since $t \notin t(T)$, either (i) u, v, w do not form a triplet in T or (ii) their triplet has a different form than t.

(i) Then one of u, v, w is a descendant of another in T. But then $\tilde{T}$ cannot be a subtree of T.

(ii) In this case, the LCAs of at least one pair must have changed. However, since $\tilde{T}$ is a subtree of T, this is not possible.

In both cases, we get a contradiction. Thus indeed $t(\tilde{T}) \subseteq t(T)$.

For the second statement, we will show a contradiction. Suppose there exists a triplet $t \in t(T) \setminus t(\tilde{T})$ such that t contains no nodes from R. Then all nodes of t are also in $\tilde{T}$. However, since $t \notin t(\tilde{T})$, one of the nodes is a descendant of another in $\tilde{T}$. But since $\tilde{T}$ is a subtree of T, it means that one of the nodes is also a descendant of another in T. This would mean that no triplet can be formed in T, which leads to a contradiction. So indeed for all $t \in t(T) \setminus t(\tilde{T})$ we have $t \cap R \neq \emptyset$. □

Figure 3.2 shows a subtree, $\tilde{T}$, of the tree, T, in Figure 2.1 where $R = a, b, c, f$. Indeed, all triplets of $t(\tilde{T})$ are also triplets of $t(T)$, and any triplet in $t(T) \setminus t(\tilde{T})$ contains a label of R. Intuitively, this can be expected as a triplet is itself a subtree. And a subtree of a subtree ought to again be a subtree of the original tree as well.

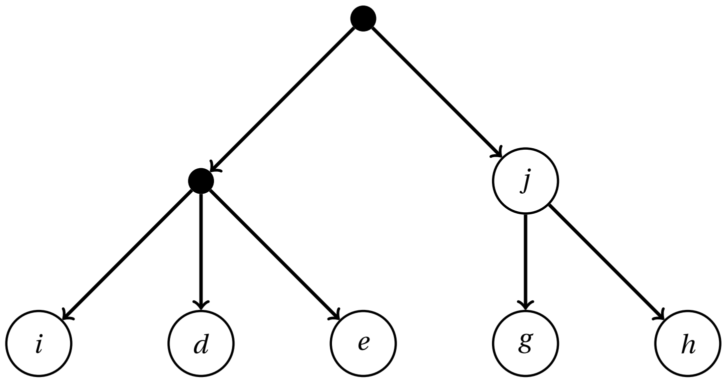


Figure 3.2: A subtree of the tree presented in Figure 2.1.

To apply induction for our encoding proof, we want to remove a label to decrease the number of labels such that the remaining trees are still non-isomorphic. However, it is not always possible to remove a single label while keeping the two trees non-isomorphic and having them satisfy Definition 3.1. Therefore, Lemma 3.4 shows that in those cases, there is a label whose D_u sets differ in the two trees.

Lemma 3.4. Let T_1, T_2 be non-isomorphic trees on X, where $|X| \geq 4$. Then if for all $x \in X$, $T_1 \setminus x$ is isomorphic with $T_2 \setminus x$ when both $T_1 \setminus x$ and $T_2 \setminus x$ are trees with no out-degree one vertices or labelled roots, then there exists a $y \in X$ such that $D_y^{T_1} \neq D_y^{T_2}$.

Proof.
Suppose that for all $x \in X$, $T_1 \setminus x$ is isomorphic with $T_2 \setminus x$ when both $T_1 \setminus x$ and $T_2 \setminus x$ are trees with no out-degree one vertices or labelled roots.

Suppose there are no labelled internal vertices in T_1 and T_2. Using Theorem 3.1, we can arrive at a contradiction. Since $T_1 \ncong T_2$, without loss of generality, T_1 contains a triplet $t = T_1|_{u,v,w}$ that is not in T_2. Since $|X| \geq 4$, take any $x \in X \setminus \{u, v, w\}$. Then if x is removed, t will still be a triplet in $T_1 \setminus x$ but not in $T_2 \setminus x$ by Lemma 3.3. And both $T_1 \setminus x$ and $T_2 \setminus x$ are still trees with no out-degree one vertices or labelled roots, as no labelled internal vertices were present. Therefore, $T_1 \setminus x$ is non-isomorphic with $T_2 \setminus x$. Thus, the first assumption of the proof cannot hold, which leads to a contradiction. So T_1 or T_2 must contain an internal vertex that is labelled.

Now, suppose without loss of generality that T_1 contains a labelled internal vertex, $y \in X$. Then either (i) y is a leaf node in T_2, or (ii) y is also an internal node in T_2.

(i) If $D_y^{T_1} \neq D_y^{T_2}$, we are done.

    If not, then all of y's labelled descendants in T_1 must be ancestors of y in T_2 since $|D_y^{T_1}| \geq 2$. Namely, suppose, without loss of generality, $x, z \in X$ are y's labelled descendants in T_1, z is in a different branch from the root than y in T_2, and x is an ancestor of y in T_2. Then x must have another labelled descendant different from y. This other labelled descendant, y, and z can thus form a triplet which implies $D_y^{T_1} \neq D_y^{T_2}$.

    Therefore, at least one of y's leaf descendants in T_1, say $x \in X$, must be an ancestor of another of y's leaf descendants, say $z \in X$. Thus, $x \notin D_z^{T_1}$, but $x \in D_z^{T_2}$ by Lemma 3.2. Therefore, we found a $z \in X$ such that $D_z^{T_1} \neq D_z^{T_2}$.

(ii) By assumption, we know $T_1 \setminus y \cong T_2 \setminus y$ as both $T_1 \setminus y$ and $T_2 \setminus y$ are trees with no out-degree one vertices or labelled roots. But then $D_y^{T_1} \neq D_y^{T_2}$, as otherwise T_1 would already have been isomorphic with T_2.

Thus we have shown that if for all $x \in X$, $T_1 \setminus x$ is isomorphic with $T_2 \setminus x$ when both $T_1 \setminus x$ and $T_2 \setminus x$ are trees with no out-degree one vertices or labelled roots, there is a labelled vertex $y \in X$ such that $D_y^{T_1}$ is unequal to $D_y^{T_2}$. □

Indeed Figure 3.3 shows such an example. The trees in Figures 3.3a and 3.3b, respectively T_1 and T_2, are non-isomorphic, but no single label can be removed to obtain two smaller non-isomorphic multifurcating trees. Removing b or c would cause T_2 not to satisfy Definition 3.1. Likewise, removing e would make T_1 a tree with a labelled root and thus also not satisfy Definition 3.1. Meanwhile, removing a or d would make the two trees isomorphic. As Lemma 3.4 suggest we have $\{b, c, d, e\} = D_a^{T_1} \neq D_a^{T_2} = \{a, c\}$, $\{a\} = D_d^{T_1} \neq D_d^{T_2} = \emptyset$, and $\{a\} = D_e^{T_1} \neq D_e^{T_2} = \emptyset$.

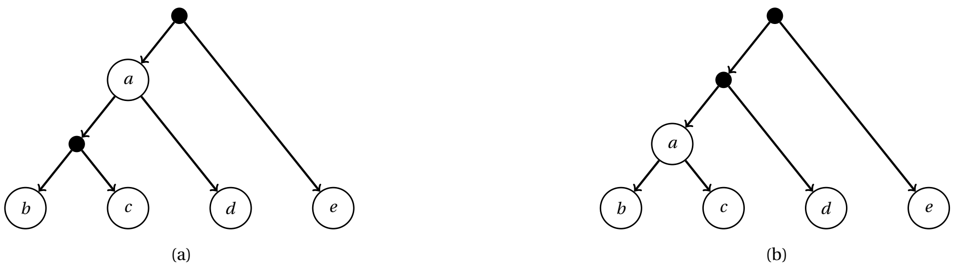


Figure 3.3: Two non-isomorphic multifurcating trees where no single label can be removed to again end up with two non-isomorphic multifurcating trees.

With the help of Lemmas 3.3 and 3.4, we can now prove that triplets encode a multifurcating tree. Theorem 3.5 gives this result and thus shows that the triplet distance as defined in Equations (2.1) and (2.2) is a metric.

Theorem 3.5 (Unique triplet encoding)**.** Let T_1 and T_2 be two trees with no out-degree one nodes and unlabelled roots with the same set of labelled nodes X. Then T_1 and T_2 are isomorphic if and only if $t(T_1) = t(T_2)$.

Proof.
The cases where $|X| \leq 2$ are trivial as $t(T_1) = t(T_2) = \emptyset$ and at most one tree is possible, so we will look at the cases where $|X| \geq 3$.

For the first direction, suppose $T_1 \cong T_2$, then for all $u, v \in X$, their least common ancestor is the same. Thus, the triplet induced by any three labelled vertices in X will be of the same form in T_1 as in T_2 if it exists, since the form is determined by the least common ancestors alone. Thus $t(T_1) = t(T_2)$.

For the other direction, we will prove the contrapositive. Let T_1 and T_2 be two trees on X, with $|X| = 3$, and no out-degree one vertices or labelled roots. Without loss of generality, assume $X = \{u, v, w\}$. Then T_1 and T_2 can only be of the form $uv|w$, $u|vw$, $uw|v$, or $u|v|w$. Suppose that $T_1 \ncong T_2$, then T_2 is of a different form from T_1. For example, T_1 is of the form $uv|w$ while T_2 is of the form $u|vw$. Then $t(T_1) = \{uv|w\}$ is unequal to $t(T_2) = \{u|vw\}$. This holds for any possible combination of forms that T_1 and T_2 can take such that $T_1 \ncong T_2$. We can therefore conclude $t(T_1) \neq t(T_2)$ if $T_1 \ncong T_2$ when $|X| = 3$.

To apply induction, suppose we have proven the claim for all $|X| \leq n-1$, with $n \geq 4$. Let T_1, T_2 be two trees on X, where $|X| = n$, such that $T_1 \ncong T_2$. Then if there exists $x \in X$ such that $T_1 \setminus x \ncong T_2 \setminus x$ and both $T_1 \setminus x$ and $T_2 \setminus x$ are trees with no out-degree one vertices or labelled roots, take any such x and let $\tilde{T}_1 = T_1 \setminus x$ and $\tilde{T}_2 = T_2 \setminus x$ be non-isomorphic trees on $X \setminus \{x\}$. By the induction hypothesis we have $t(\tilde{T}_1) \neq t(\tilde{T}_2)$ since $|X \setminus \{x\}| \leq n-1$. By Lemma 3.3, we know that $t(T_1) \neq t(T_2)$.

If no such label exists, by Lemma 3.4 there exists a label y such that $D_y^{T_1} \neq D_y^{T_2}$. Therefore, without loss of generality, there exists a $x \in D_y^{T_1}$ such that $x \notin D_y^{T_2}$. Thus, a triplet exists in T_2 with x and y while this triplet does not exist in T_1. Thus $t(T_1) \neq t(T_2)$. □

3.1.2. Algorithm

Now that we know triplets encode a multifurcating tree, we can look at creating an algorithm to reconstruct the tree based on the triplet set. Aho et al. (1981) first proposed a version of this problem and gave an algorithm that can construct a tree based on triplets such that all triplets are consistent with the obtained tree. This problem has been further explored and formalised. The problem is known as the $\mathcal{R}^{+-}\mathcal{F}^{+-}$ consistency problem and is well-researched. As an input it has four sets, $\mathcal{R}^+, \mathcal{R}^-, \mathcal{F}^+$, and $\mathcal{F}^-$, where $\mathcal{R}^+$ and $\mathcal{R}^-$ are sets of resolved triplets, and $\mathcal{F}^+$ and $\mathcal{F}^-$ are sets of fanned triplets. The problem asks whether or not a phylogenetic tree exists such that the triplets in $\mathcal{R}^+$ and $\mathcal{F}^+$ are consistent while the triplets in $\mathcal{R}^-$ and $\mathcal{F}^-$ are not.

Harvey et al. (2024) have proven the last open cases of this problem, where one also requires that every vertex of the resulting tree must have an out-degree at most a given $D \in \mathbb{N}_+$. The more general cases, where the vertices of the resulting tree can have any out-degree, had already been solved by Ng and Wormald (1996).

The problem discussed in this section is known as the $\mathcal{R}^+\mathcal{F}^+$ consistency problem, which is equivalent to the general consistency problem where $\mathcal{R}^- = \mathcal{F}^- = \emptyset$. It thus concerns finding a tree such that all the given triplets are consistent with this tree, if it exists. Ng and Wormald have shown that this problem is polynomially solvable for phylogenetic trees. Their reconstruction algorithms theoretical running time is bounded by $\mathcal{O}(|X|^2|t(T)|\alpha(|X|+|t(T)|))$ (Ng & Wormald, 1996, Theorem 4), where α is the inverse Ackermann function. Note that their result has been rewritten to our notation, and therefore, their theoretical running time is slightly lower. Their algorithms running time is thus capped at $\mathcal{O}(|X|^2|t(T)|)$ for reasonable values of $|X|+|t(T)|$ (Cormen et al., 2009, Section 21.4).

Due to Theorem 3.5, we know that if the full triplet set is given as input, the same tree must be returned. If a subset of the triplets is returned, any tree that has the input triplets in its triplet set suffices. We will present an algorithm that can find a tree for any subset of a triplet set in $\mathcal{O}\left(|t(T)|^2|X| + |X|^2|t(T)|\right)$ running time. Different from other algorithms such as those from Aho et al. (1981) and Ng and Wormald (1996), this algorithm is capable of handling internal labels and, whenever possible, tries to place a label as an internal label instead of a leaf label. Therefore, this algorithm can construct more compact trees.

Our proposed algorithm works by dividing the labels into branches, looking if that branch contains a child of the root and then recursively resolving the branches. Lemma 3.7 gives us the required result to be able to divide the labels into their respective branches. Since multifurcating trees do not have a labelled root, we need to manually remove the labelled roots of the branches. Lemma 3.6 proves that the child of a root can only occur in specific triplets. This is then used in our algorithm to find the children of the root, which allows us to use recursion on the remainder of the branches.

Lemma 3.6. Let T be a (partially-)labelled tree on X and take $u \in X$. Then u is a child of the root if and only if all triplets $t_i \in t(T)$ containing u are of the form $u|vw$ or $u|v|w$ with $v, w \in X$.

Proof.
Suppose first that $u \in X$ is a child of the root. By Lemma 3.2, no triplet exists that contains both u and one of its descendants. Thus, any triplet containing u has the root separating u from the other labelled nodes in the triplet, say v and w. These other labelled nodes are either in different branches from the root, or in the same branch from the root; in which case, they share one of u's siblings as a common ancestor. If v and w are in the same branch, the triplet is of the form $u|vw$; otherwise, the triplet is fanned and thus of the form $u|v|w$. Therefore for all $v, w \in X$, the triplet on $\{u, v, w\}$, if it exists, is of the form $u|vw$ or $u|v|w$.

For the other direction, we prove the contrapositive. Suppose $u \in X$ is not a child of the root. Then u and its sibling have a least common ancestor that is also a descendant of the root. Let v be the sibling of u if the sibling is labelled and one of the labelled descendants of the sibling, otherwise. Without loss of generality, let u and v be on the first branch of the root. Then take any labelled node, say w, from any other branch of the root. Then $uv|w \in t(T)$. □

Indeed, note how node j in Figure 2.1 is a child of the root and only has triplets such as $ab|j$, $ac|j$, $ie|j$, etc.

Lemma 3.7. Let T be a tree. Then for all triplets $t_i \in t(T)$ of the form $uv|w$, with $u, v, w \in X$, we have u, v are in the same branch B_j. Also if $u, v \in B_j$ and $u|v|w \in t(T)$, then also $w \in B_j$.

Proof.
For a contradiction of the first statement, suppose there exists a $uv|w \in t(T)$ such that $u \in B_i$ and $v \in B_j$ with $i \neq j$. Then LCA(u, v) is the root of T. While, according to the triplet, LCA(u, v) is a descendant of both LCA(u, w) and LCA(v, w). However, LCA(u, v) is the root and thus cannot be a descendant of any node. Thus, no such triplet can exist.

For the second statement, suppose $u, v \in B_i$ and $u|v|w \in t(T)$. Assume $w \notin B_i$, then w is separated by the root from u and v, and LCA(u, v) is a descendant of the root. Thus, a triplet of the form $uv|w$ exists, not $u|v|w$. This leads to a contradiction and thus $w \in B_i$. □

Algorithm 6 divides the labels of X into their corresponding branches by iterating over the triplets and properly handling them according to Lemma 3.7. The algorithm uses Algorithms 4 and 5, which handles resolved and fanned triplets, respectively. Algorithm 6 runs in $\mathcal{O}\left(|t(T)|^2 + |t(T)||X|\right)$ time. Algorithms 4, 5 and 6 are direct implementations of Lemmas 3.2 and 3.7 and thus will not be proven directly. The mentioned algorithms can be found in Appendix A.1.1.

Algorithm 1 runs in $\mathcal{O}\left(D(|t(T)| + |t(T)|^2 + |t(T)||X| + |X|) + |t(T)|\right)$ where D is the depth of the tree. Note that D is bounded by $|X| - 1$. Therefore, the algorithm's runtime is bounded by $\mathcal{O}\left(|t(T)|^2|X| + |X|^2|t(T)|\right)$

Theorem 3.8 proves that the tree returned by Algorithm 1 contains all triplets of the triplet set given as input.

Theorem 3.8. Let T be a tree on X with triplet set $t(T)$. Let $t'(T)$ be a subset of $t(T)$, and $\tilde{T}$ be the output from Algorithm 1 with input $(t'(T), X)$. Then $\tilde{T}$ contains all the triplets in $t'(T)$. Moreover, if $t'(T) = t(T)$ then $\tilde{T} = T$.

Proof.
We will prove the theorem through induction. Without loss of generality, let T be a tree on $X = \{u, v, w\}$, where $|X| = 3$. Then either (i) $t(T) = \{u|v|w\}$ or, without loss of generality, (ii) $t(T) = \{uv|w\}$.

(i) Whether $t'(T)$ is the empty set or equal to $t(T)$, no resolved triplets will be present. Therefore, $C = X$, Algorithm 6 will place all labels in separate branches, and T will be returned by Algorithm 1.

(ii) Again, two cases exist: $t'(T) = t(T)$ or $t'(T) = \emptyset$

If $t'(T) = t(T) = \{uv|w\}$, we only have resolved triplets and $C = \{w\}$. Then Algorithm 6 will place u and v in the same branch and w in a separate one.

We, therefore, obtain the tree $uv|w$, which contains all triplets in $t'(T)$.

If $t'(T)$ is the empty set, we, by the same reasoning as above, obtain the tree with u, v, w as children of the root. Since $t'(T) = \emptyset$, we indeed have that every triplet in $t'(T)$ is contained in the obtained tree.

Algorithm 1: Multifurcating tree reconstruction

```
1  Function BuildTree($t'(T), X$):
       Input: $t'(T)$ — a set of triplets
       Input: $X$ — the labelled nodes of T
       Output: $T$ — a tree such that it contains all triplets in $t'(T)$ if it exists
2      Compute $D_x$ for all $x \in X$                    // Line is only called in the first iteration
3      if $|X| = 2$ then
4          $T$ = cherry containing the nodes in $X$ as leaves
5      else
6          $C = X$
7          for $t \in t'(T)$ do
8              if $t_i$ is of the form $x|vw$ then
9                  $C$.remove($v, w$)
10         Divide $X$ into the branches $\{B_1, B_2, ..., B_k\}$ using Algorithm 6.
11         if $k = 1$ then
12             raise Error               // The triplets are contradictory or the tree is not
                multifurcating
13         forall $B_i$ do
14             forall $x \in B_i \cap C$ do
15                 if $B_i \setminus D_x = \{x\}$ then
16                     Add $x$ as a child to $T$ and obtain the tree, with $x$ as the root, for its descendants using
                        BuildTree($\{t_i \in t'(T) | t_i \subseteq B_i \setminus x\}, B_i \setminus x$).
17                     break
18             else
19                 Add an unlabelled child to $T$ and obtain the tree, with this unlabelled node as its root, for
                    its descendants using BuildTree($\{t_i \in t'(T) | t_i \subseteq B_i\}, B_i$)
20     return $T$
```

Thus, for $|X| = 3$ we always obtain a tree which contains all triplets in $t'(T)$ and see that if $t'(T) = t(T)$ we obtain T itself.

To apply induction, suppose that the theorem is true for all trees with $|X| \leq n-1$. Then Algorithm 6 properly places all labels in their respective branches. Any child of the root will be in C by Lemma 3.6 and must have all the other nodes in the branch in their D set by Lemma 3.2. Moreover, any label chosen in Line 16 cannot have a triplet with any of the other labels in that branch. Thus, any $c \in C$ can be chosen as a child of the root, if it exists, and picked as the root of this branch. Due to the induction hypothesis, the algorithm can properly resolve each branch, as B_i contains $n-1$ or fewer labels.

We can conclude that the obtained tree will contain all triplets in $t'(T)$.

Lastly, if $t'(T) = t(T)$, the obtained tree will contain all triplets induced by T. Therefore, by Theorem 3.5, the obtained tree must be the same as T. □

If a given set of triplets is contradictory (i.e. no tree satisfies the triplet set), then the algorithm should not return a tree. When a tree exists that satisfies the triplets, then such a tree should be returned, even if it is not unique.

Theorem 3.9. Let $t'(\tilde{T})$ be some set of triplets. If there is a tree with no out-degree one nodes or labelled root, T, satisfying the triplets $t'(\tilde{T})$, then Algorithm 1 returns a tree. Otherwise, no tree is returned.

Proof.
Suppose the theorem does not hold and assume that a T that satisfies the triplet set $t'(\tilde{T})$ is as small a counterexample as possible, such that no tree is returned by Algorithm 1. Then, by Line 12 in Algorithm 1, either only one branch was returned from the root down, or for some internal node, only one branch was returned.

In the first case, let $w \in X$ be a labelled node in a branch different from a given $u, v \in X$ in T. Then all nodes were merged into one branch in one of two scenarios:

(i) Without loss of generality there is a triplet $uw|v$ in $t(\tilde{T})$. Then these branches are merged into B_1 by Algorithm 4. And no further branches were created by the remainder of the algorithm; otherwise, we would not end up with one branch. However, this triplet is impossible as u, v were in different branches than w. Therefore, $t'(\tilde{T})$ is impossible to satisfy.

(ii) Two branches remained after processing all resolved triplets and some fanned triplets, and without loss of generality, there is a triplet $u|v|w$. And no further branches were created by the remainder of the algorithm. By the same reasoning as above, this means $t'(\tilde{T})$ is impossible to satisfy.

In the second case, the subtree rooted at that internal node (where the root has been made unlabelled) is a smaller counterexample. Which contradicts the premise of the proof.

Thus no T can exist that satisfies $t'(\tilde{T})$. □

An example of Algorithm 1 will be discussed here. Let T be the tree in Figure 3.4. Note that T has $X = \{a, b, c, d, e\}$ and the following triplets: $ab|c$, $ab|d$, $ab|e$, $ab|f$, $a|cd$, $b|cd$, $a|ce$, $b|ce$, $a|de$, $b|de$, $c|d|e$.

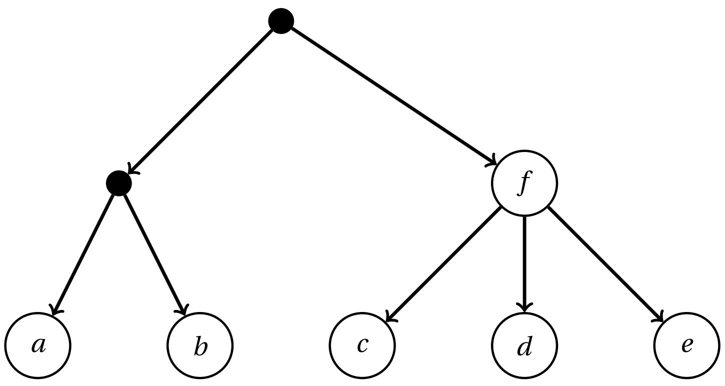


Figure 3.4: The tree T used for an example of the steps in the algorithm.

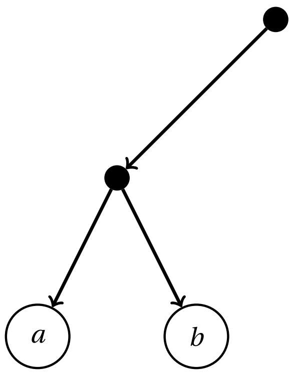


Figure 3.5: The intermediate tree obtained while running the algorithm for the tree in Figure 3.4.

First, since $|X| > 2$, we find the set C. We see that for f, every triplet is either in the form $uv|f$ or $u|v|f$. Thus $C = \{f\}$. The labels are then divided into their branches by Algorithm 6. This gives us $\{a, b\}$ and $\{c, d, e, f\}$. For the first branch, $\{a, b\}$, no label in C is also in the branch, so we add an unlabelled child to the root. The branch is then resolved through recursion. Since $|X| = 2$, they are both added as children of the root. At this point, we obtained the tree as shown in Figure 3.5.

The other branch, $\{c, d, e, f\}$ does contain a label in C, namely f. This label is chosen as the child of the root since $D_f = \{c, d, e\}$ and the other labels are resolved recursively. We will obtain $C = \{c, d, e\}$ and each one is put in their own branch by Algorithm 6. Thus, in Line 16, they are all added as children of f. We, therefore, obtain the tree as shown in Figure 3.4.

3.2. General trees

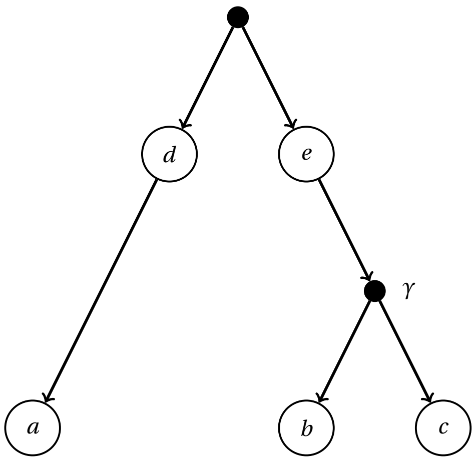
As mentioned in Section 3.1, multifurcating trees are limited in the information they can represent. The trees used in this section do allow for out-degree one nodes as well as labelled roots. Therefore, these trees can now visualise texts that have been copied to a single new text, as well as having a known archetype for the stemma. In phylogenetics, there are no comparable trees as they do not include internal labels and therefore also cannot carry any additional information by allowing internal nodes with out-degree one.

For this section, assume all trees T are according to Definition 3.3.

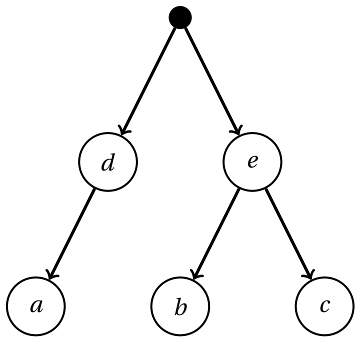
Definition 3.3. We call $T = (V, E, l)$ a *(partially-)labelled stemmatic tree* if the following holds:

1. T is a tree according to Definition 2.7
2. All out-degree one nodes are labelled and point to a labelled node

A tree that does not satisfy the second assumption would have an unlabelled node that represents a relationship that was impossible to know without knowing of the existence of that node in real life. Figure 3.6a shows such a tree.



(a) An incorrect tree according to the assumptions. The unlabelled node named γ violates the assumption that an out-degree one node should point to a labelled node.



(b) The corrected version of the tree in figure a

Figure 3.6: Two general trees where (a) shows a tree not satisfying the assumptions made, and (b) does satisfy the assumptions.

The unlabelled node γ in Figure 3.6a suggests that node e had a descendant of which no record remains, which itself had two descendants b and c. However, without knowing about the existence of node γ, Figure 3.6b visualises the same knowledge. Namely, that nodes b and c are descendants of e.

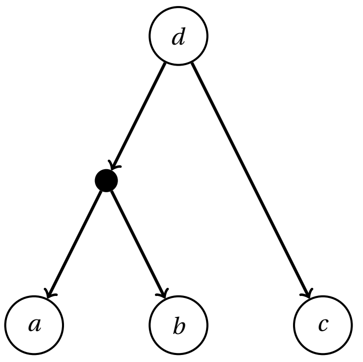
Through a quick counterexample, it can be observed that only using the resolved and fanned triplets used before will not be able to uniquely encode a tree in this case.

Consider T_1 to be the tree in Figure 3.7a and T_2 to be the tree in Figure 3.7b. Indeed, using only the triplets considered in Section 3.1, we would get $t(T_1) = \{ab|c\} = t(T_2)$, and therefore they cannot be distinguished.

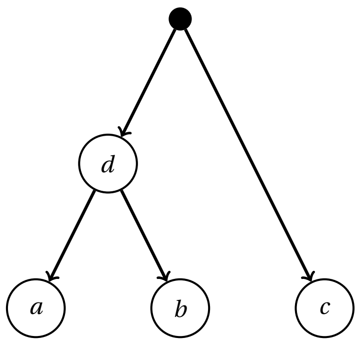
To resolve this issue, more types of triplets need to be included. Figure 3.8 shows all forms of the additional triplets that are possible according to Definition 2.6.

The triplets in Figures 3.8a to 3.8c are all trees in their own right. Therefore, they are all required to encode general trees properly. Without one, the order of the nodes becomes impossible to retrieve. Meanwhile Figure 3.8d does not satisfy Definition 3.3. In fact, Observation 3.10 explains that such a triplet is not needed to encode a general tree.

Observation 3.10. Given the general tree, T, as depicted in Figure 3.9 and $c \in \gamma$ and $d \in \delta$, the triplet $T|_{acd}$ is not needed to fully recover T.

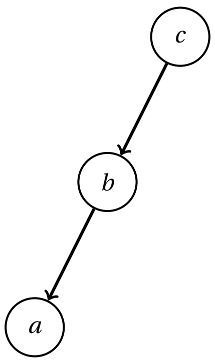


(a)

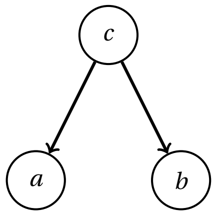


(b)

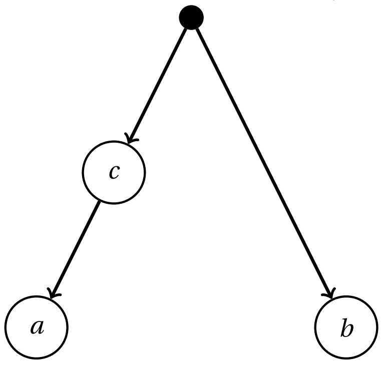
Figure 3.7: Two trees where the label d is placed at different vertices.



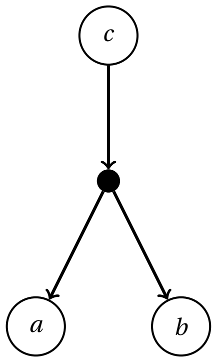
(a) A triplet of the form $a/b/c$ or equivalently $c\backslash b\backslash a$.



(b) A triplet of the form $a/c\backslash b$ or equivalently $b/c\backslash a$



(c) A triplet of the form $a/c|b$ or equivalently $b|c\backslash a$



(d) A triplet of the form $a/|c|\backslash b$ or equivalently $b/|c|\backslash a$

Figure 3.8: Additional triplets for general trees.

Proof.
Given the whole triplet set $t(T)$, using triplets $a\backslash b\backslash c$, $a\backslash b\backslash d$, and $c/b\backslash d$, the correct relation can be recovered. Namely, a has b, c and d as descendants, b has c and d as descendants, and c is in a different branch from d in the subtree rooted at b. Or equivalently, we know the triplet $c/|a|\backslash d$ also exists since a is an ancestor of b.

Moreover, if b were not labelled, then a had another child. Let $v \in X$ be that child (or one of its labelled descendants). Then $t(T)$ contains the triplets $cd|v$, $c/a\backslash v$, and $d/a\backslash v$. Again, these triplets suggest that c, d, and v are all descendants of a and that c and d must be in the same branch in the subtree rooted at a. So, we know the triplet $c/|a|\backslash d$ also exists. □

From now on, we assume $t(T)$ to contain all triplets of T according to Definition 2.6 except those as described in Observation 3.10.

Adding these additional triplets means the sets D_x no longer have any use. Namely, because it is now allowed

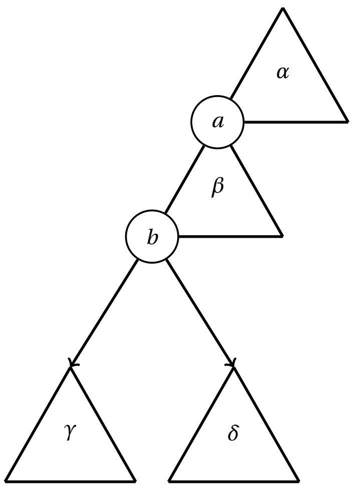


Figure 3.9: A tree where α, β, γ, and δ can be any trees that satisfies the assumptions.

for the labelled nodes in a triplet to be descendants of another, $D_x = \emptyset$ for all x in X as long as $|X| \geq 3$. Therefore, no knowledge about the structure of the tree can be learned from the sets D_x. Thus, a new approach to reconstruction must be created.

We will prove that the triplet encoding is unique for each tree, which satisfies the assumptions that any out-degree one node is labelled and points to another labelled node, and that all leaf nodes are labelled. Afterwards, an algorithm will be presented and proven to reconstruct the tree based on the triplets.

3.2.1. Unique triplet encoding

Induction on the number of labels will again be used to prove that triplets encode general trees. To do this, however, we need to again show that all triplets of a subtree are also triplets of the tree itself. Lemma A.1 and Theorem A.2 prove that when a path exists between two labels in a tree, then this path still exists in the tree after a different label is removed. It is presented in Appendix A.1.2. This result allows us to prove the above-mentioned statement.

Lemma 3.11. Let T be a tree on X, and $\tilde{T}$ be a subtree of T obtained by removing any number of labelled vertices, say $R \subseteq X$, as described in Definition 2.2. Then $t(\tilde{T}) \subseteq t(T)$ and for all $t \in t(T) \setminus t(\tilde{T})$ we have $t \cap R \neq \emptyset$.

Proof.
Suppose T is a tree and $\tilde{T}$ is a subtree of T. For a contradiction, assume $t(\tilde{T}) \not\subseteq t(T)$, then there exists a $t \in t(\tilde{T})$ such that $t \notin t(T)$. Without loss of generality, assume $u, v, w \in X$ form this triplet t. Then $T|_{u,v,w}$ is of a different form than $\tilde{T}|_{u,v,w}$. By Definition 2.6 there exists paths in $\tilde{T}$ such that t is consistent with $\tilde{T}$. By repeatedly applying Lemma A.1 and Theorem A.2 for all $x \in R$, similar paths still exist in T such that t is consistent with T. However, by assumption $t \notin t(T)$. Therefore, it must hold that $t(\tilde{T}) \subseteq t(T)$.

But by Lemma A.1 and Theorem A.2, there still exist paths in $\tilde{N}$ such that t is consistent with $\tilde{N}$. Thus $t \in t(\tilde{N})$, which leads to a contradiction. So for all $t \in t(N) \setminus t(\tilde{N})$ we have that $t \cap R \neq \emptyset$.

For the second statement, we will again show a contradiction. Suppose there exists a triplet $t = T|_{u,v,w} \in t(T) \setminus t(\tilde{T})$ such that t contains no nodes from R. Then all nodes of t are also in $\tilde{T}$. However, since $t \notin t(\tilde{T})$ it must be that $t \neq \tilde{T}|_{u,v,w}$. But by Lemma A.1 and Theorem A.2, there still exist paths in $\tilde{T}$ such that t is consistent with $\tilde{T}$. Thus $t \in t(\tilde{T})$, which leads to a contradiction. So for all $t \in t(T) \setminus t(\tilde{T})$ we have that $t \cap R \neq \emptyset$. □

Additionally, to apply induction, we need to show that a label can be removed such that the remaining trees are still non-isomorphic. Lemma 3.14 shows this result but requires the following theorem shown by Murakami (2021).

Theorem 3.12 (Murakami, 2021, Theorem 25)**.** Let N be a semi-binary tree-clone-free network. Then there exists only one automorphism of N, namely the identity map.

Here, a semi-binary network is a network such that all tree vertices (vertices with in-degree one and out-degree at least two) have degree at most 3. And a network is tree-clone-free if for every two tree vertices, at least one leaf node exists such that the paths between these tree nodes and the leaf node differ in length. However, this theorem can easily be extended to our definition of trees with only labelled leaves by the same reasoning used for the original proof, where we now consider tree vertices as vertices with in-degree one and out-degree at least one.

This theorem consequently shows that the automorphism of a general tree must also be unique.

Observation 3.13. Since the automorphism of a tree according to Definition 3.3 with only labelled leaves is unique, so must the automorphism of any tree according to Definition 3.3 also be unique.

Proof.
Let T be a tree and $\tilde{T}$ be the same tree but with all internal labels removed. Then the automorphism for $\tilde{T}$ is unique by Theorem 3.12, and any automorphism for T must also be an automorphism for $\tilde{T}$. Therefore, the automorphism for T is unique. □

Using this observation, we can now prove that for two non-isomorphic trees on at least four labels, there must always exist a label that can be removed such that the resulting trees are still non-isomorphic.

Lemma 3.14. Let $T_1 = (V_1, E_1, l_1)$ and $T_2 = (V_2, E_2, l_2)$ be two (partially-)labelled trees on the same set X, with $|X| \geq 4$, such that T_1 is non-isomorphic with T_2. Then there always exists a labelled node $x \in X$ such that $T_1 \setminus x$ is non-isomorphic with $T_2 \setminus x$.

Proof.
Let $\tilde{V_1}$ and $\tilde{V_2}$ be the vertex sets of $T_1 \setminus x$ and $T_2 \setminus x$, respectively, for a given x. If for any $x \in X$ no bijection exists between $\tilde{V_1}$ and $\tilde{V_2}$, due to a different number of vertices, then choose that x. So assume that for all $x \in X$ a bijection exists between $\tilde{V_1}$ and $\tilde{V_2}$. By Theorem 3.12 and Observation 3.13 for $T_1 \setminus x \cong T_2 \setminus x$ to hold, the bijection must map the leaf nodes of $T_1 \setminus x$ to leaf nodes with the same label in $T_2 \setminus x$. Thus, if for any $x \in X$ such a bijection does not exist, take that x. If for all $x \in X$ such a bijection does exist, we know by Theorem 3.12 this bijection must be unique for any given x and map the leaf nodes of $T_1 \setminus x$ to the leaf nodes of $T_2 \setminus x$ which have the same labels.

So assume that for all $x \in X$ such a bijection does exist for $T_1 \setminus x$ and $T_2 \setminus x$ and thus $(T_1 \setminus x)' \cong (T_2 \setminus x)'$, where $(T_1 \setminus x)'$ and $(T_2 \setminus x)'$ denote the trees without labels. Thus, we can conclude that T_1 and T_2 must have the same number of leaves with the same labels. To show that $T_1 \setminus x \ncong T_2 \setminus x$ for some x we need to find an x such that there is an internal label $y \in X$ for which $\tilde{f}(\tilde{l_1}(y)) \neq \tilde{l_2}(y)$. Where we denote $\tilde{f}, \tilde{l_1}$, and $\tilde{l_2}$ as the obtained bijection and injections after removing a label x.

By Definition 2.5 we know either (i) $T_1' \ncong T_2'$ or (ii) $T_1' \cong T_2'$ but there exists no corresponding bijection such that $f(l_1(x)) = l_2(x)$ for all $x \in X$.

(i) Harary and Palmer (1966) proved that any rooted tree is reconstructible based on its maximum subtrees, or equivalently, that any tree T is encoded by the subtrees $T_i = T - v_i$ where v_i are the leaves of T. Note that $T - v_i$ differs from $T \setminus v_i$ as $T - v_i$ is obtained by removing the node v_i and its edges without "cleaning up" the graph afterwards. However, "cleaning up" is only required after removing a leaf node if its parent was unlabelled and had out-degree two. So if the number of leaf nodes is at least three, the subtrees $T - v_i$ can be recovered from $T \setminus v_i$. Namely, if removing v_i causes a contraction and its parent p_i has two leaf descendants, take another leaf that is not a leaf descendant of p_i, say v_j. Then $T \setminus v_j$ preserved p_i as a vertex. Otherwise, if p_i has three or more leaf descendants, take any leaf descendant different from v_i, say v_j. Then, again, $T \setminus v_j$ preserved p_i as a vertex. So if T_1 and T_2 have three or more leaf nodes, it cannot hold that $T_1' \ncong T_2'$ while $(T_1 \setminus x)' \cong (T_2 \setminus x)'$ for all $x \in X$.

    Now, if T_1 and T_2 have one or two leaf nodes, then regardless of the trees' structures, the parents of the leaf nodes are labelled since $|X| \geq 4$. Thus $T - v_i$ is the same as $T \setminus v_i$ for all leaves v_i of T_1 and T_2. Therefore, $T_1' \cong T_2'$,

(ii) Since T_1 and T_2 share the same number of leaves with the same labels and $T_1' \cong T_2'$, we know there exists a bijection that maps the leaf nodes of T_1 to the leaf nodes of T_2 with the same labels. Thus there must be an internal label $y \in X$ such that $f(l_1(y)) \neq l_2(y)$. By Theorem 3.12 we know both f and $\tilde{f}$ are unique. So

$l_1(y) = v_1$ and $l_2(y) = v_2$ for some $v_1 \in V_1$ and $v_2 \in V_2$. If there is a node $x \in X$ such that removing x does not cause a contraction of an edge starting from v_1 or v_2 in T_1 and T_2 respectively, then removing x will still result in $\tilde{f}(\tilde{l}_1(y)) \neq \tilde{l}_2(y)$ since $\tilde{l}_1(y) = v_1$ and $\tilde{l}_2(y) = v_2$ still hold and $\tilde{f}(v_1) \neq v_2$. If there is no such node, then all labels are the only child of y in either T_1 or T_2 and are leaves, or are a child of y such that y has out-degree two and the other child of y is unlabelled. But since $|X| \geq 4$, this is not possible. So there always exist such an $x \in X$ such that $\tilde{f}(\tilde{l}_1(y)) \neq \tilde{l}_2(y)$ in $T_1 \setminus x$ and $T_2 \setminus x$.

It has been shown that (i) cannot hold, and if (ii) holds, a labelled node $x \in X$ can be found such that (ii) still holds for $T_1 \setminus x$ and $T_2 \setminus x$.

Therefore, by Definition 2.5 there exists an $x \in X$ such that $T_1 \setminus x \ncong T_2 \setminus x$. □

Intuitively, we can imagine that between two non-isomorphic trees, there is some relation between two labels different in both trees. This differing relation should be possible to keep when removing other labels. Take, for example, the trees in Figure 3.3. Here, the relation between a and d is different in both trees. In Figure 3.3a a is the parent of d, while in Figure 3.3b a is a sibling of d. Indeed, removing any of the other labels keeps this differing relation in both trees and gives two smaller non-isomorphic trees.

Using Lemmas 3.11 and 3.14 we can now prove that triplets encode general trees.

Theorem 3.15. Let T_1 and T_2 be two trees on the same set of labelled nodes X, with $|X| \geq 3$. Then T_1 and T_2 are isomorphic if and only if $t(T_1) = t(T_2)$

Proof.
For the first direction, suppose $T_1 \cong T_2$. Then for all $u, v \in X$, their least common ancestor is the same in both trees. Thus, the triplet induced by any three labelled vertices in X will be of the same form in T_1 as in T_2 since the form is determined by the least common ancestors alone. Thus $t(T_1) = t(T_2)$.

For the other direction, the contrapositive will be proven through induction. Let T_1 and T_2 be two trees on X, with $|X| = 3$. Without loss of generality, assume $X = \{u, v, w\}$. Then T_1 and T_2 can only be of the forms $u|v|w$, $uv|w$, $u/v|w$, $u/v/w$,$u/v\backslash w$, or one of its variants. In any case, the triplet sets of T_1 and T_2 will only contain one triplet that is the tree itself. Therefore, if $T_1 \ncong T_2$, their triplet sets must be different.

To apply induction, suppose we have proven the claim for all $|X| \leq n-1$, with $n \geq 4$. Let T_1, T_2 be two trees on X, where $|X| = n$, such that $T_1 \ncong T_2$. Then take any $x \in X$ such that $T_1 \setminus x \ncong T_2 \setminus x$. By Lemma 3.14, there always exists such a labelled node. By the induction hypothesis, we have that $t(T_1 \setminus x) \neq t(T_2 \setminus x)$. And by Lemma 3.11 we obtain $t(T_1) \neq t(T_2)$. □

3.2.2. Algorithm

Different from Section 3.1.2, the problem presented in this section has not yet been researched. We wish to find a tree satisfying Definition 3.3 that contains a given set of triplets, if it exists. Our proposed algorithm works by identifying a label, if it exists, as the root of the tree and then separating the remaining labels in their respective branches. It works similarly to the algorithm proposed by Aho et al. (1981). However, our algorithm differs by being able to include internal vertices, labelled out-degree one nodes, and a labelled root. Just like Algorithm 1, the algorithm proposed here is also able to find a tree that contains a subset of a triplet set. Again, this found tree does not have to be unique for a subset of triplets. The algorithm has $\mathcal{O}(|t(T)|^2|X|^3)$ theoretical running time.

Lemma 3.16 shows the triplets a label can be in if it is the root, and Lemma 3.17 shows how we can determine if labels are in the same branch based on a triplet.

Lemma 3.16. Let T be a tree on X, with $|X| \geq 3$, and take $u \in X$. Then u is the root of T if and only if for all $t \in t(T)$ containing u t is of the form $u\backslash v\backslash w$ or $v/u\backslash w$ with $v, w \in X$.

Proof.
First, suppose u is the root of T. Then, any triplet containing u must have u as the ancestor of all other nodes. Thus, triplets containing u can only be of the form $u\backslash v\backslash w$ or $v/u\backslash w$.

For the other direction, to prove the contrapositive, suppose u is not the root of T. Then the root has either an out-degree of one or more.

If the root has out-degree one, then the root is labelled, say v. Since $|X| \geq 3$, there must be another labelled node, say w. Any triplet containing u, v, and w can therefore not be of the form $v/u\backslash w$, $u \setminus w \setminus v$, or $u \setminus v \setminus w$.

If the root has out-degree two or more, then the root can be labelled or not. If it is labelled, take v to be the root and w to be a labelled node in a branch not containing u. Then the triplet containing u, v, and w is of the form $u/v\backslash w$. If the root is not labelled, take v to be any labelled node in a branch not containing u, and w to be any other labelled node. Then the triplet containing u, v, and w must have u separated from v by the root. Thus, the triplet will be, for example, of the form $uw|v$, $w/u|v$, $u|w\backslash v$, etc.

In both cases, we have that there exists triplets containing u that are not of the form $u\backslash v\backslash w$ or $v/u\backslash w$. □

In essence, Lemma 3.16 simply states that the root of a tree cannot be in a triplet such that it is not an ancestor of the other labels. Algorithm 8 finds the labels that satisfy Lemma 3.16, and is a direct implementation of the lemma and will therefore not be proven. It runs in $\mathcal{O}(|X|^2 + |t(T)|^2)$. Algorithm 8 can be found in Appendix A.1.2.

Lemma 3.17 shows how the triplets define whether or not labels are in the same branch or different ones.

Lemma 3.17. Let T be a tree according to Definition 3.3. Then for all triplets t_i in $t(T)$ the following statements hold:

(i) If t_i is of the form $u/v/w$, then u, v, and w are all in the same branch B_i if w is not the root of T. If w is the root of T, then u and v are still in the same branch B_i.

(ii) If t_i is of the form $u/v\backslash w$, then if v is not the root of T u, v, and w are all in the same branch B_i. If v is the root of T, then u and w are in two different branches B_i and B_j with $i \neq j$.

(iii) If t_i is of the form $u/v|w$, then u and v are in the same branch B_i.

Proof.
Each statement will be proven through contradiction.

(i) Suppose there is a triplet $u/v/w$ such that $u \in B_i$ and $v \in B_j$ with $i \neq j$. Then v cannot be an ancestor of u as they are in different branches. This makes the triplet $u/v/w$ impossible. If w is not the root, the same reasoning holds as above.

(ii) Suppose there is a triplet $u/v\backslash w$ where v is not the root of T such that u and w are not in the same branch. Then LCA(u, w) is the root of T and the only triplet of the form $u/*\backslash w$ must be $u/r\backslash w$ where r is the root of T. However, since v was not the root, the triplet $u/v\backslash w$ is impossible. v must also be in the same branch by the reasoning in (i).

Suppose there is a triplet $u/v\backslash w$ where v is the root of T such that u and w are in the same branch. Then LCA(u, w) is not the root, and thus a triplet of the form $u/*\backslash w$, if it exists, would not be $u/v\backslash v$.

(iii) Suppose there is a triplet of the form $u/v|w$ such that u and v are in two different branches. Then v cannot be an ancestor of u. This makes the triplet $u/v|w$ impossible.

In each case, a contradiction was reached. □

Algorithm 10 places the labels in the correct branches by iterating over the triplets and using their descendants. The algorithm is a direct implementation of Lemmas 3.7 and 3.17 and will therefore not be proven directly. Note that Lemma 3.7 still holds by the same reasoning. It uses Algorithm 9 to properly handle fanned triplets and runs in $\mathcal{O}(|t(T)|^2|X|)$ time. Algorithms 9 and 10 can be found in Appendix A.1.2.

Algorithm 2 can reconstruct a general tree based on the triplets or find a tree that satisfies the triplets if a subset of triplets has been used. It uses several properties of the triplets. Namely, the descendants given by triplets such as $u/v|w$ or $u/v \setminus w$, as well as the separation between nodes, such as the separation between u and w in $uv|w$. These descendants and separations are computed using Algorithm 7 in $\mathcal{O}(|t(T)|+|X|)$ time. Algorithm 2 runs in $\mathcal{O}(|t(T)| + |X| + D(|X| + |t(T)| + |t(T)|^2|X|^2))$, where D is the depth of the tree which is bounded by $|X|$. Using the bound for D and simplifying the runtime gives $\mathcal{O}(|t(T)|^2|X|^3)$. If, however, we assume we have a full triplet set, then Line 6 of Algorithm 2 is run at most once every iteration, as at most one label can be returned by Algorithm 8. So in that case, the theoretical running time is $\mathcal{O}(|t(T)|^2|X|^2)$. Algorithm 7 can be found in Appendix A.1.2.

Theorem 3.18. Let T be a tree on X with triplet set $t(T)$. Let $t'(T)$ be a subset of $t(T)$, and $\tilde{T}$ be the output from Algorithm 2 with input $(t'(T), X)$. Then $\tilde{T}$ contains all the triplets in $t'(T)$. Moreover, if $t'(T) = t(T)$ then $\tilde{T} \cong T$.

Proof.
We will prove the theorem through induction. Without loss of generality, let T be a tree on $X = \{u, v, w\}$, where $|X| = 3$. Then either, without loss of generality, $t'(T) = \{u|v|w\}$, $t'(T) = \{u \setminus v \setminus w\}$, $t'(T) = \{u/v \setminus w\}$, $t'(T) = \{u/v|w\}$, $t'(T) = \{uv|w\}$, or $t'(T) = \emptyset$.

In all cases, it is easy to see we end up with a $\tilde{T}$ that satisfies the triplet set $t'(T)$ by following the logic of the algorithms.

To apply induction, suppose that the theorem holds for all trees with $|X| \leq n-1$, and let T be on $|X| = n$. We will show that an arbitrary node is placed in the correct branch, and then apply induction on the branches. Let $x \in X$ be any node. Then either (i) x is the root or (ii) it is not.

(i) If x is the root by Lemma 3.16 every triplet containing x is of the form $x \setminus u \setminus v$ or $u/x \setminus v$. Therefore, by the logic of Algorithm 8, $x \in R$ since x is indeed the root of T. If $R = [x]$, we are done. Namely, x will be chosen as the root, and the branches will be created by Line 6 in Algorithm 2, and the branches will be created correctly by the reasoning of (ii). If $\{x\} \subset R$, either (a) x is eventually chosen, or (b) another $y \in R$ is chosen as the root.

   (a) If x is chosen, then by the same reasoning as above, the remainder will be correctly divided in the different branches.

   (b) Then, by Line 8 in Algorithm 2, no triplet exists of the form $a/y \setminus b \in t'(T)$ such that a and b are placed in the same branch by Algorithm 10. Therefore, either a (and its descendants) and/or b (and its descendants) do not share a triplet with x. If they both did, $\{a, b\} \subseteq D[x]$ and thus by Line 12 in Algorithm 10 they would have been placed in the same branch. By this reasoning, all labelled nodes that share a triplet with x will be placed in a branch with x until (for some subtree) x is chosen as the root.

(ii) If x is not the root, it must be in some branch B_i. In the algorithms, either (a) $x \in R$, or (b) $x \notin R$.

   (a) If $x \in R$, then, by Algorithm 8, $S[x] = \emptyset, x \notin D[u]$ for all $u \in X$. Therefore, we know no triplet exists with x and any $z \in X \setminus B_i$ nor with any $y \in B_i$ such that x is a descendant of y. As a consequence, we can split X into two disjoint sets X_1 and X_2. Here, X_1 contains x and all labels that share a triplet with x, and X_2 contains all remaining labels. Note $X \setminus B_i \subseteq X_2$. Then no triplet t exists with $x, u, v \in t$ such that $u \in X_1$ and $v \in X_2$. Thus, placing x as the root will not cause any triplets to be impossible to make. Namely, $X_1 \setminus x$ will be placed in a different branch from X_2 by Line 12 of Algorithm 10. Since $X_2 \cap D[x_1] = \emptyset$ for all $x_1 \in X_1$ and $X_1 \cap D[x_2] = \emptyset$ for all $x_2 \in X_2$.

   (b) If $x \notin R$, then x will be placed in a branch together with all its descendants and ancestors (except for the root) based on the triplets by Algorithm 10.

   In all cases, we see that x is placed such that all triplets can still be formed properly.

Since x is arbitrary, we know all nodes are placed such that the triplets are possible to make. Now $|B_i| \leq n-1$ and therefore by the induction hypothesis, these branches can be properly resolved by the algorithm. Thus the output $\tilde{T}$ of Algorithm 2 with input $(t'(T), X)$ will contain all triplets in $t'(T)$.

For the second statement, if $t'(T) = t(T)$ then $\tilde{T}$ will contain all triplets in $t(T)$ and thus by Theorem 3.15 $\tilde{T} \cong T$ □

Theorem 3.19. Let $t'(\tilde{T})$ be some set of triplets. If there is a tree that follows Definition 3.3, T, satisfying the triplets $t'(\tilde{T})$, then Algorithm 2 returns a tree. Otherwise, no tree is returned.

Proof.
We will prove the theorem through a contradiction. Suppose the theorem does not hold and that T that satisfies the triplet set $t'(\tilde{T})$ is as small a counterexample as possible such that no tree is returned by Algorithm 2. Then by Line 16 of Algorithm 2, either only one branch was returned from the root down while the root was unlabelled, or for some internal unlabelled node, only one branch was returned by Algorithm 10.

In the first case, the branches are divided by Line 14 in Algorithm 2 and only one branch is returned. In Algorithm 10, either (i) Line 11 merges all branches into one and no further branches are created, (ii) Line 19 merges all branches into one, or (iii) Line 21 merges all branches into one.

(i) In this case there were two branches, B_1, B_2, such that for some $x, u, v \in X$, $u \in B_1, v \in B_2$ and $u, v \in D[x]$. But then there must be a node in X that is an ancestor of all other nodes. Therefore, that node would have been made the root, which contradicts the assumption that the root was unlabelled.

(ii) In this case a triplet $uv|w$ existed such that two branches, B_1, B_2, were present and $u \in B_1$ and $v \in B_2$. Then Line 19 merges B_1 and B_2 into one branch. But then, w was also placed in either B_1 or B_2 already. Suppose, without loss of generality, $w \in B_1$. Then either w has a descendant that is also a descendant of u or there exists a $y \in B_1$ such that y has both u and w as descendants. The first case is not possible because of the triplet $uv|w$. The second case implies that either y is the root of T or there is another branch not containing y. Clearly, y cannot be the root of T since otherwise y would have been chosen as the root. Also there cannot be a second branch not containing y since then we would not be left with just one branch.

(iii) Now by Line 6 in Algorithm 9, without loss of generality, there were two branches, B_1, B_2, and a triplet $u|v|w$ exists such that $u, v \in B_1$ and $w \in B_2$. But this is only possible if indeed u, v, and w are supposed to be in the same branch from the root down. Which in turn implies that either the root is labelled or there is a $y \in X$ such that y is in a different branch from u, v, and w. The first case is not possible as in that case this label would have been chosen to be the root. The second case implies that either no triplets exist containing y, in which case y would have been chosen as the root, or $D[y] \cap (B_1 \cup B_2) = \emptyset$. But then, by Line 12 in Algorithm 10 y would have been placed in a separate branch, and thus we would not have ended up with one branch.

In all cases, we see that the triplet set must be contradictory.

In the second case, the subtree rooted at this unlabelled internal node is a smaller counterexample. This contradicts the fact that T was the smallest counterexample.

Thus no T can exist that satisfies $t'(\tilde{T})$. □

We will, again, discuss a small example of the algorithm. We assume T to be the tree in Figure 3.10.

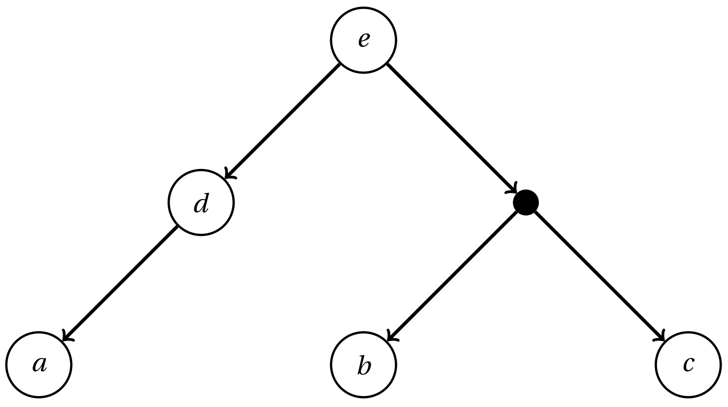


Figure 3.10: The tree T used for an example of the steps in the algorithm.

After computing D and S, we look at the possible roots. Algorithm 8 will return $\{e\}$. Therefore, we try e as the root and divide the branches. The two branches obtained by Algorithm 10 are $\{a, d\}$ and $\{b, c\}$. Since no triplet of the form $u/e \setminus v$ exists such that u and v are in the same branch, we choose e as the root.

The two branches are then solved directly since their size is two. For the first branch, $\{a, d\}$, we have that $a \in D_d$ and thus d is added as a child of e with a as its child. For the second branch, $\{b, c\}$, neither label is in the other's D set. Therefore, an unlabelled child is added to e, which has both b and c as its children. This gives us the tree as in Figure 3.10.

Algorithm 2: General tree reconstruction

```
Function BuildTree($t'(T), X$):
    Input: $t'(T)$ — a set of triplets
    Input: $X$ — the labelled nodes of T
    Output: $T$ — a tree such that it contains all triplets in $t'(T)$ if it exists
    $D, S$ = GetDescendantsAndSeperation($t'(T), X$) using Algorithm 7
                                        // Line is only called in the first iteration
    $R$ = PossibleRoots($t'(T), X, D, S$) using Algorithm 8
    while $|R| \geq 1$ do
        Choose $r \in R$
        $B_1, B_2, ..., B_k$ =DivideBranches($t'(T), X \setminus \{r\}, D$) using Algorithm 10
        if for any triplet of the form $a/r \setminus b, a$ in the same branch as $b$ then
            // The branches do not agree with the triplets
            $R$.remove($r$)
        else
            $T$ has $r$ as root
            break
    else
        // No labelled node can be the root
        $T$ has an unlabelled root
        $B_1, B_2, ..., B_k$ =DivideBranches($t'(T), X, D$) using Algorithm 10.
        if $k = 1$ then
            raise Error                              // The triplets are contradictory
    forall $B_i$ do
        if $|B_i| = 1$ then
            $T$ has the node in $B_i$ as a child
        else if $|B_i| = 2$ then
            $u, v \in B_i$
            if $u \in D[v]$ then
                $T$ has $v$ as a child and $u$ is a child of $v$
            else if $v \in D[u]$ then
                $T$ has $u$ as a child and $v$ is a child of $u$
            else $T$ has an unlabelled child with $u$ and $v$ as children
        else
            $T$ has BuildTree($\{t_i \in t'(T) | t_i \subseteq B_i\}, B_i$) as a child
    return $T$
```

4 Level-1 Networks

In the previous chapter, we formulated how rooted trees show the vertical descent relationships among texts (or, in a phylogenetic context, among species). However, in many practical stemmatological scenarios, a scribe may consult - or "contaminate from" - multiple sources when copying a manuscript. Such contamination events cannot be captured by a rooted tree, since they do not allow nodes to have in-degree two or higher. Instead, a network is required to represent simultaneous inheritance from multiple predecessors. We already formally defined such networks in Definition 2.9.

Roelli (2020) defines two different kinds of contamination: simultaneous and successive. Simultaneous contaminations occur when a scribe uses several sources for the same part of a text. It occurs naturally when a scribe wants to correct a given text by consulting another (older) text to compare. Successive contaminations occur when a scribe uses different sources for different parts of a text. This can occur when the scribe has incomplete sources or the quality of a given source is inconsistent. A mixture of these types of contaminations is also possible. Networks are capable of representing such events by giving a labelled node more than one parent.

Because networks can introduce cycles (in the underlying undirected graph), the one-to-one correspondence between any three labels and at most a single rooted triplet present in trees no longer holds. Indeed, given three labels $u, v, w \in X$, there may be multiple triplets on those labels that are consistent with N. Therefore, let $N|_{u,v,w}$ for $u, v, w \in X$ denote the set of all triplets of N containing only these labels.

An inverted cherry, as shown in Figure 4.1, is not included in $t(N)$ as it is not a rooted tree. Nor do we include triplets as described in Observation 3.10.

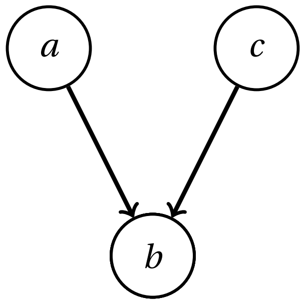


Figure 4.1: A tree of the form $a \setminus b / c$ or equivalently $c \setminus b / a$ that is not included as a triplet as it is not a rooted tree.

To allow for such contamination events, this chapter focuses on the simplest network class: level-1 networks. A level-1 network N is a directed acyclic graph in which every undirected biconnected component contains at most one cycle. According to Gusfield et al. (2004), these types of networks can represent the most common phylogenies as biological mutations are not as frequent. However, in stemmatology, contaminations are more common and therefore higher-level networks might be needed to properly represent a stemma (Roelli, 2020). However, we will only consider the triplet encoding for level-1 networks.

For level-1 networks with at least three leaves, only labelled leaves, and all internal vertices having degree three, Gambette and Huber (2012) have proven that the number of non-isomorphic networks that share the same triplet sets is 3^b. Here b is the number of biconnected components of N with four vertices. Indeed Figure 4.2 shows a simple example with $|X| = 3$ and $b = 1$ where the triplet set of each network is $\{ab|c, a|bc\}$.

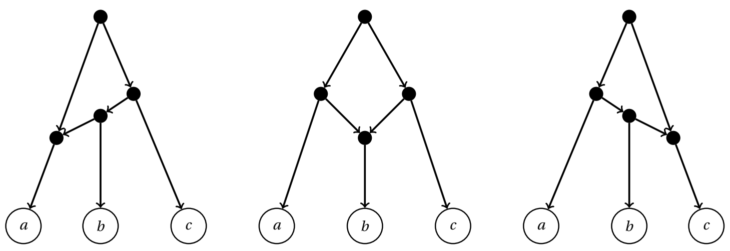


Figure 4.2: Three different networks by Gambette and Huber (2012) that share the same triplet set.

It can be noted that the networks contain unlabelled internal nodes with out-degree one. As discussed in Section 3.2, these nodes do not contain any information we would be able to differentiate between and thus should be contracted. This does not influence the outcome of Gambette and Huber (2012), as they do not change the triplet sets.

For this chapter, we assume all networks to follow Definition 4.1.

Definition 4.1. We call $N = (V, E, l)$ a *(partially-)labelled level-1 stemmatic network* if the following holds:

1. N is a network according to Definition 2.9
2. All out-degree one nodes are labelled and point to a labelled node
3. Every undirected biconnected component of N contains at most one cycle
4. Every cycle in the underlying undirected graph of length 3 or 4 contains at least one labelled vertex that is not the source nor the sink
5. The sink of every cycle in the underlying undirected graph is labelled.

Conditions 4 and 5 prevent "triangles" or "squares" in a network whose triplet sets cannot uniquely encode it. Indeed when "cleaning up" the networks in Figure 4.2 according to Definition 2.2 and making a vertex of the cycles that is not the sink nor the source labelled we obtain the networks in Figure 4.3 which all have different triplet sets.

In Section 4.1 we will prove that the triplets resulting from these networks encode the network. This extends the triplet encoding result for trees to level-1 networks. Moreover, a polynomial-time algorithm will be presented that is able to reconstruct a network based on its full triplet set. Unlike the previous algorithms, this algorithm is not always able to find a network for a subset of triplets.

4.1. Unique triplet encoding

Unlike we did in Sections 3.1.1 and 3.2.1, we will prove that triplets encode level-1 networks by showing no smallest counterexample exists. We will still need that all triplets in a subnetwork are also triplets in the network itself. Moreover, we will also introduce a new concept called *SN* sets, which allows us to more easily separate specific segments of a network.

Note that Lemma A.1 and Theorem A.2, which state that a path between any two nodes still exists when a different node is removed from the network, also holds for networks by the same reasoning. Therefore, we can also use it when proving the following lemma.

Lemma 4.1. Let N be a network on X, and $\tilde{N}$ be a subgraph of N obtained by removing any number of labelled vertices, say $R \subseteq X$, as described in Definition 2.2. Then $t(\tilde{N}) \subseteq t(N)$ and for all $t \in t(N) \setminus t(\tilde{N})$ we have $t \cap R \neq \emptyset$.

Proof.
The proof follows the same reasoning as Lemma 3.11. □

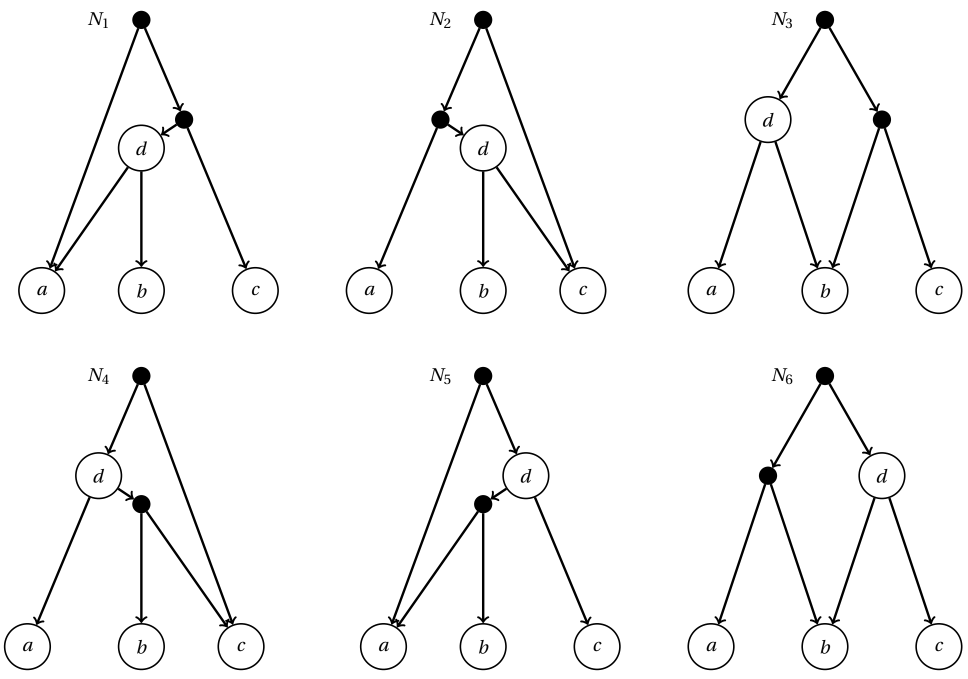


Figure 4.3: The networks from Figure 4.2 altered to comply with Definition 4.1. Note that the triplet sets are different:

$$\begin{aligned}
t(N_1) &= \{ab|c,\ a|bc,\ a|d\backslash b,\ a/d|c,\ b/d|c,\ a/d\backslash b,\ a|cd\ \},\\
t(N_2) &= \{ab|c,\ a|bc,\ a|d\backslash b,\ a|d\backslash c\ \ b/d|c,\ b/d\backslash c,\ ad|c,\ \},\\
t(N_3) &= \{ab|c,\ a|bc,\ a/d|b,\ a/d|c,\ b/d|c,\ a/d\backslash b,\ bc|d\ \},\\
t(N_4) &= \{ab|c,\ a|bc,\ a/d\backslash b,\ a/d|c,\ b/d|c,\ a/d\backslash c\ \},\\
t(N_5) &= \{ab|c,\ a|bc,\ a|d\backslash b,\ a|d\backslash c,\ b/d\backslash c,\ a/d\backslash c,\ \},\\
t(N_6) &= \{ab|c,\ a|bc,\ a|d\backslash b,\ a|d\backslash c,\ b/d\backslash c,\ ab|d,\ b|d\backslash c\},
\end{aligned}$$

The concept of SN sets was first introduced by Jansson and Sung (2006). They used it to partition the leaf labels of a level-1 phylogenetic network into disjoint subsets, which could then be solved separately to reconstruct such a network. Labels in an SN set correspond to a subnetwork, which is why they named those sets "SN sets". We will also show that these sets correspond to specific segments of a given network and use this correspondence to get to our desired results.

In the paper of Jansson and Sung, SN sets are defined recursively for any $L \subseteq X$ of a network N as $SN(L \cup \{c\})$ for $c \in X \setminus L$ if there exists some triplet of the form $xc|x'$ for $x, x' \in L$ and as L otherwise. To extend the concept for stemmatic networks, we can use the following definition.

Definition 4.2. Let $N = (V, E, l)$ be a network on X and $L \subseteq X$. Then the SN set on L, $SN(L)$, is defined as $SN(L \cup c)$ for any $c \in X$ if there exists a triplet in $t(N)$ such that c is a descendant of LCA(x, x') or LCA(x, x') itself in a triplet in $N|_{x,x',c}$ for any $x, x' \in L$ and as L otherwise. If $|L| = 1$, then $SN(L) = L$.

$SN(\{x_1, x_2\})$ with $x_1, x_2 \in X$ is called trivial if it equals X and maximal if it is non-trivial and is not a proper subset of another non-trivial $SN(\{y_1, y_2\})$ with $y_1, y_2 \in X$.

van Iersel et al. (2009) proved that for phylogenetic networks with only labelled leaves, a maximal SN set corresponds to the leaves below a highest cut-arc. We will argue that a similar statement holds for level-1 stemmatic networks with internal labels.

Observation 4.2. Any non-trivial SN set containing the labelled root of N contains only the root.

Proof.
Suppose S is an SN set containing the labelled root $p \in X$ and another labelled node $x \in X$. Then $\text{LCA}(p, x) = p$ and thus any other label $y \in X$ is a descendant of p, so $y \in S$. Therefore, $S = X$ and thus S is trivial. □

Lemma 4.3. Let N be a (partially-)labelled level-1 stemmatic network. Any non-trivial SN set of N corresponds to a single label, the union of all labels underneath a cut-arc, or the sink and its descendants of a cycle.

Proof.
By Definition 4.2 any $\{x\}$ is an SN set for $x \in X$.

Let L be the union of all labels underneath a cut-arc. Then any triplet containing labels $x, x' \in L$ and $y \notin L$ would have x and x' separated from y by this cut-arc or y as an ancestor of both x and x'. Thus $y \notin SN(L)$.

Lastly, let L be the union of the sink of a cycle and all labels underneath it. Then, since N is a level-1 network, any triplet containing labels $x, x' \in L$ and $y \notin L$ will have y separated from x and x' by a node above the sink, or y is an ancestor of both x and x'. So $y \notin SN(L)$.

For the other direction, we will come up with a contradiction. So suppose $L \subset X$ such that $|L| \geq 2$ and $x, x' \in L$ are not both beneath a cut-arc nor part of a sink and its descendants of a cycle. By Observation 4.2 we may assume the labelled root, if it exists, is not in L.

If the root of N is not the source of a cycle, then the root must have out-degree two or higher. Therefore, $x \in B_i$ and $x' \in B_j$ for some $i \neq j$. Thus $\text{LCA}(x, y)$ is the root and therefore any $z \in X$ is in $SN(L)$. Then $SN(L) = X$, so $SN(L)$ is trivial.

If the root of N is the source of a cycle C, then either $x \in B_j^C$ and $x' \in B_i^C$ or $x, x' \in B_j^C$ but the corresponding cycle vertex has out-degree three or higher and B_j^C is not the sink of C.

In the first case, $\text{LCA}(x, x')$ is either the root of N, in which case we are done, or it is a cycle vertex of C. If it is a cycle vertex of C, then there exists a triplet with x, x' (or one of their labelled descendants) and the sink of C such that the sink is placed in $SN(L)$. Now take the sink s, and x (or x') different from the sink. Then there exists a triplet with x (or x'), s and any other label in X such that $\text{LCA}(x, s)$ (or $\text{LCA}(x', s)$) is the root in some triplet. So $SN(L) = X$ and thus trivial.

In the second case, if the cycle vertex is not the source of another cycle there exists a triplet with x, x' and the sink of C, s, such that s is a descendant of $\text{LCA}(x, x')$ and thus $s \in SN(L)$. By the same reasoning as before, $SN(L) = X$ and thus trivial. If the cycle vertex is the source of another cycle, then by iteratively applying the reasoning above, this whole lower cycle is in $SN(L)$. We can then obtain a triplet containing the sink of the highest cycle (with the root as its source), such that that sink is in $SN(L)$, after which the same reasoning holds. □

Now that we know what these SN sets can correspond to, we can also explore what the maximal SN sets can correspond to. After all, the correspondence of any non-trivial SN set does not give much information about how and where these segments might be located in the network.

Corollary 4.4. Let N be a (partially-)labelled level-1 stemmatic network. A maximal SN set of N corresponds to the labelled root itself, the union of all labels underneath a highest cut-arc, a highest sink of a cycle with no cut-arc above it and its labelled descendants, or a labelled non-sink cycle vertex with no cut-arcs or sinks above it.

Proof.
Clearly, by Lemma 4.3, these sets are all SN sets.

Suppose S is a maximal SN set and $L \subset S$, where L is any set as described in the corollary and take $l \in L$. Then $x \in S \setminus L$ is such that $\text{LCA}(x, l)$ is either the root of N or a cycle vertex of a cycle such that the sink of that cycle is a highest sink with no cut-arcs above it. By the same reasoning as in Lemma 4.3, $S = X$ must hold and therefore S is not maximal. □

Together with the previous results, we are now able to prove the main result of this thesis: triplets encode a level-1 stemmatic network. To do so, we will show that no two smallest networks exist such that they are non-isomorphic but share the same triplet sets. The proof relies on several claims concerning the structure of the supposed counterexample, which will be proven as well. Although the proofs of these claims are rather extensive, they are necessary for the final result.

Theorem 4.5. Let N_1 and N_2 be two (partially-)labelled level-1 stemmatic networks on the same set of labelled nodes X, with $|X| \geq 3$. Then N_1 and N_2 are isomorphic if and only if $t(N_1) = t(N_2)$.

Proof.
For the first direction, since $N_1 \cong N_2$, any triplet in $t(N_1)$ is also consistent with N_2 by Definition 2.6 and thus is also in $t(N_2)$. So $t(N_1) \subseteq t(N_2)$. Likewise, any triplet in $t(N_2)$ is also consistent with N_1 by Definition 2.6 since $N_1 \cong N_2$. Therefore, also $t(N_2) \subseteq t(N_2)$. We can thus conclude $t(N_1) = t(N_2)$.

For the other direction, we will come up with a contradiction. Let N_1 and N_2 be two networks on X such that they form the smallest counterexample on the size of X. So $N_1 \ncong N_2$ and $t(N_1) = t(N_2)$.

We will first prove that N_1 and N_2 must contain exactly one cycle since it is the smallest counterexample, and that their roots must be the sources of these cycles. Then, we will prove these networks must actually be isomorphic for $t(N_1) = t(N_2)$ to hold.

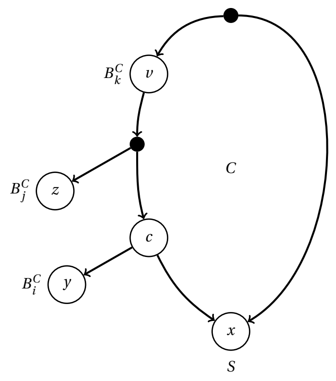
Claim 4.5.1. N_1 and N_2 must both contain at least one cycle.

Proof.
Suppose N_1 had no cycles, then if N_2 also did not have any cycles by Theorem 3.15, they would have different triplet sets. Therefore, at least one network has at least one cycle. Assuming N_1 has a cycle it would have $a, b, c \in X$ such that $|N_1|_{a,b,c}| = 2$. Namely, if the cycle has length 4 or larger, take a to be the sink and b and c such that they are in distinct branches different from the sink and its descendants, and either both are in the cycle or both are not in the cycle. Then two triplets exist with a, b, and c: $a/b/c$ and $a|c \setminus b$, $a/b|c$ and $a/c|b$, $ab|c$ and $a|bc$, or $ab|c$ and $ac|b$. If the cycle has length 3, take a to be the sink and b the labelled node in the cycle that is not the source nor the sink. Then take c any other node in X that is not an ancestor of the source of the cycle. Again two triplets exist with a, b, and c: $c/a/b$ and $c/a|b$, $a/b \setminus c$ and $a|b \setminus c$, $a/b/c$ and $a/c \setminus b$, or $ab|c$ and $a/b|c$. If N_2 would have no cycles then $|N_2|_{a,b,c}| \leq 1$. Thus, both N_1 and N_2 must contain cycles. ■

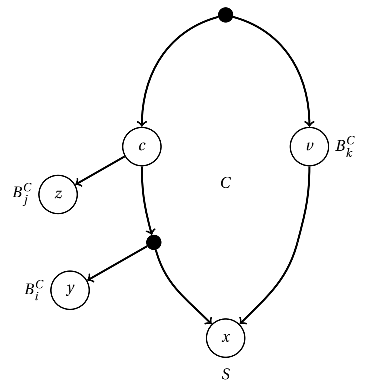
Claim 4.5.2. N_1 and N_2 each contain exactly one cycle.

Proof.
By Claim 4.5.1, we know N_1 and N_2 both contain at least one cycle. Suppose N_1 contains at least two cycles. Since $t(N_1) = t(N_2)$, their SN sets must be the same, as they are determined by the triplet sets alone. Therefore, let S be the sink and its descendants of a cycle C in N_1 such that not all other cycles are in S. By Lemma 4.3, S is an SN set of N_1. Then, by Lemma 4.1, $t(N_1 \setminus S) = t(N_2 \setminus S)$ and we know $N_1 \setminus S$ is a valid network with fewer cycles. Since their SN sets are the same, S is also an SN set in N_2. By Lemma 4.3, S contains either a sink and its descendants, a single label, or all nodes below a cut-arc in N_2. If S is a sink and its descendants in N_2, we know $N_2 \setminus S$ is also a valid network. If S is a single non-leaf vertex in N_2, then $x \in S$ is a leaf in N_1. Now, in N_2, there exists a triplet such that x has a descendant, while in N_1 this is not possible. So the triplet sets were different to start with. If S contains all nodes below a cut-arc and the starting node of this arc is not a cycle node, we know $N_2 \setminus S$ is also a valid network. If the starting node is a cycle vertex but is labelled or has out-degree three or more, then $N_2 \setminus S$ is valid as well.

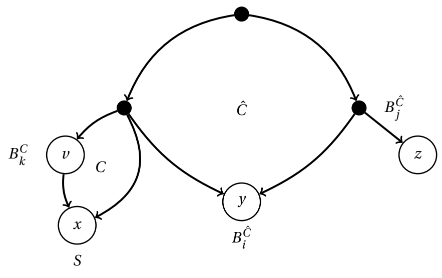
The only possible problem could occur if the starting node is an unlabelled cycle vertex with out-degree 2 of a cycle, C', of length 5. Namely, $N_2 \setminus S$ would contract the remaining out-going edge such that the cycle would become of length 4. If the cycle had no non-sink, non-source labelled cycle vertex, a cycle of length 4 would remain without such a labelled node. But in this case take $x \in S$, $y \in B_i^{C'}$, $z \in B_j^{C'}$ such that y is the sink of C' and z is in a different cycle branch from both $B_i^{C'}$ and S. Then in N_2 we have (a) $N_2|_{x,y,z} = \{x|yz, xy|z\}$, (b) $N_2|_{x,y,z} = \{xz|y, x|yz\}$, or (c) $N_2|_{x,y,z} = \{xz|y, z|xy\}$.



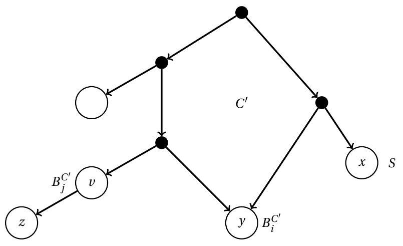
(a) N_1 as described in (a)(i)(1)



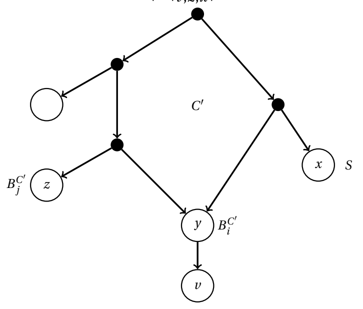
(b) N_1 as described in (a)(i)(2). Note that the location of the node v is not fixed but must be such that $|N|_{v,z,x}| = 2$.



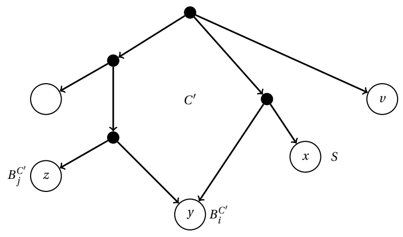
(c) N_1 as described in (a)(ii). Note that the location of the node v is not fixed but must be in a cycle branch of C.



(d) N_2 as described in (a)(i)(1)



(e) N_2 as described in (a)(i)(2)



(f) N_2 as described in (a)(ii)

Figure 4.4: The structure of the networks as described in Theorem 4.5 in case (a) when proving that the number of sinks can be reduced to one. Curved edges signify that any number of nodes and cycle branches can be along this edge.

(a) This case occurs when $B_j^{C'}$ and S are on different paths from the source of C' to y. For these triplets to occur in N_1 either (i) $y \in B_i^C$, $z \in B_j^C$ are such that their branches are on the same path from the source of C to x but are not themselves cycle nodes, or (ii) the source of C is a non-sink, non-source cycle vertex of $\hat{C}$, $y \in B_i^{\hat{C}}$, $z \in B_j^{\hat{C}}$, $B_i^{\hat{C}}$ is the branch containing the sink, and C and $B_j^{\hat{C}}$ are on different paths form the source to the sink of $\hat{C}$.

(i) If B_i^C, B_j^C, and S are the only branches of C, then at least B_i^C or B_j^C has a labelled cycle vertex, $c \in X$. This is visualised in Figures 4.4a and 4.4b where the node v can be disregarded. So there are either triplets $x/c \setminus y$ and $x|c \setminus y$, or $x/c \setminus z$ and $x|c \setminus z$. But in N_2 these triplets cannot exist, as can be seen when looking at Figures 4.4d and 4.4e.

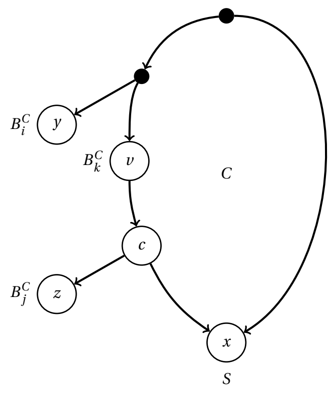
So, assume C has a length greater than four. Then there is another branch B_k^C containing a label v such that $|N_1|_{v,z,x}| \geq 1$.

(1) If $|N_1|_{v,z,x}| = 1$, then $N_1|_{v,z,x} = \{z/v|x\}$. So $v \in C$ and z is a descendant of v. Now if $N_2|_{v,z,x} = \{z/v|x\}$, then either v is a labelled cycle vertex of C' or $v \in B_j^{C'} \setminus C'$. Since, by assumption, C' has no labelled cycle vertices, the first possibility cannot hold. Therefore, it must be that $v \in B_j^{C'} \setminus C'$ and $N_2|_{v,z,y} = \{z/v|y\}$. But in N_1, B_i^C is on the same path from the source to the sink as B_j^C. So this triplet is impossible. See Figures 4.4a and 4.4d for clarification on the structure of N_1 and N_2, respectively.

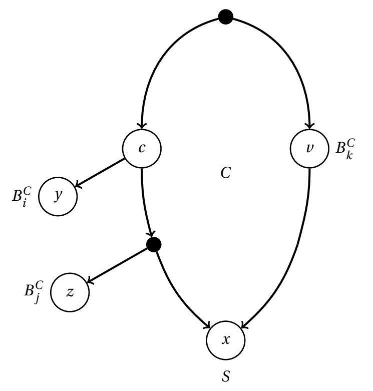
(2) If $|N_1|_{v,z,x}| = 2$ then for $|N_2|_{v,z,x}| = 2$ to hold $v \in B_i^{\hat{C}}$. Then v is a descendant of y in N_2, so $N_2|_{v,y,x} = \{v/y|x\}$. However, as $v \in B_k^C$ and $y \in B_i^C$, this triplet is not possible in N_1. See Figures 4.4b and 4.4e for clarification on the structure of N_1 and N_2, respectively. Note that the location of v in Figure 4.4b is not fixed, nor does v need to be a cycle vertex.

(ii) Then there exists a branch B_k^C containing a label v such that the triplet $x|v|y$ occurs in N_1. But this triplet is only possible in N_2 if B_k^C is a branch coming off from the source of C', as y is the sink of

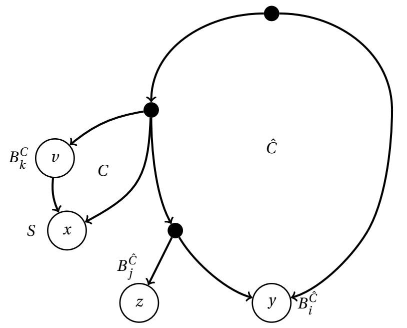
C' and x is in a branch of C'. But in N_1, another triplet (either $x/v|y$ or $xv|y$) also exists, which is not possible if B_k^C is a branch coming off from the source of C'. Such N_1 and N_2 are visualized in Figures 4.4c and 4.4f, respectively. Again, the location of v in Figure 4.4b is not fixed, nor does v need to be a cycle vertex of C.



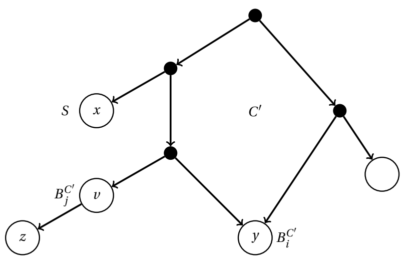
(a) N_1 as described in (b)(i)(1)



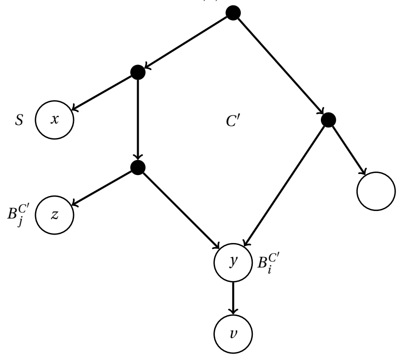
(b) N_1 as described in (b)(i)(2). Note that the location of the node v is not fixed but must be such that $|N|_{v,z,x}| = 2$.



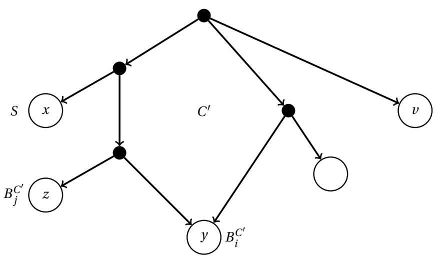
(c) N_1 as described in (b)(ii). Note that the location of the node v is not fixed but must be in a cycle branch of C.



(d) N_2 as described in (b)(i)(1)



(e) N_2 as described in (b)(i)(2)



(f) N_2 as described in (b)(ii)

Figure 4.5: The structure of the networks as described in Theorem 4.5 in case (b) when proving that the number of sinks can be reduced to one. Curved edges signify that any number of nodes and cycle branches can be along this edge.

(b) In this case $B_j^{C'}$ and S are on the same path from the source of C' to y and LCA(x,z) is an ancestor of LCA(y,z) (i.e. S is "above" $B_j^{C'}$). For these triplets to occur in N_1 either (i) $y \in B_i^C$, $z \in B_j^C$ such that they are on the same path from the source of C to x, or (ii) the source of C is a non-sink, non-source cycle vertex of $\hat{C}$, $y \in B_i^{\hat{C}}$, $z \in B_j^{\hat{C}}$, $B_i^{\hat{C}}$ is the branch containing the sink, C and $B_j^{\hat{C}}$ are on the same path form the source to the sink of $\hat{C}$, and LCA(x,z) is an ancestor of LCA(y,z) (i.e. C is "above" $B_j^{\hat{C}}$).

(i) Now, in N_1, LCA(y,z) is an ancestor of LCA(x,z) in N_1 (i.e. B_i^C is "above" B_j^C). By the same reasoning as in (a)(i), C must have a length greater than four. The corresponding networks for the argument there are presented in Figures 4.5a and 4.5b.

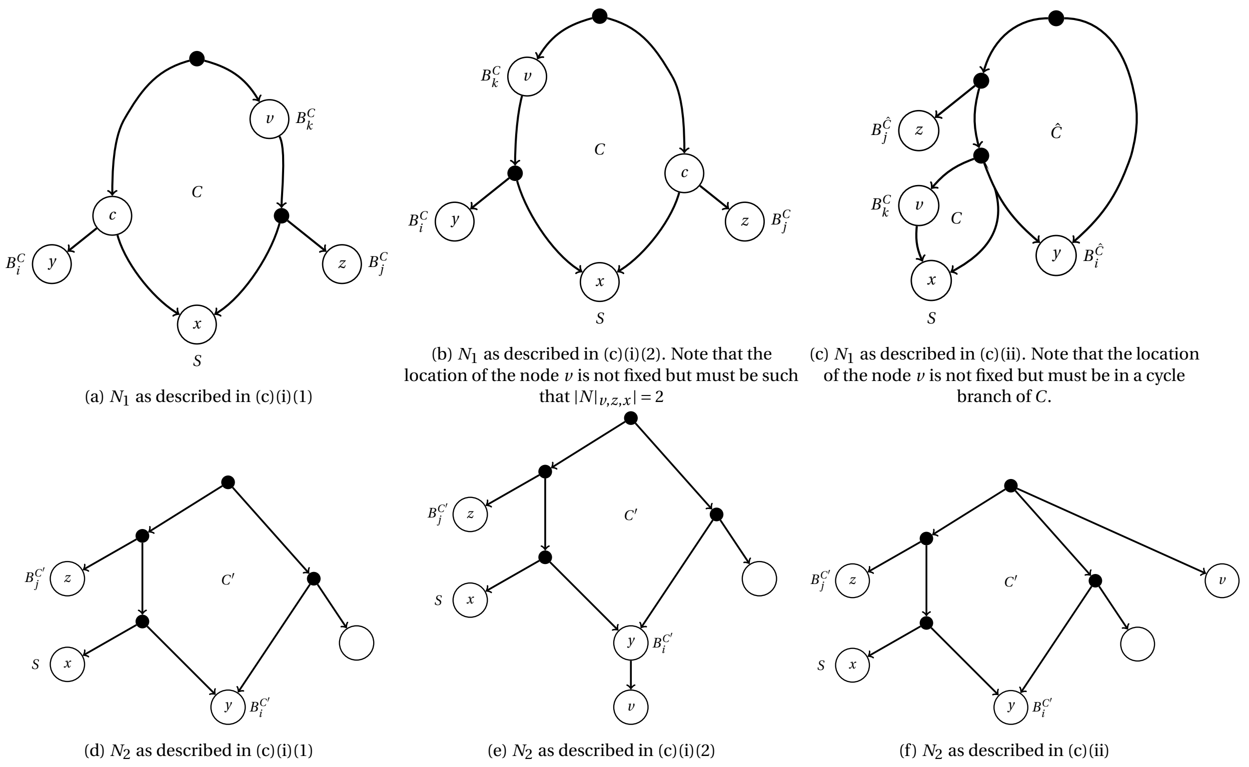
So, assume C has a length greater than four. Then there is another branch B_k^C containing a label v such that $|N_1|_{v,z,x}| \geq 1$.

(1) If $|N_1|_{v,z,x}| = 1$, then $N_1|_{v,z,x} = \{z/v|x\}$. So $v \in C$ and z is a descendant of v. Now if $N_2|_{v,z,x} = \{z/v|x\}$, then either v is a labelled cycle vertex of C' or $v \in B_j^{C'} \setminus C'$. Since, by assumption, C' has no labelled cycle vertices, the first possibility cannot hold. Therefore, it must be that $v \in B_j^{C'} \setminus C'$ and $N_2|_{v,z,y} = \{z/v|y\}$.

But in N_1 for $N_1|_{v,z,y} = \{z/v|y\}$ to hold, we also have $N_1|_{v,z,y} = \{y|v \setminus x\}$. However, in N_2 this triplet is not possible as $v \in B_j^{C'}$. See Figures 4.5a and 4.5d for clarification on the structure of N_1 and N_2, respectively.

(2) This is not possible by the same reasoning as (a)(i)(2). See Figures 4.5b and 4.5e for clarification on the structure of N_1 and N_2, respectively.

(ii) By the same reasoning as (a)(ii), this is not possible. Such N_1 and N_2 are visualized in Figures 4.5c and 4.5f, respectively.



(a) N_1 as described in (c)(i)(1)

(b) N_1 as described in (c)(i)(2). Note that the location of the node v is not fixed but must be such that $|N|_{v,z,x}| = 2$

(c) N_1 as described in (c)(ii). Note that the location of the node v is not fixed but must be in a cycle branch of C.

(d) N_2 as described in (c)(i)(1)

(e) N_2 as described in (c)(i)(2)

(f) N_2 as described in (c)(ii)

Figure 4.6: The structure of the networks as described in Theorem 4.5 in case (c) when proving that the number of sinks can be reduced to one. Curved edges signify that any number of nodes and cycle branches can be along this edge.

(c) In this case $B_j^{C'}$ and S are on the same path from the source of C' to y and LCA(x,z) is an ancestor of LCA(y,x) (i.e. $B_j^{C'}$ is "above" S). For these triplets to occur in N_1 either (i) $y \in B_i^C$, $z \in B_j^C$ such that they are on different paths from the source of C to x, or (ii) the source of C is a non-sink cycle vertex of $\hat{C}$, $y \in B_i^{\hat{C}}$, $z \in B_j^{\hat{C}}$, $B_i^{\hat{C}}$ is the branch containing the sink, C and $B_j^{\hat{C}}$ are on the same path form the source to the sink of $\hat{C}$, and LCA(x,z) is an ancestor of LCA(y,x) (i.e. $B_j^{\hat{C}}$ is "above" C).

(i) By the same reasoning as in (a)(i), C must have a length greater than four. The corresponding networks for the argument there are presented in Figures 4.6a and 4.6b.

So, assume C has a length greater than four. Then there is another branch B_k^C containing a label v such that $|N_1|_{v,z,x}| \geq 1$.

(1) If $|N_1|_{v,z,x}| = 1$, then $N_1|_{v,z,x} = \{z/v|x\}$. So $v \in C$ and z is a descendant of v. Then $N_1|_{v,x,y} = \{y|v \setminus x, yx|v\}$. But these triplets are not both possible in N_2. See Figures 4.6a and 4.6d for clarification on the structure of N_1 and N_2, respectively.

(2) This is not possible by the same reasoning as (a)(i)(2). See Figures 4.6b and 4.6e for clarification on the structure of N_1 and N_2, respectively.

(ii) By the same reasoning as (a)(ii), this is not possible. Such N_1 and N_2 are visualized in Figures 4.6c and 4.6f, respectively.

Thus, $N_2 \setminus S$ must also be a valid network. So let $N_1' = N_1 \setminus S$ and $N_2' = N_2 \setminus S$. Suppose N_2' would contain no cycles. Then for some $a,b,c \in X$, $|N_1'|_{a,b,c}| = 2$, while for N_2', $|N_2'|_{a,b,c}| \leq 1$. Which would imply $t(N_1') \neq t(N_2')$. Thus, N_2' must still contain a cycle.

It is left to prove that $N_1' \ncong N_2'$. We know, by the previous reasoning, that S is either a sink and its descendants of a cycle or a union of all labels below a cut-arc in N_2. Let S_i denote the sink and its descendants for all cycles, C_i, in N_1. Then $N_1 \setminus S_i \cong N_2 \setminus S_i$ for all i can only hold if all cycles in N_1 are connected from sink to source (i.e. one cycle's sink is the source of another) and for the lowest sink and its descendants, S_k, $N_1 \setminus S_k \cong N_2 \setminus S_k$ holds, as N_1 and N_2 are level-1 networks and N_1 has at least two cycles. But in that case, one can simply remove all outgoing cycle branches (different from the sink and its descendants) of a higher cycle. This would result in N_1' and N_2' such that the number of cycles in N_1' is smaller than in N_1 and $N_1' \ncong N_2'$. Also, N_1' and N_2' would still be valid networks as these cycle branches are located similarly in both N_1 and N_2 since $N_1 \setminus S_k \cong N_2 \setminus S_k$. Therefore, we found a smaller counterexample with fewer cycles. Following this reasoning, the smallest counterexample must have exactly one cycle in both N_1 and N_2. ■

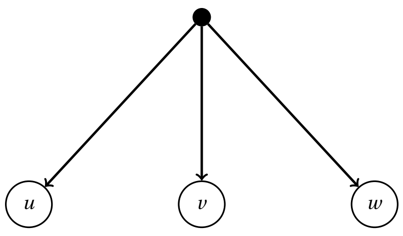
Now that we know N_1 and N_2 contain exactly one cycle, we wish to show that their respective roots must the the sources of their cycles and have no other outgoing branches.

Claim 4.5.3. N_1's and N_2's roots must be the source of their corresponding cycles and have out-degree two.

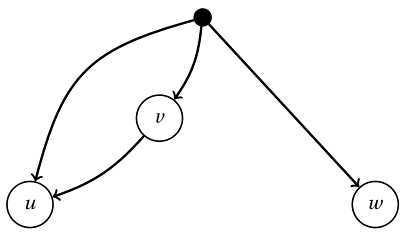
Proof.
Suppose N_1's root is not the source of its cycle. Then the root has k outgoing branches, each of which is a maximal SN set by Corollary 4.4. So N_1 has at most $k+1$ maximal SN sets; one for each branch and the root if it is labelled. These branches must, therefore, also be maximal SN sets in N_2.

Suppose N_2's root is the source of a cycle, then each of its cycle branches and labelled cycle vertices must correspond to one of the $k+1$ maximal SN sets.

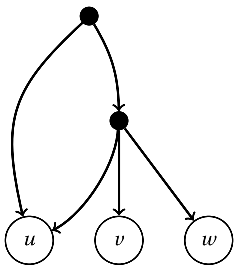
If $k >= 3$, then in N_1 there exists a triplet of the form $u|v|w$ whenever u, v, and w are from different branches. Since at least two maximal SN sets are part of the cycle in N_2, if there is a triplet $u|v|w$, there must also be a triplet, without loss of generality, of the form $u/v|w$ or $u|vw$. Meanwhile, these triplets cannot exist in N_1. See Figure 4.7a for the general structure of such N_1 and Figures 4.7b and 4.7c for the general structures of N_2.



(a) N_1 as described when $k \geq 3$. Note that the root could have more branches and could be labelled.



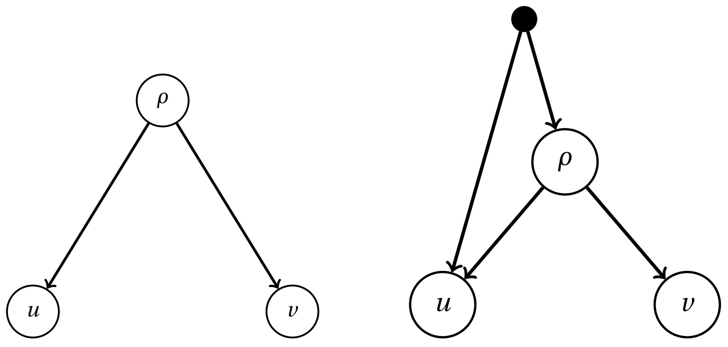
(b) N_2 as described when $k \geq 3$ and a triplet of the form $u/v|w$ exists. Note that the root could have more branches and could be labelled.



(c) N_2 as described when $k \geq 3$ and a triplet of the form $u|vw$ exists. Note that the root could have more branches and could be labelled.

Figure 4.7: The general networks as described in the case when $k \geq 3$. The curved edges signify that any number of cycle branches or nodes could be located there.

If $k = 2$, then N_1 has at most 3 maximal SN sets. N_2 has at least 3 maximal SN sets: the sink and its descendants, another cycle branch, and the labelled non-sink non-source cycle vertex. So, the root, ρ, of N_1 must be labelled, and therefore there is a triplet of the form $u/\rho \setminus v$. For this to hold in N_2, ρ must be the labelled non-sink non-source cycle vertex, and, without loss of generality, u must be part of the sink and v in the remaining cycle branch. However, the triplet $u|\rho \setminus v$ also exists in N_2 but not in N_1. Such networks are shown in Figure 4.8.



(a) N_1 as described when $k = 2$. Note that u and v could have more ancestors or descendants.

(b) N_2 as described when $k = 2$. Note that u and v could have more descendants.

Figure 4.8: The two general networks as described in the case when $k = 2$.

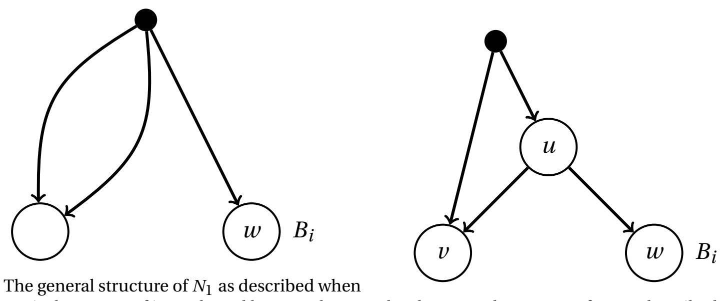
Lastly, if $k = 1$, then N_1 has only two maximal SN sets while N_2 has at least three. So this is not possible. So we conclude that if N_1's root is not the source of its cycle, neither is N_2's root the source of its cycle. Likewise, if N_1's root is the source of its cycle, so is N_2's root the source of its cycle by the same reasoning.

Now, if both roots are not the sources of their cycles, then they must have the same out-degree as the maximal SN sets, and triplets are the same. So let B be the union of all labels in the branches from N_1 that do not contain the cycle. Then $N_1 \setminus B$ still contains a cycle and must be a valid network, as is $N_2 \setminus B$ since the corresponding SN sets must also be branches from N_2. $N_2 \setminus B$ must also still contain a cycle as otherwise there exist $a, b, c \in X$ such that $|N_1|_{a,b,c}| = 2$ while $|N_2|_{a,b,c}| \leq 1$. If both roots have out-degree one, then their roots, ρ, must be the same and therefore $N_1 \setminus \rho$ and $N_2 \setminus \rho$ are both valid networks. The obtained networks must still be non-isomorphic, as otherwise one of these SN sets is a smaller counterexample without cycles, which contradicts Theorem 3.5.

If both roots are the sources of their cycles, but without loss of generality, the root of N_1 has out-degree three or higher, then the branches in the blob graph of N_1 not containing the cycle are maximal SN sets by Corollary 4.4. Let $B_i \subset X$ be such a branch and take $w \in B_i$. If B_i is not a branch in the blob graph of N_2 not containing the cycle, it must be a cycle vertex, the labelled root, or a cycle branch in N_2 by Corollary 4.4. Note that B_i cannot be the sink of the cycle in N_2 as otherwise $t(N_1 \setminus B_i) \neq t(N_2 \setminus B_i)$ which would contradict Lemma 4.1.

In the case that B_i is a cycle vertex or the labelled root in N_2, then there is a triplet such that w has a descendant that is not in B_i, which cannot hold in N_1. If B_i is a cycle branch, then either the cycle size in N_2 is 3 or larger.

If the cycle size is 3, then the labelled non-sink, non-source cycle vertex, u, of the cycle forms $w/u|v$ and $w/u \setminus v$ as triplets in N_2 where v is the sink of the cycle. The triplet $w/u|v$ is not possible in N_1 as $u, v \notin B_i$. Such networks are shown in Figure 4.9

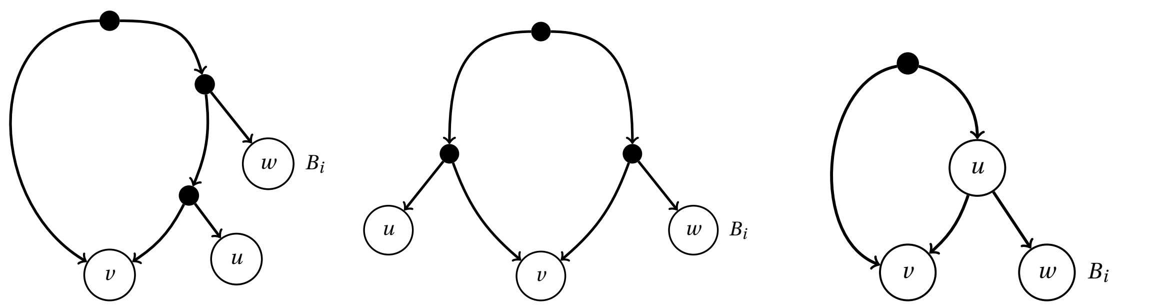


(a) The general structure of N_1 as described when its root is the source of its cycle and has out-degree three or higher.

(b) The general structure of N_2 as described when its cycle has length 3.

Figure 4.9: The two general networks as described in the case when the roots are the sources of the cycles, the root of N_1 has out-degree three or higher, and the cycle in N_2 has size three. The curved edges signify that any number of cycle branches or nodes could be located there.

If the cycle size is 4 or larger, take u a label from another cycle branch and v the sink of the cycle. Then there is a triplet in N_2 of the form $wu|v$, $wv|u$, or $w/u|v$. Such networks are shown in Figures 4.10a, 4.10b and 4.10c, respectively. All of these triplets are not possible in N_1 as $u, v \not\in B_i$. N_1's general structure is again shown by Figure 4.9a.



(a) N_2 such that its cycle has length four or larger and it contains a triplet $v|uw$.
(b) N_2 such that its cycle has length four or larger and it contains a triplet $wv|u$.
(c) N_2 such that its cycle has length four or larger and it contains a triplet $v|u \setminus w$.

Figure 4.10: The three general network structures of N_2 as described in the case when the roots are the sources of the cycles, the root of N_1 has out-degree three or higher, and the cycle in N_2 has size four or larger. The curved edges signify that any number of cycle branches or nodes could be located there.

So B_i must also be a branch in the blob graph of N_2, not containing the cycle. Therefore, $N_1 \setminus B_i$ and $N_2 \setminus B_i$ are both valid networks, and $N_1 \setminus B_i \ncong N_2 \setminus B_i$ must hold as otherwise B_i would be a smaller counterexample which contradicts Theorem 3.5.

Applying this reasoning until neither case is true for N_1 or N_2 gives us that their roots must be the sources of their cycles and have out-degree two. ■

So, by Claims 4.5.2 and 4.5.3, N_1 and N_2 both contain exactly one cycle whose sources are their respective roots and have out-degree two. To prove that these networks are actually isomorphic, we will first show that all branches of the cycle must contain the same labels, then that both cycles have the same branch as their sink, and lastly that the ordering of the cycles' branches must be the same.

Claim 4.5.4. Let C^1, C^2 be the cycles in N_1 and N_2, respectively. Then, their cycles must have the same length, l, and their exists a bijection η on these branches, $B_i^{C^1}$ and $B_i^{C^2}$ for $i \in \{1, 2, \ldots, l\}$, such that $B_i^{C^1}$ and $\eta(B_i^{C^1})$ contain the same labels and $B_i^{C^1} \cong \eta(B_i^{C^1})$.

Proof.
We again know the maximal SN sets are the same for N_1 and N_2. To show the cycle branches are the same, it is left to show that the cycle vertices of N_1 are also cycle vertices of N_2 and that they have the same maximal SN sets as their corresponding cycle branches.

Let $V = \{v\}$ be a maximal SN set such that v is a cycle vertex in N_1. Then there exists a triplet with v and the sink s such that s is a descendant of v. Therefore, v must also be a cycle vertex in N_2 as otherwise it would not have a triplet such that it has a label from another maximal SN set as a descendant. Likewise, if V is a maximal SN set such that it is not a cycle vertex in N_1, it can also not be a cycle vertex in N_2.

To show that the cycle vertices have the same maximal SN sets as part of their corresponding cycle branches, let v be such a cycle vertex such that it has out-degree k in N_1, with $k \geq 2$. Let $B_{v,i}^{C^1}$ for $i = 1 \ldots k-1$ be these branches from the subtree rooted at v not containing the sink in N_1 and take $u_i \in B_{v,i}^{C^1}$ and s the sink in N_1. Then there are triplets of the form $u_i / v \setminus s$ and $u_i / v | s$ in N_1. These triplets can only exist in N_2 if $B_{v,i}^{C^1}$ is also a branch from the subtree rooted at v in N_2.

Moreover, if $v \in V$ is an unlabelled cycle vertex with out-degree three or higher, then each of the branches from the subtree rooted at v not containing the sink in N_1 must also be in the same cycle branch in N_2. Namely, these branches are also all maximal SN sets in N_2. So take u and w from two distinct such branches and s the sink in N_1. Then they form $u|w|s$ and $u, w|s$ as triplets, which is only possible in N_2 if both branches are also rooted at the same cycle vertex in N_2.

Thus, by this reasoning, for $t(N_1) = t(N_2)$ to hold, all the branches of the cycle must be the same, and both networks have the same number of branches. Therefore, from now on, we will refer to the cycle C instead of C^1 and C^2. Also, the branches in N_1 and N_2 must be isomorphic to one another, as otherwise such a branch would be a smaller counterexample. ■

Claim 4.5.5. N_1 and N_2 have the same branch as their respective sinks.

Proof.
By Claim 4.5.4 we know the cycle, C, of N_1 and N_2 has the same cycle branches. Suppose now that $S_1 \subset X$ is the branch that contains the sink of C in N_1, while $S_2 \subset X$ is different from S_1 and contains the sink of C in N_2. Let $l \geq 3$ be the size of the cycles.

If $l \geq 5$, then there exist different i and j such that $S_1 \neq B_i^C \neq S_2$ and $S_1 \neq B_j^C \neq S_2$ and $a \in B_i^C$, $b \in B_j^C$, and $c \in S_1$ such that $|N_1|_{a,b,c}| = 2$ as a, b, and c are in different branches of the cycle, while c is in the branch of the sink. However, in N_2, $|N_2|_{a,b,c}| \leq 1$ since none of the labels are in the branch of the sink.

Now, if $l = 4$, denote the remaining branch as B. Either (i) N_1 has both branches on the same path from the root to the sink or (ii) N_1 has both branches on a different path from the root to the sink. Take $s \in S_1$ and $v \in X$, the labelled non-sink, non-source cycle vertex of C.

(i) If v has out-degree two, take $u \in X$ its non-cycle child (or one of its labelled descendants). Then the triplets on s, u and v in N_1 are of the form $u/v \setminus s$ and $u/v|s$. By Claim 4.5.4 we know u and v are also in the same cycle branch in N_2 and s in another. So these triplets are only possible in N_2 if S_1 also contains the sink in N_2.

    If v has out-degree one and its child is not the sink of the cycle, then take $u \in X$ to be that child. Then N_1 has $u/v|s$ and $s/u/v$ as triplets. Again, these triplets are only possible in N_2 if S_1 is also the sink in N_2.

    Lastly, if v has out-degree one and its child is the sink of the cycle, then its parent must be a non-source cycle vertex. If it is labelled, let $u \in X$ be that label. Otherwise, it must have another child. So let $w \in X$ be that other child (or one of its labelled descendants). Then the following triplets occur in N_1: $s/v/u$ and $v/u|s$, or $w|v \setminus s$ and $w, v|s$. These triplets are again only possible if S_1 is also the sink in N_2.

(ii) If the other non-sink, non-source cycle vertex is labelled, let $u \in X$ be that label. In N_1, we would have $s/u|v$ and $s/v|u$ as triplets. In N_2, these triplets can only exist if S_1 is its sink.

    If, however, the other non-sink, non-source cycle vertex is not labelled, let $u \in X$ be its non-cycle child (or one of its labelled descendants). The triplets on u, v and s would then be $s/v|u$ and $v|u, s$. Again, these triplets are only possible if S_1 is also the sink in N_2.

Lastly, if $l = 3$, take $a \in S_1$ and $b \in S_2$ the roots of these branches. Then also $a, b \in C$. Now, since $|X| \geq 3$ take $c \in X \setminus \{a, b\}$. If c is the root of N_1, then $a/c \setminus b$ and $a/b/c$ are triplets in N_1. These triplets can only occur in N_2 if S_1 also contains its sink. If $c \in S_1$, the triplets would be $c/a/b$ and $c/a|b$. Now if $c \in S_2$ we have the triplets $a/b \setminus c$ and $a|b \setminus c$. Again, both triplet combinations are only in N_2 if S_1 also contains the sink in N_2.

Therefore, for $l \geq 3$, both N_1 and N_2 must have the same branch as their sink. ■

To prove the ordering of the branches is the same, without loss of generality, either $l \geq 5$ or $l = 4$, and the source of the cycle points directly to the sink in N_1. Namely, if $l = 4$ and N_1 and N_2 both have only one branch on both paths from the source to the sink, there is no ordering. And if $l = 3$, clearly there is no ordering either.

In both cases, there are at least two non-sink branches on one of the paths from the source to the sink in N_1. Denote these branches as B_i^C and B_j^C such that the least common ancestor of any two nodes from these branches is in B_i^C, and let S be the branch containing the sink. Take $a \in B_i^C$, $b \in B_j^C$, and $c \in S$.

Suppose in N_2, B_i^C is on a different path from the root to the sink than B_j^C. Then the triplets induced by a, b, and c in N_2 are of the form $ac|b$ and $a|bc$, $c/a|b$ and $a|bc$, $ac|b$ and $a|b \setminus c$, or $c/a|b$ and $a|b \setminus c$. While in N_1, they would be of the form $cb|a$ and $c|ab$, $c|a \setminus b$ and $a \setminus b \setminus c$, $c/b|a$ and $c|ab$,or only $c|a \setminus b$. Therefore, the triplet sets are different.

Now suppose that in N_2, the least common ancestor of any two nodes from B_i^C and B_j^C is in B_j^C. Then the triplets induced by a, b, and c in N_2, are of the form $ca|b$ and $c|ab$, $c|b \setminus a$ and $b \setminus a \setminus c$, $c/a|b$ and $c|ab$,or only $c|b \setminus a$. While the triplets for N_1 are still the same as described above. Thus, again, the triplet sets are different.

We see that in both cases, the triplet sets would differ. We can thus conclude that the ordering of the branches of the cycle must therefore, also be the same.

To conclude, we have shown that the branches of the cycle must be the same, that the branches containing the sink of the cycle must be the same and that the ordering of the branches is the same. Combining this with the fact that the structure of the branches themselves must be the same, do not contain more cycles, and Theorem 3.15, we can conclude that in fact $N_1 \cong N_2$. Therefore, no smallest counterexample exists. □

4.2. Algorithm

Similar to section Section 3.2.2, the problem of reconstructing a level-1 stemmatic network has not been researched previously. Studies in the reconstruction of level-1 phylogenetic networks, however, have been performed (Jansson & Sung, 2006; Huber et al., 2011; Gambette et al., 2017; van Iersel & Kelk, 2008). Combined, they have presented several polynomial-time algorithms that are able to find a network that is consistent with a subset of triplets, and at times will always return a level-1 phylogenetic network. However, these algorithms are not capable of handling labelled internal vertices and often restrict the out-degree of vertices to two. We will present an algorithm that can reconstruct a network that satisfies Definition 4.1 based on the full triplet set of a given network N. Moreover, if N is a tree, the algorithm will return a tree. Unlike Algorithms 1 and 2, this algorithm is not capable of handling any partial triplet set.

Our algorithm works by checking if N has a labelled root using Algorithm 8 based on Lemma 3.16, which still holds for networks by the same reasoning. Next, it divides the labels in their respective branches using Algorithm 10. This algorithm will return all labels in a cycle in the same branch. Then, it resolves each branch separately. If the branch corresponds to a cycle, it finds the sink of that cycle and partitions all cycle branches based on which path from the source to the sink they are on. Lastly, both of these sets can then be solved independently. This results in a network that is consistent with the triplet set.

Algorithm 11 computes all SN sets for a given network N based on the triplet set and labels. It is based on the original algorithm from Jansson and Sung (2006). The algorithm has been altered to properly handle the additional types of triplets used here. However, the same reasoning for correctness holds as they used (Lemma 7, Jansson and Sung, 2006). Jansson and Sung proved that computing $SN(\{a, b\})$ for any $a, b \in X$ can be done in $\mathcal{O}(|X|^3)$ time. Thus, computing all SN sets can be done in $\mathcal{O}(|X|^5)$ time. The algorithm can be found in Appendix A.2.

Algorithm 12 processes a fanned triplet by ensuring its labels are in the proper branches. It runs in $\mathcal{O}(|X||t(T)|)$ time. Lemma A.3 proves the correctness of the algorithm and its running time. Both Algorithm 12 and Lemma A.3 can be found in Appendix A.2.

When a source of a cycle has been reached, it is necessary to find the sink of that cycle. Algorithm 13 identifies the maximal SN set corresponding to highest sinks with no cut-arcs above it in $\mathcal{O}(|X|^3 + |t(N)|^2)$ time. The algorithm and its corresponding proof, Lemma A.4, are presented in Appendix A.2.

If Algorithm 13 returns several maximal SN sets, then a cycle vertex must be the source of another cycle. Algorithm 14 finds the maximal SN set that corresponds to the sink of the highest cycle. It does this in $\mathcal{O}(|X|^3)$ runtime as is proven in Lemma A.5. Again, this algorithm and its proof can be found in Appendix A.2.

Once the sink of a cycle has been identified, the cycle branches and their order must be determined. Algorithms 15 and 16 do these tasks, respectively. Algorithm 15 uses the triplets on any two labels together with one label in the sink to identify different branches. Its theoretical running time is $\mathcal{O}(|X|^3)$. Its correctness and running time are proven in Lemma A.6. To compute the ordering of the branches, Algorithm 16 again looks at the combination of triplets on a sink label and two labels from different cycle branches to identify if the branches are on the same path from the source to the sink or not. The theoretical running time is $\mathcal{O}(|X|^4)$, which, together with the correctness of the algorithm, is proven in Lemma A.7. Algorithms 15 and 16 and Lemmas A.6 and A.7 are presented in Appendix A.2.

Once both "sides" of the cycle are identified, recursion can be used to solve both separately. However, the correct triplets need to be passed on, as some triplets use paths from the other "side" and thus should not be included. Algorithm 18 checks a triplet for whether or not it should be included for the recursion. It has a theoretical running time of $\mathcal{O}(|X|)$. Its correctness and running time are proven in Lemma A.8. The algorithm and proof can be found in Appendix A.2.

Algorithm 3 is the overarching algorithm of this process, and its correctness is proven in Theorem 4.6.

The network in Figure 4.11 will be used as an example to show how the algorithm works.

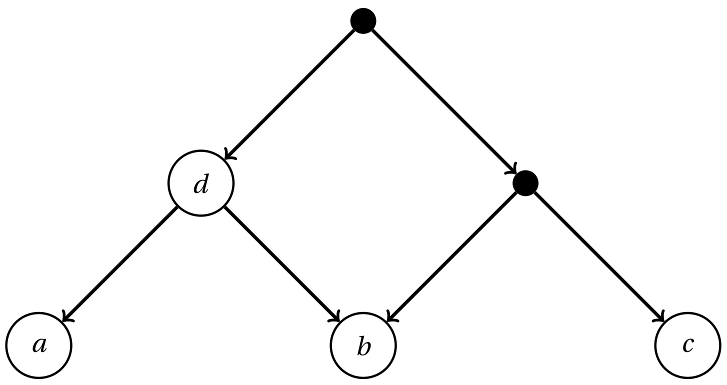


Figure 4.11: The network N used for an example of the steps in the algorithm.

After computing S, D, and the hashmaps, the SN sets will be constructed. Each node will be its own maximal SN set. Algorithm 8 will not return any possible roots, and Algorithm 10 will return only one branch containing all labels. Therefore, in Line 24 we know we are dealing with a cycle and thus find its sink. Algorithm 13 will correctly identify b as its sink, and since only one SN set is returned by this algorithm Algorithm 14 will not do anything. $Algorithm$ 15 will find $\{a, d\}$ and $\{c\}$ as the cycle branches and d as the internal cycle vertices. Then, Algorithm 16 will place the two cycle branches on different paths from the source to the sink. The loop in Line 30 will correctly filter the triplets such that for the left side we get the triplet $a/d \setminus b$ and no triplets for the right side.

The two sides are then resolved by recursion. For the left side, d is chosen as its root, and a and b are placed in separate branches. For the right side, only two labels are left, so using the previously computed S and D sets, we know neither is a descendant of the other. Therefore, by Line 14 the root is left unlabelled and b and c are added as the children. Combining both sides gives us the network as in Figure 4.11.

Algorithm 3: Reconstruct

```
1  Function Reconstruct($t(N), X$):
       Input: $t(N)$ — a set of triplets
       Input: $X$ — the labelled nodes of N
       Output: $N$ — a network such that it contains all triplets in $t(N)$
2      Compute $S$ and $D$ using Algorithm 7 if $|X| \geq 3$ and use $S$ and $D$ from a previous iteration otherwise
3      Create hashmaps mapping every three labels and every two labels to all triplets containing those
        labels
4      Compute all $SN$ sets using Algorithm 11 and keep all maximal $SN$ sets
5      if $|X| = 1$ then
6          return the single node in $X$
7      else if $|X| = 2$ then
8          if either is a descendant of another according to $D$ then
9              if this descendant is also separated from the other according to $S$ then
10                 return an unlabelled root with both labels as its children and the descendant also the
                    child of the other label
11             else
12                 return the ancestor as the root and the descendant as its child
13         else
14             return an unlabelled root with both labels as its children
15     $R$ = PossibleRoots($t(N), X, D, S$) using Algorithm 8
16     if $|R| = 1$ then pick $\rho \in R$ to be the root of $N$
17     else let the root $\rho$ of $N$ be unlabelled
18     $B$ =DivideBranches($t(N), X, D, \rho$) using Algorithm 10
19     if $|B| = 1$ then
20         if $\rho \in X$ then
21             Let $t'(N) \subseteq t(N)$ be all triplets in $t(N)$ that do not contain $\rho$
22             if |PossibleRoots($t'(N), B, D, S$)| = 1 then
23                 return $\rho$ as the root of $N$ with Reconstruct($t'(N), X \setminus \{\rho\}$) as its child
24         $SDs$ =FindSinkOfCycle($\rho, t(N), X, MaxSN, D$) using Algorithm 13
            $SD$ =RemoveOuterSinks($SDs, \rho, X, D, t(N)$) using Algorithm 14
25         $CB, CV$ = ResolveCycle($SD, \rho, D, t(N)$) using Algorithm 15
26         $L, R$ =FindCycleOrder($CB, SD, CV, \rho, t(N)$) using Algorithm 16
27         $LL$ = all labels in $L$ and $SD$
28         $RL$ = all labels in $R$ and $SD$
29         Initialize $LT$ and $RT$ as two empty lists
30         foreach $t \in t(N)$ do
               // Using Algorithm 18 to filter triplets
31             if all labels of $t$ are in $LL$ and FilterTriplets($t, L, SD, CV, t(N)$) then
32                 Append $t$ to $LT$
33             if all labels of $t$ are in $RL$ and FilterTriplets($t, R, SD, CV, t(N)$) then
34                 Append $t$ to $RT$
35         Let $\rho$ be the root of $N$ with Reconstruct($LT, LL$) and Reconstruct($RT, RL$) as children.
36     else
37         Let $\rho$ be the root of $N$
38         foreach $b \in B$ do
39             Let $t'(N) \subseteq t(N)$ be all triplets in $t(N)$ that only contain labels in $B$
40             Add Reconstruct($t'(N), b$) as a child of $\rho$
41             if Reconstruct($t'(N), b$)'s root is unlabelled and the source of a cycle then
42                 if $\exists u|v|w \in t(N)$ such that $u, v \in b$ and $w \notin b$ then
43                     Replace Reconstruct($t'(N), b$)'s root by $\rho$
44     return $N$
```

Theorem 4.6. Let N be a network according to Definition 4.1 on X and $|X| \geq 3$. Then Algorithm 3 with input $(t(N), X)$ returns N in $\mathcal{O}(|X|^6 + |t(N)|^2|X|^2)$ time.

Proof.
Let N be a network on X, with $|X| \geq 3$. Then, if the root of N is labelled, Line 15 will compute which label it is. Otherwise, the root must be unlabelled. Let ρ be the root of N.

Algorithm 10 is used to compute the different branches coming from the root. A cycle is regarded as a single branch. Two cases can arise: (i) there is only one branch returned by Algorithm 10, or (ii) more than one branch is returned.

(i) Since only one branch is returned, either the root has out-degree one, or the root is the source of a cycle. In the first case, this would require ρ to be labelled and point to a labelled node by Definition 4.1. So if Algorithm 8 does return a single labelled root, ρ must have had out-degree one and thus the branch can be reconstructed correctly by induction. This reconstructed branch is then added as a child in Line 23.

    In the second case, either ρ is not labelled or Algorithm 8 would not return a single labelled root for the branch in Line 22. Therefore, we know the root must be the source of a cycle. Algorithms 13 and 14 are used in Line 24 to identify which labels are part of the sink of this cycle by Lemmas A.4 and A.5. Algorithms 15 and 16 are then used in Lines 25 and 26 to correctly identify the different cycle branches and which cycles are on which path from the source to the sink (or "sides") by Lemmas A.6 and A.7. To correctly identify the triplets corresponding to the two paths, Algorithm 18 is used. Indeed, by Lemma A.8, Algorithm 18 correctly returns whether or not a triplet containing only labels from one "side" of the cycle did not use edges from the other "side". Then, by induction, these "sides" are correctly reconstructed by Algorithm 3 and added as children of ρ in Line 35.

(ii) For each branch B_i, Algorithm 3 is called with $(t'(N), b)$ as input, where b are the labels of the branch and $t'(N)$ is the set of triplets only containing labels in b. By induction, this correctly reconstructs the branch, which is then added as a child of the root. Note that if B_i is a cycle with ρ as its source, the reconstructed branch will be a cycle with an unlabelled root different from ρ. Meanwhile, there would be a triplet of the form $u|v|w$ such that $u, v \in b$ and $w \notin b$. Therefore, if this is the case, the root of the branch is removed and replaced by ρ in Line 43

Note that if Algorithm 3 is called recursively on a branch with one or two labels Lines 5 to 14 can handle these situations based on the descendants and separation (D, S) computed in a previous iteration.

Looking at the time complexity for one iteration of Algorithm 3, computing S and D takes $\mathcal{O}(|t(N)|)$ time, computing the hashmaps takes $\mathcal{O}(|X|^3)$, and the maximal SN sets $\mathcal{O}(|X|^5)$. Then Algorithm 8 runs in $\mathcal{O}(|X| + |t(N)|)$ time, and Algorithm 10 in $\mathcal{O}(|t(N)|^2|X|)$ time. If ρ is labelled and the source of a sink, Algorithm 8 is run on the single branch returned by Algorithm 10, which runs in $\mathcal{O}(|X| + |t(N)|)$ time. Then, since the root is the source of a cycle, Algorithms 13 to 16 are run, which run in $\mathcal{O}(|X|^3 + |t(N)|^2)$, $\mathcal{O}(|X|^3)$, $\mathcal{O}(|X|^3)$, and $\mathcal{O}(|X|^4)$ time, respectively. Lastly, for every triplet Algorithm 18 is run, which therefore in total runs in $\mathcal{O}(|X||t(N)|)$ time.

Thus one whole iteration of Algorithm 3 runs in at most $\mathcal{O}(|X|^5 + |t(N)|^2|X|)$ Reconstructing any network $N = (V, E, l)$ calls Algorithm 3 at most $|V|$ times. Now $|V|$ is capped by $2|X|$, so reconstructing any network runs in at most $\mathcal{O}(|X|^6 + |t(N)|^2|X|^2)$ time. Since $|t(N)| \leq |X|^3$, the theoretical running time could also be expressed in only the size of X, which would give a running time of $\mathcal{O}(|X|^8)$. □

5

Computational Performance

The theoretical running times of Algorithms 1 to 3 have been shown in the previous sections. In this section, we will compare the theoretical running times to the practical running times and look at the overall performance of the algorithms. To do this, artificial datasets have been created randomly. Section 5.1 explains how this has been done. The computational tasks in this study were performed on an HP ZBook Power G7 Mobile Workstation, equipped with an Intel® Core™ i7-10750H processor (6 cores, 12 threads, base frequency 2.6 GHz), 16 GB of RAM, and an NVIDIA Quadro T1000 GPU with 4 GB VRAM. All the code used to obtain the results in this section and Section 6.1, as well as the implementation of the algorithms, can be found on https://github.com/TMALevert/triplet_distance.

5.1. Dataset Creation

The multifurcating and general trees have been constructed using the method `random_labeled_rooted_tree` from the `Python` package `networkx` (Hagberg et al., 2008). This function creates a random rooted tree where each node has an arbitrary degree. So to obtain multifurcating trees, we check for each node if it has at least out-degree two. If it does not, we randomly add up to a given number of children, but at least one. To obtain the labels for the tree, all leaf labels and a random number of internal labels are chosen. For the general trees, we also ensure that for each node with out-degree one, the label of that node and its child are chosen.

To obtain random level-1 networks, the function `generate_network_random_tree_child_sequence` from the `Python` package `phylox` is used (Janssen, 2024). This function was created for test cases for the algorithm described in Janssen and Murakami, 2020, and creates a random tree-child network with a given number of leaves and reticulations. So to obtain a level-1 network with n cycles, we create n such random networks with only one reticulation and randomly connect them. These subnetworks are connected by either adding a root as a child of a node of another subnetwork or by replacing the root of a subnetwork with the node of another subnetwork. This way, we ensure that the resulting network is level-1. To choose its labels, we choose all required labels; the leaves, sinks, out-degree one nodes and their children, and at least one non-sink, non-source cycle vertex for each cycle of length 3 or 4. Then, a randomly chosen subset of the remaining labels is added.

For each network type, 900 random networks have been created using the above-mentioned methodologies. Figure 5.1 shows the distribution of the number of labels for the obtained dataset. The number of labels ranges from 4 to nearly 100, although the general trees only have a maximum of roughly 60 labels. This is likely due to the fact that out-degree one nodes are allowed and therefore, no additional children (and thus leaves) are added. Although the distribution is not uniform for nearly every number of labels, the sample size is large enough to obtain reasonable results.

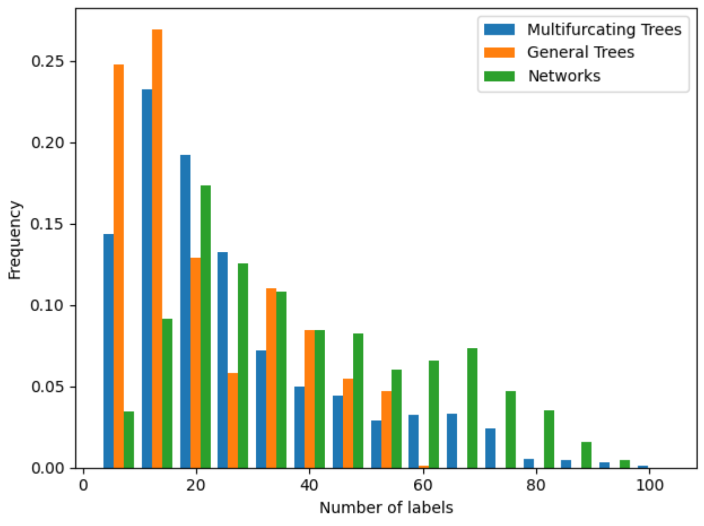
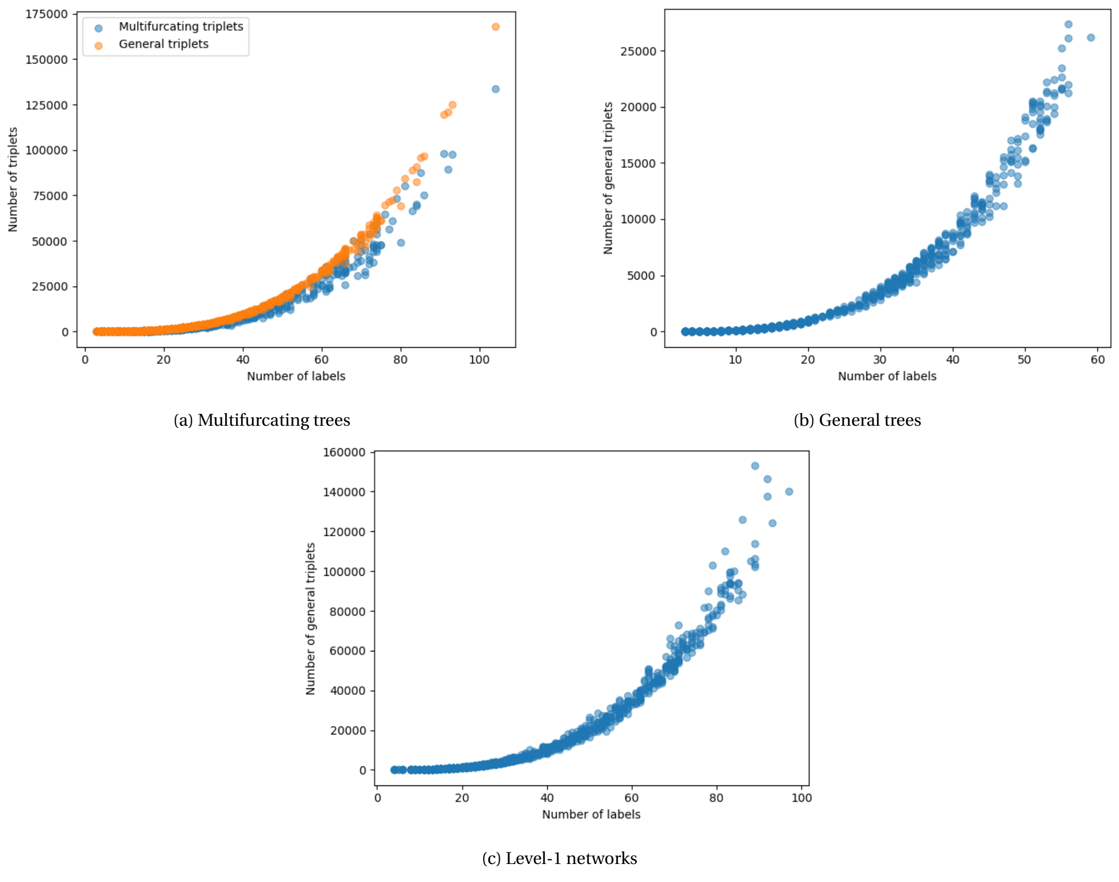


Figure 5.1: Distribution of the number of labels per network type.

The number of triplets per number of labels can be found in Figure 5.2 for different network types. Indeed, for all network types, the number of triplets scales with $\mathcal{O}(|X|^3)$ as can be expected. Moreover, the number of triplets for a network is generally more than for a general tree with the same number of labels. This is caused by the fact that three labels can induce more than one triplet in a network, but not in a tree.



(a) Multifurcating trees

(b) General trees

(c) Level-1 networks

Figure 5.2: The number of triplets against the number of labels for a given network type.

Contrary to what one might expect, the number of triplets does not significantly increase as the number of cycles increases. In fact, any three labels can at most induce four triplets. This happens when one of the nodes is a sink of a cycle C, and the other two are from another cycle whose source is a cycle vertex of C. Indeed, Figure 5.3 shows that although the number of triplets seems to increase as the number of cycles does, it is mostly due to the corresponding increase of labels.

To time the reconstruction algorithms, the built-in `timit` function has been used to run the algorithm 5 times per instance, after which the fastest runtime is saved. We choose the fastest runtime as this should most clearly represent the actual running time without any additional overhead time. The runtime does not include the time taken to create the networks or obtain the triplet sets.

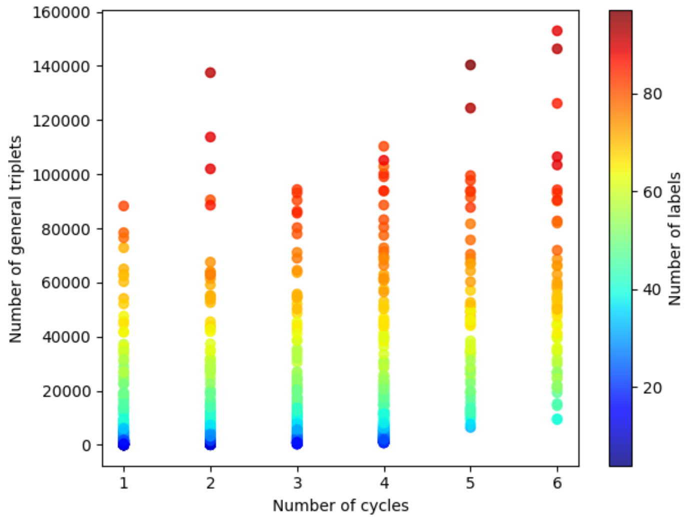
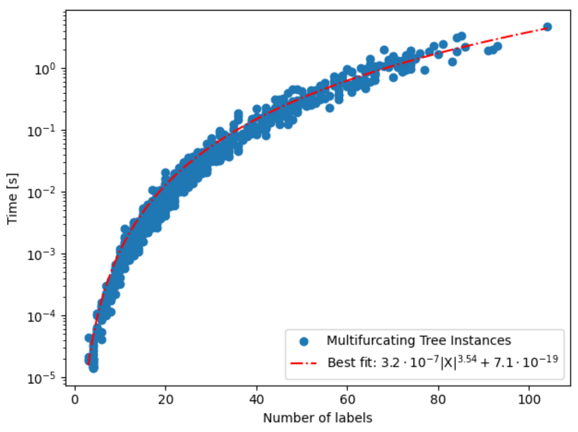


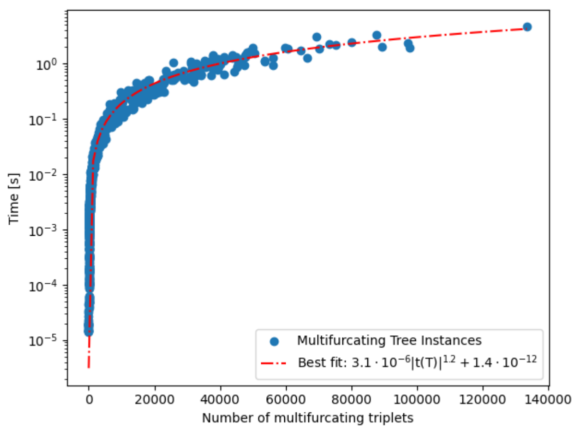
Figure 5.3: Number of triplets against the number of cycles in a level-1 network. The colour of the points resembles the number of labels in that network.

5.2. Multifurcating Tree Reconstruction

As was shown in Section 3.1.2 the reconstruction algorithm for multifurcating trees runs in $\mathcal{O}(|t(T)|^2|X| + |X|^2|t(T)|)$. Using that $|t(T)| \leq \binom{|X|}{3}$ and thus $\mathcal{O}(|t(T)|) = \mathcal{O}(|X|^3)$, we get a theoretical running time only in terms of $|X|$ of $\mathcal{O}(|X|^7)$. Likewise, if we wish to express the theoretical runtime only in terms of $|t(T)|$, we obtain $\mathcal{O}(|t(T)|^{\frac{7}{3}})$. Looking at Figure 5.4a, we indeed see that the time of the algorithm seems to run polynomially with respect to the number of labels. Moreover, Figure 5.4b shows the same running times but now against the number of triplets. We see that this follows a lower-degree polynomial than in Figure 5.4a. Comparing the practical runtimes to the theoretical runtimes, the degree of the polynomials differs significantly. With respect to the number of labels, as well as with respect to the number of triplets, the degree is nearly half of the theoretical runtime's degree. Therefore, based on these datasets, the algorithm runs much faster than the theoretical runtime in practice.



(a) The running time against the number of labels.



(b) The running time against the number of triplets.

Figure 5.4: Two plots of the same running times obtained using the reconstruction algorithm for multifurcating trees. Here (a) shows the running time against the number of labels, while (b) shows the running time against the number of triplets.

It is also interesting to note that the running time is not, in fact, influenced as much by the structure of the tree as it is by the number of labels. One way to visualise this is by looking at the running time against the max depth of the tree. Figure 5.5 plots these points where the colour corresponds to the number of labels of that tree. As can be seen, given a maximum depth, the running time is not governed by the maximum depth but rather by the number of labels.

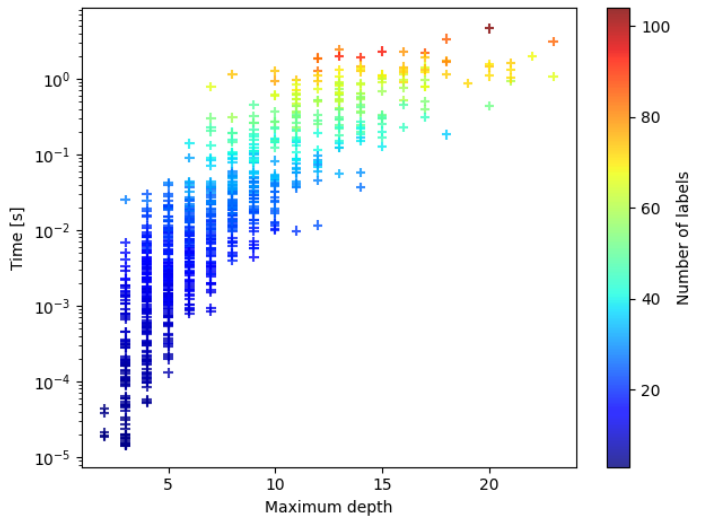


Figure 5.5: The running time of the reconstruction algorithm for multifurcating trees against the max depth of the tree.

Recall that the algorithm also works on fractional inputs. It then outputs a consistent network, which may not be unique in general. Some interesting observations can be made about the algorithm's running time on partial triplet sets. Figure 5.6 shows the fractional runtime of the same multifurcating trees as in Figures 5.4 and 5.5. The fractional runtime for a given $\alpha \in [0,1]$ is calculated by taking a random sample of $\alpha|t(T)|$ triplets and timing the reconstruction algorithm. Per tree, this is repeated five times, after which the quickest runtime is chosen. Note that during each of these five repetitions, a different random sample is chosen.

Somewhat unexpectedly, considering the theoretical runtime, we can see in Figure 5.6 that the runtime for $\alpha = 0.8$ or $\alpha = 0.6$ is actually higher than when using the full triplet set. Likely, this is due to missing some vital triplets that convey specific information, causing the algorithm to loop over all triplets more frequently instead of stopping when a specific triplet has been found. This is further supported by the fact that the fractional time does not approach $\alpha^{\frac{7}{3}}$ as we might expect based on the theoretical runtime of $\mathcal{O}(|t(T)|^{\frac{7}{3}})$. This indicates that using the full triplet set, generally, allows the algorithm to run faster than the worst-case scenario, as the required triplet is always present to allow a loop to stop early. However, we see that for $\alpha < 0.6$, a decrease in running time is obtained for enough labels. The small number of triplets seems to outweigh the additional runtime due to having to loop over the inputted triplet set more often. Lastly, for few enough labels, the runtime is rather spread out, which might be because the original runtime was already low; therefore, the overhead time may have a larger effect on the fractional runtime calculated.

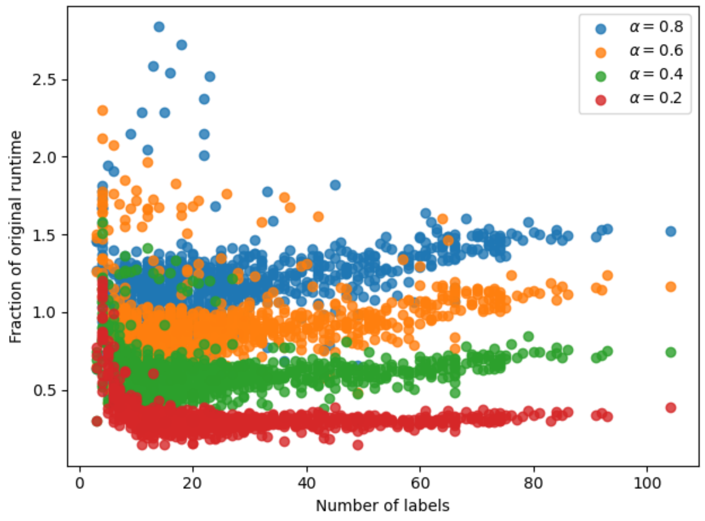


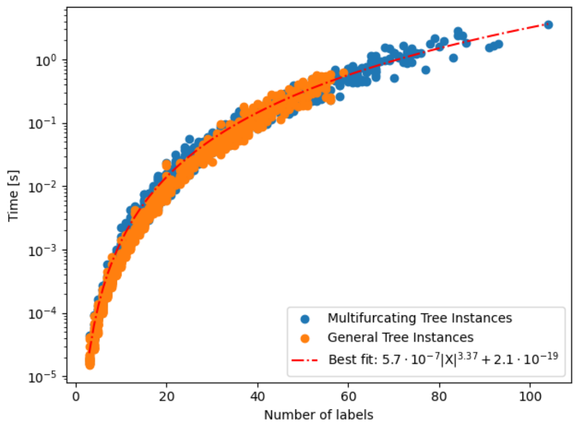
Figure 5.6: The fractional running time of the reconstruction algorithm for multifurcating trees for a subset of the triplet sets. α indicates the fraction of triplets used.

5.3. General Tree Reconstruction

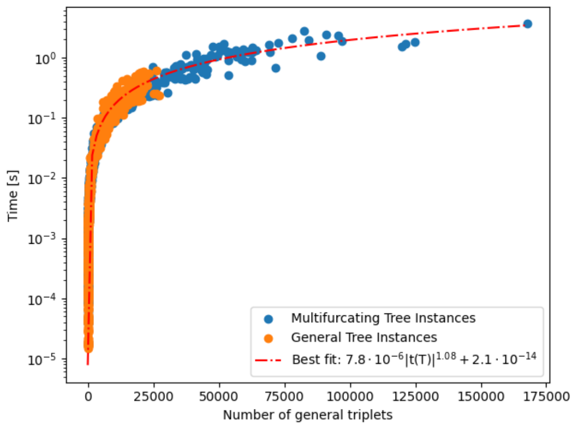
The running time for the reconstruction algorithm for general trees is $\mathcal{O}(|t(T)|^2|X|^3)$ as was shown in Section 3.2.2. However, as we look at the running time for full triplet sets, we can assume the theoretical running time to be $\mathcal{O}(|t(T)|^2|X|^2)$ as was also argued in Section 3.2.2. This algorithm can also handle multifurcating trees, although the theoretical running time is higher than that of the multifurcating tree reconstruction algorithm. Again, the running time can be expressed only in terms of $|X|$ or $|t(T)|$, which gives $\mathcal{O}(|X|^8)$ and $\mathcal{O}(|t(T)|^{\frac{8}{3}})$.

We wish to compare the theoretical and practical running time of the reconstruction algorithm for general trees on randomly obtained multifurcating and general trees. Figure 5.7 shows the practical running time with respect to the number of labels and the number of triplets for both multifurcating and general tree instances. Again, we can see that the running time follows a lower-order polynomial with respect to the number of triplets than to the number of labels. This seems consistent with the theoretical running times of $\mathcal{O}(|t(T)|^{\frac{8}{3}})$ and $\mathcal{O}(|X|^8)$, respectively. Note that the plot shows both the running times of the algorithm for multifurcating trees and general trees. We also see that the practical running times follow a lower-order polynomial than the theoretical running time suggests. This suggests that, just like for the multifurcating reconstruction algorithm, the running time in practice is faster than the theory suggests.

Again, we can see that the runtime does not depend on the structure of the tree. This is further supported by Figure 5.8, where the runtime is plotted against the maximum depth of the trees. It shows the runtime for both multifurcating trees and general trees. The colours correspond to the number of labels and clearly show that the number of labels more clearly defines the runtime than the depth of the tree.



(a) The running time against the number of labels.



(b) The running time against the number of triplets.

Figure 5.7: Two plots of the same running times obtained using the reconstruction algorithm for general trees on multifurcating and general tree instances. Here (a) shows the running time against the number of labels, while (b) shows the running time against the number of triplets.

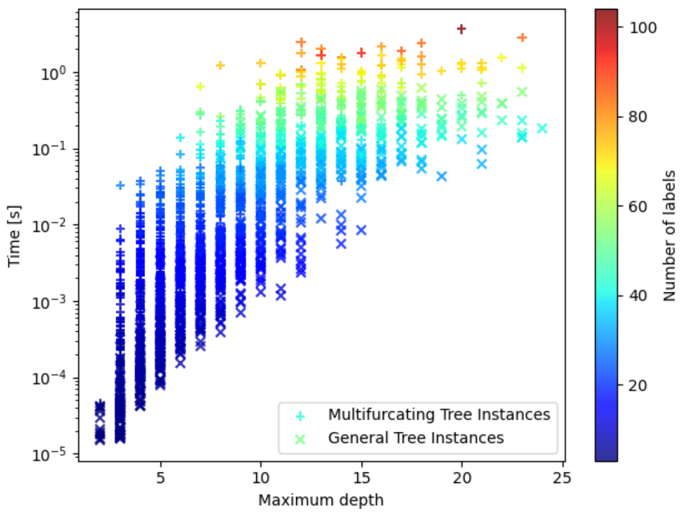


Figure 5.8: The running time of the reconstruction algorithm for general trees against the max depth of the tree for both multifurcating and general trees.

As this algorithm is also capable of reconstructing multifurcating trees, it seems natural to compare the runtime of both algorithms for a given multifurcating tree. Figure 5.9 shows the runtimes of both algorithms for the same multifurcating trees. Contrary to expectations, the running time of the general tree reconstruction algorithm follows a polynomial of a slightly lower order than the reconstruction algorithm for multifurcating trees. Since the theoretical running time is based on the worst-case scenario, this does not suggest that the runtimes are incorrect but rather that for most practical trees with enough labels, Algorithm 2 is faster than Algorithm 1. The advantage of using Algorithm 1 over Algorithm 2 when reconstructing multifurcating trees is that fewer triplets are required to properly reconstruct it. Algorithm 1 only looks at resolved and fanned triplets, while Algorithm 2 also considers the additional triplets as discussed in Section 3.2. Although this does not reduce the running time, as can be seen in Figure 5.9, it does allow a stemmatologist to get a reconstruction of a multifurcating tree while defining fewer triplets by hand.

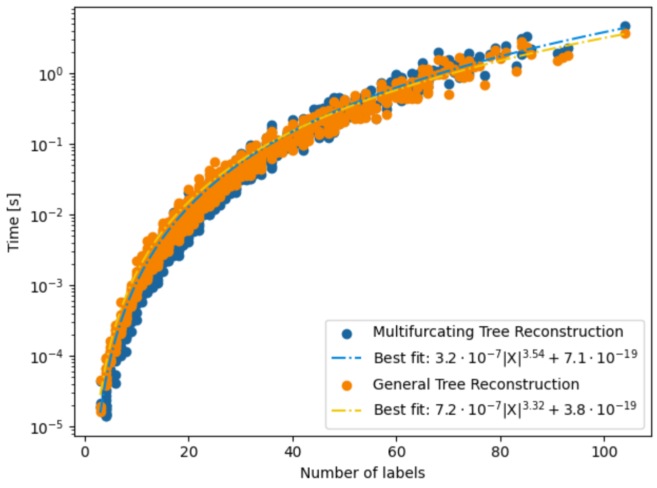


Figure 5.9: The running time of Algorithms 1 and 2 for the same multifurcating trees against the number of labels.

We can again look at the fractional runtime as done in Section 5.2. Figure 5.10 shows the fractional runtime for four values of α. The fractional runtime was obtained using the same method as before. The figure shows similar results to Figure 5.6. Again, the runtime increases when using a larger subset of triplets and decreases when using a smaller subset of triplets.

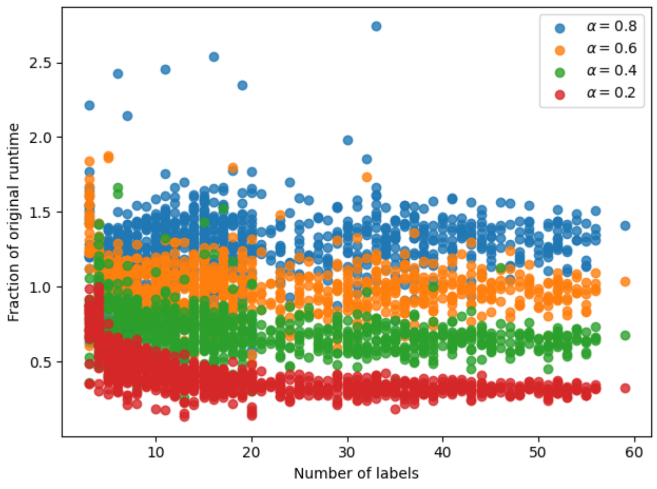


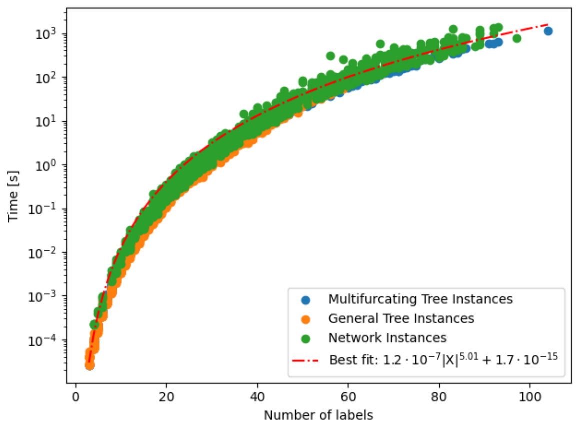
Figure 5.10: The fractional running time of the reconstruction algorithm for general trees for a subset of the triplet sets. α indicates the fraction of triplets used.

5.4. Level-1 Network Reconstruction

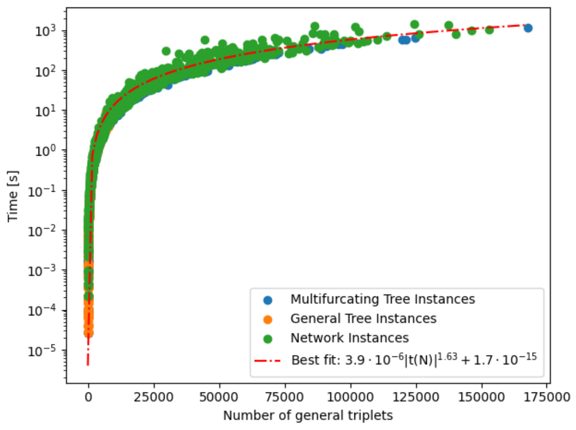
Lastly, the reconstruction algorithm for level-1 networks has a theoretical running time of $\mathcal{O}(|X|^6 + |t(N)|^2|X|^2)$. Expressed only in terms of $|X|$ we obtain a theoretical running time of $\mathcal{O}(|X|^8)$, and in terms of $|t(T)|$ a runtime of $\mathcal{O}(|t(T)|^{\frac{8}{3}})$. This running time is equivalent to the runtime of Algorithm 2 as described in the previous section. However, we already saw that in practice Algorithm 2 runs more quickly than this. For the network reconstruction algorithm, we can see the same, although the difference is slightly less.

Looking at Figure 5.11, we can see that the structure of the network does not seem to affect the runtime of the algorithm. Namely, regardless of running an instance for a multifurcating stemmatic tree, a general stemmatic tree, or a level-1 stemmatic network, the runtime follows the same trend with respect to the number of labels. A slight decrease, although roughly constant, in time may be noted when comparing networks to trees. This is likely due to not having to resolve cycles for trees.

That the structure does not influence the runtime is supported by Figure 5.12. As we can see, the runtime is more heavily determined by the number of labels than the structure properties. The number of cycles and the maximum cycle size do seem to have slightly more influence on the running time than the maximum depth. This is most likely due to the high contribution of resolving the cycles to the theoretical runtime.

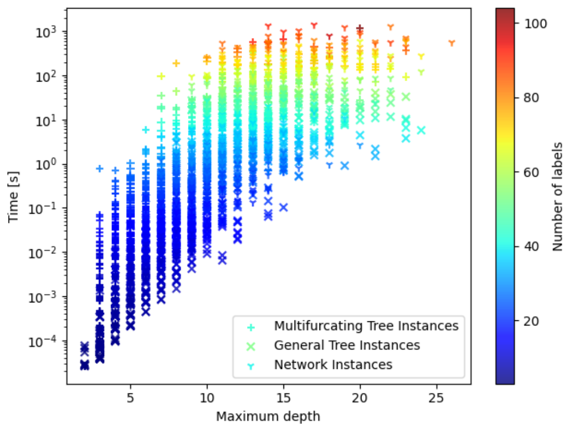


(a) The running time against the number of labels.

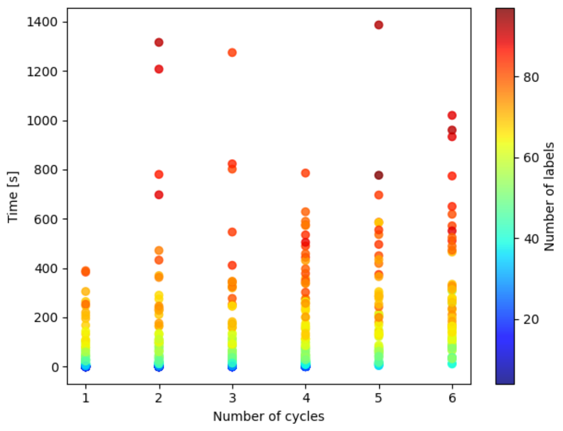


(b) The running time against the number of triplets.

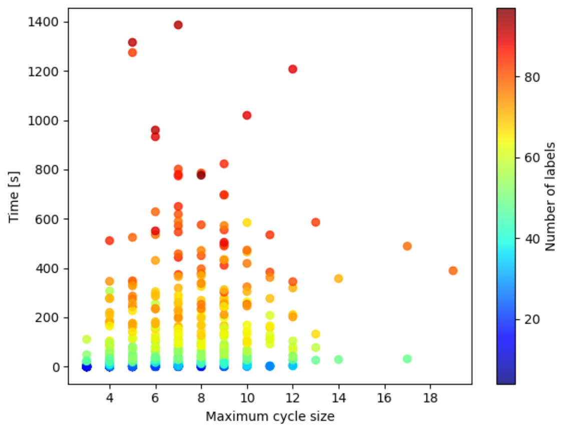
Figure 5.11: Two plots of the same running times obtained using the reconstruction algorithm for networks for all types of networks. Here (a) shows the running time against the number of labels, while (b) shows the running time against the number of triplets.



(a) Runtime against the max depth of the networks and trees.



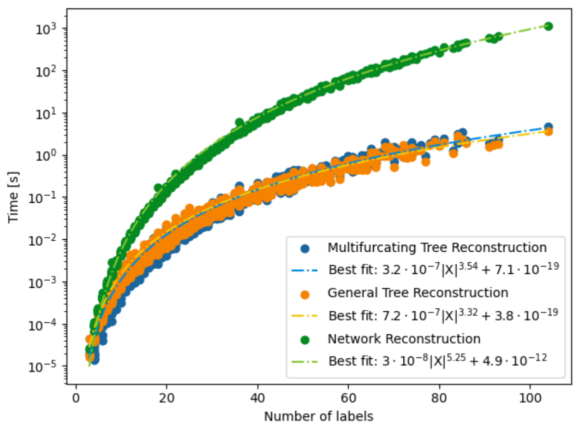
(b) Runtime against the number of cycles in the networks.



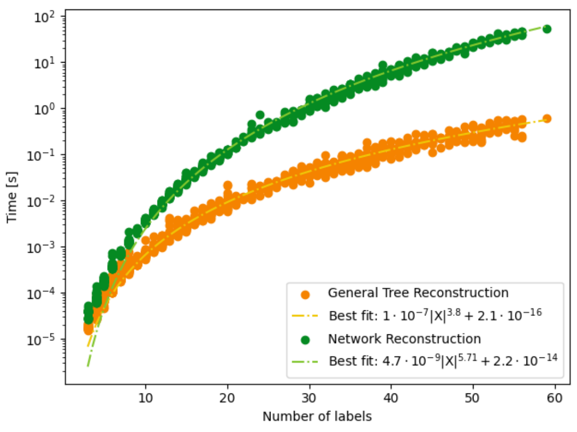
(c) Runtime against the maximum cycle size of the networks.

Figure 5.12: The runtime of Algorithm 3 against different structure properties of the networks.

The advantage of using Algorithm 2 over Algorithm 3 for a quicker runtime is clearly visualised in Figure 5.13. It can be seen that for both multifurcating trees (Figure 5.13a) and general trees (Figure 5.13b), using their respective algorithms is faster than using the algorithm for level-1 networks. Figure 5.13 together with Figure 5.11a also further show how the structure of the instance does not heavily influence the running time of Algorithm 3. Namely, the degrees of the lines of best fit do not change significantly for the different types of graphs.



(a) The running time against the number of labels for multifurcating trees using different algorithms.



(b) The running time against the number of labels for general trees using different algorithms.

Figure 5.13: Two plots of the running times obtained using different algorithms against the number of labels for multifurcating and general trees.

6 Triplet Distance Metric

As discussed in Section 2.5 the proposed measures in Equations (2.1) and (2.2) are indeed metrics for the networks as proposed in Sections 3.1 and 3.2 and Chapter 4 since it has now been proven that triplets encode the networks. In this section, the behaviour of this metric is studied by looking at how it changes based on specific SPR moves on synthetic data in Section 6.1. Ciccolella et al., 2021 also looked at how SPR moves affected their triplet distances. We will look at similar moves but on slightly more involved trees. Moreover, the obtained triplet distances are compared with other measures and metrics used in literature in Section 6.2

6.1. SPR Moves

Subtree pruning and regrafting moves, or SPR moves, are tree rearrangement moves used commonly in maximum parsimony or maximum likelihood searches for phylogenetic trees (Hordijk & Gascuel, 2005; Stamatakis & Alachiotis, 2010; Janssen, 2021, Section 1.3.2). When performing an SPR move on a tree, one removes a subtree and reattaches it elsewhere to the remaining tree. Other tree rearrangement moves include nearest neighbour interchange (NNI) moves, and tree bisection and reconnection (TBR) moves.

The SPR move is also defined for phylogenetic networks by Janssen, 2021. Since an SPR move can change the level of the network, one must be careful to ensure the resulting network is still level-1 to be able to apply our metric. We will, however, only look at SPR moves on trees for these experiments. Although similar experiments could be done on networks as well.

In this section, controlled SPR moves will be applied to some synthetic datasets in order to look at the behaviour of the triplet distance. To be specific, a given subtree will be removed and reattached at various points in the remaining tree through the use of tail moves. It is expected that the triplet distance increases the further the subnetwork is reattached in terms of the length of the path between its original and new position.

Definition 6.1. Let $T = (V, E, l)$ be a tree, $v \in V$ any node, and $uw \in E$ an edge such that w is not a descendant of v or v itself. Removing the edge uw and the incoming edge of v, adding an unlabelled node, say l, and ul, lw, and lv as edges, and "cleaning up" the network according to the rules mentioned in Definition 2.2 is called an *SPR move*. T' is the obtained network after performing such an SPR move. This move is denoted by $\text{SPR}(T, v, uw)$.

An SPR move for trees with internal labels is formally defined by Definition 6.1. This definition could be extended to networks with internal labels as well by removing all incoming edges of v. Since we are dealing with trees with internal labels, the definition as proposed by Janssen is slightly different, as their definition would also move the label of v's parent. The SPR move length for a given $\text{SPR}(T, v, uw)$ is defined as the shortest length of a path from w to v in the undirected graph of T. Note that this length is different from the SPR distance, which is the number of SPR moves needed to morph one network into another.

Figure 6.1 shows two examples of SPR moves. Note that the SPR move length of the top move is 4 and the SPR move length of the bottom move is 3. The SPR move at the top shows how Definition 6.1 differs from the definition in Janssen, 2021. According to their definition, l would have been labelled by e, and the node originally belonging to e would have been suppressed.

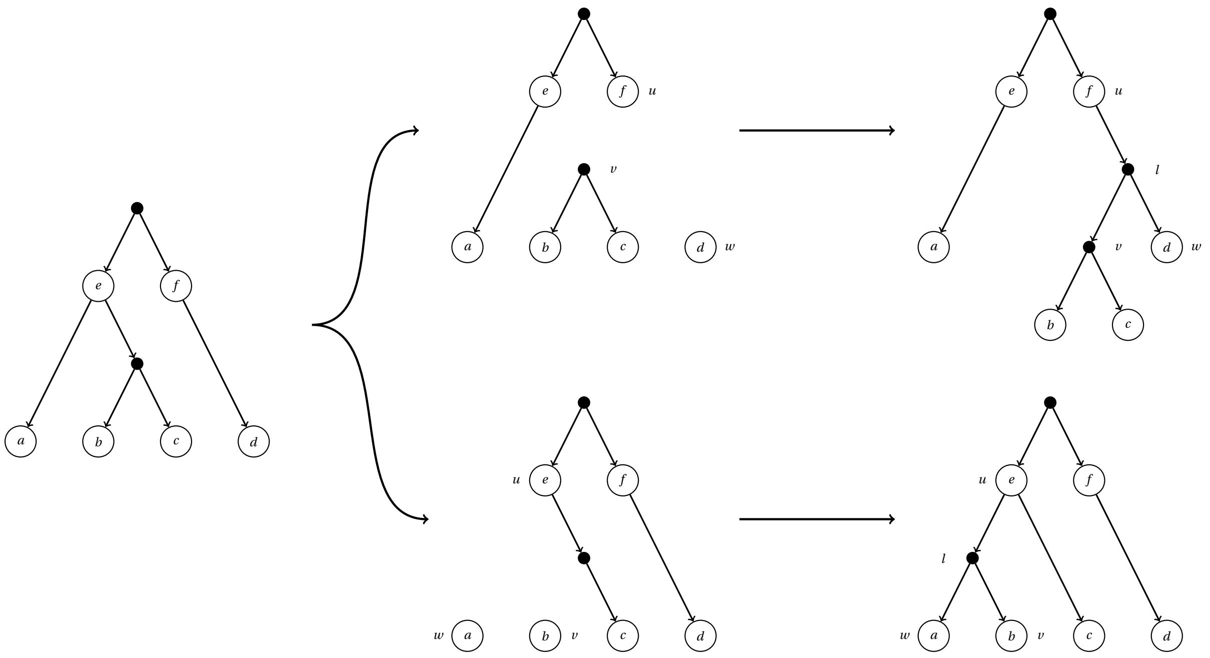


Figure 6.1: Two distinct SPR moves made on the same tree. The relevant u, v, w, and l are shown per move. After the first arrow, the edges are removed, and after the second arrow, the newly added unlabelled node and edges are included, and the tree has been "cleaned up".

First, we will look at how the triplet distance changes when applying SPR moves on a fully-labelled caterpillar tree. A caterpillar tree is a tree obtained by taking a central path and adding any number of leaf nodes to any of the path's vertices. The caterpillar we used has a central path of length h, and each of the path's vertices has r leaf children. Given that tree, T_0, and a $v \in V$ we construct a sequence of trees $T_1^v, T_2^v, \dots$ as follows. Let w_1 be the parent of v and u_1 the parent of w_1. Now take $w_i = u_{i-1}$ and u_i the parent of u_{i-1}. Then T_i is obtained by applying SPR$(T_0, v, u_i w_i)$. We stop when u_i no longer has a parent. The triplet distances between T_0 and the obtained trees, $T_1^v, T_2^v, \dots$, are then calculated. We first pick v to be the leaf node furthest from the root. This process is then repeated by taking v' to be v's grandparent, creating the sequence of trees $T_1^{v'}, T_2^{v'}, \dots$, and calculating the triplet distances. We keep repeating this until the grandparent of the last v is the root or does not have a grandparent. To denote the number of labels that are descendants of v, including v itself, we use $n \in \mathbb{N}$. This gives us Figure 6.2 for a caterpillar tree with $h = 15$ and $r = 4$.

The expectation that the triplet distance increases the higher the SPR move length and the larger the size of the subtree moved seems to be correct. If, however, we look at a partially-labelled caterpillar, we see that this expectation is not always correct. Figure 6.3 shows the triplet distance for several SPR moves on a partially-labelled caterpillar. We randomly selected half the internal nodes to be labelled. We still see that the triplet distance increases as the SPR move length increases. However, it is not necessarily the case that the triplet distance increases for a larger subtree. For example, when comparing the SPR move of length 1 for $n = 14$ with the same SPR move for $n = 28$, we see that the triplet distance is larger for the first than for the second. Most likely, this is due to the number of labels on the central path above and below the root of the subtree. The more labelled ancestors the root of the subtree has, the higher the triplet distance after an SPR move. If there are few labelled ancestors, the triplet distance is less as there are fewer labels whose triplets might be affected.

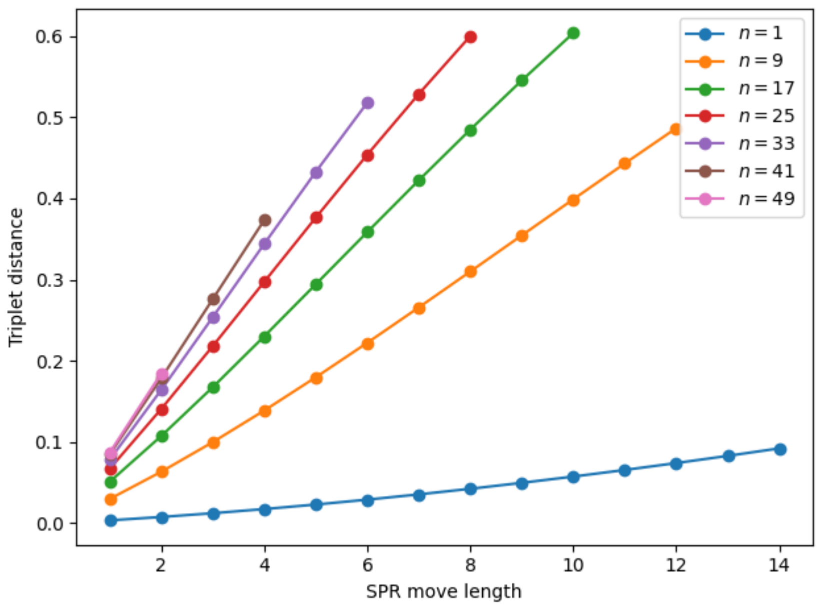


Figure 6.2: The triplet distance for a fully labelled caterpillar tree ($|X| = |V| = 61$) with $h = 15$ and $r = 4$ for different subtrees of n labels and SPR move lengths.

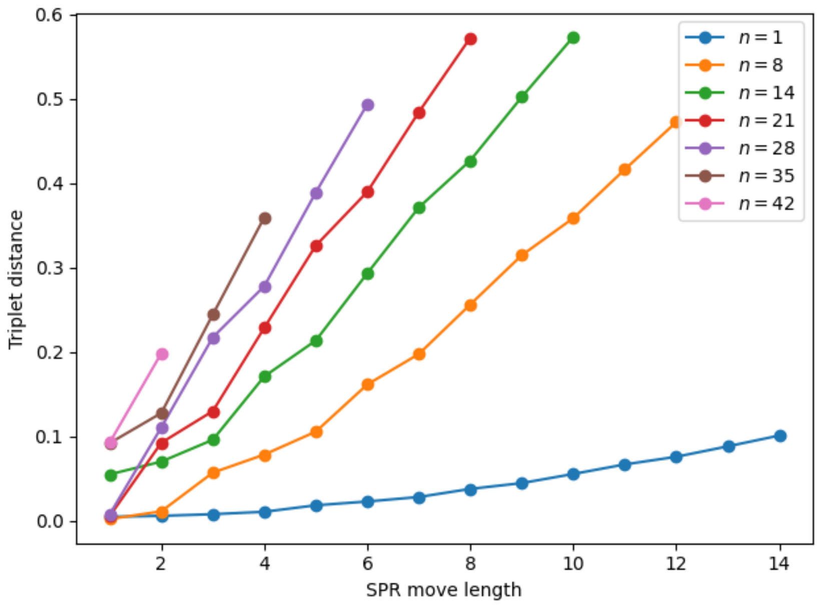


Figure 6.3: The triplet distance for a partially-labelled caterpillar tree ($|X| = 54$, $|V| = 61$) with $h = 15$ and $r = 4$ for different subtrees of n labels and SPR move lengths.

We can also look at a balanced tree, which is obtained by starting with one node and iteratively adding r children to all leaf nodes h times. The height of this tree is h, and every internal node has out-degree r. Figure 6.4 shows the triplet distance for a fully-labelled balanced tree with $h = 6$ and $r = 2$. Due to limited computing power and available RAM, higher values for h or r were not achievable. The edges used for the SPR moves are all the edges on the path from the root to the subtree's root. We again see a clear monotonic increase in the triplet distance as the SPR move length or the number of labels in the subtree increases.

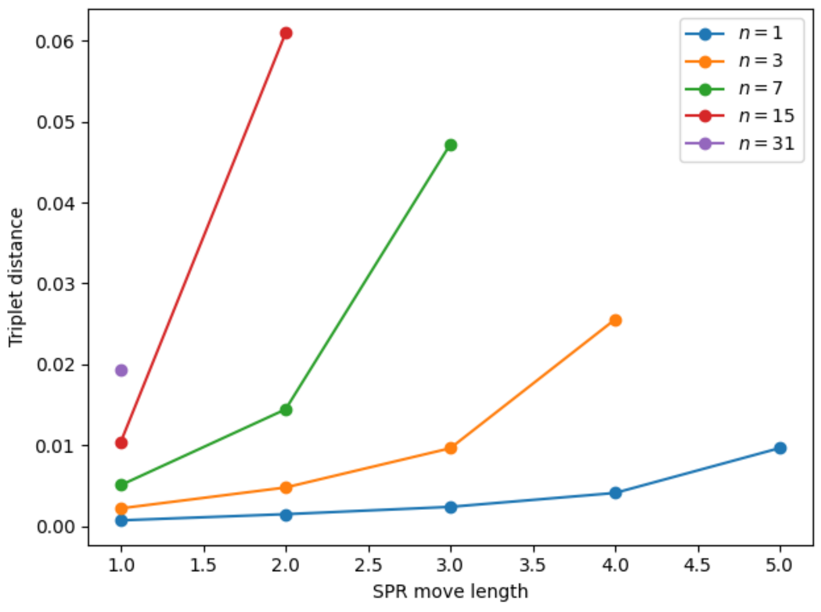


Figure 6.4: The triplet distance for a fully-labelled balanced tree ($|X| = |V| = 127$) with $h = 6$ and $r = 2$ for different subtrees of n labels and SPR move lengths.

6.2. Triplet Distance Comparison on Real Datasets

To understand how this metric behaves in more practical circumstances, we will apply it to a real dataset. We will use the artificial "Notre besoin" tradition created by Macé et al. (2006). It was obtained by having 11 scribes copy either the original or someone else's copy by hand. The true stemma had an unlabelled node with out-degree one corresponding to a lost manuscript whose existence was known. To make it satisfy our definitions, we suppressed this node. The updated stemma of this tradition, which we take to be true for these experiments, is depicted in Figure 6.5. The unlabelled node we removed was a child of U and the parent of B.

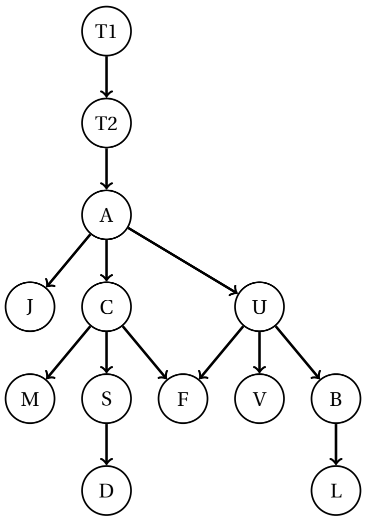
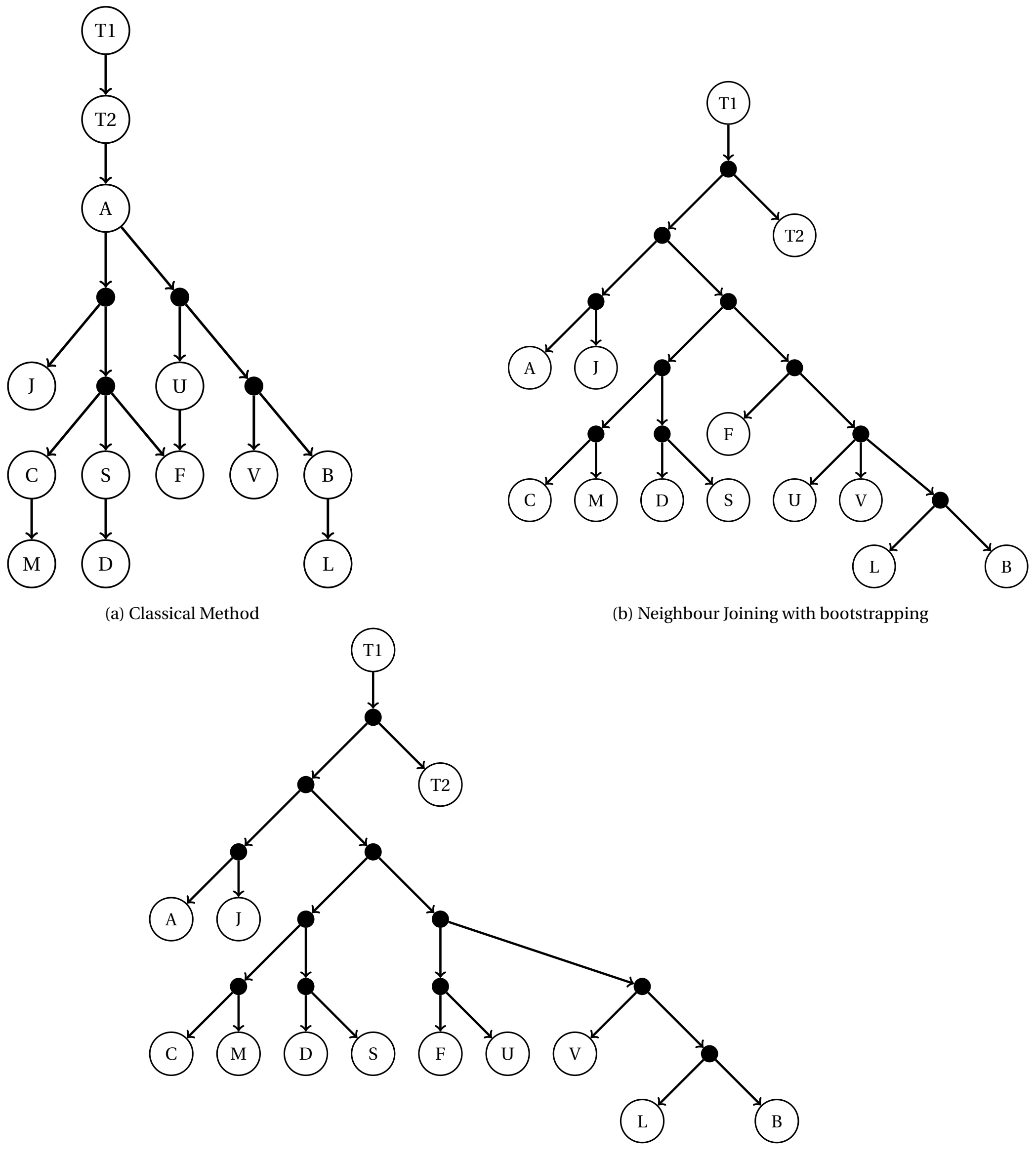


Figure 6.5: The updated stemma of the Notre besoin set, which we take as the true stemma (Macé et al., 2006).

Roos and Heikkila (2009) have compared different methods to reconstruct several datasets, including the "Notre Besoin" tradition. They used the *average sign distance* to determine the effectiveness of their methods. The three highest scoring methods were a manually constructed solution using a classical method, Neighbour Joining with bootstrapping, and RHM. Neighbour Joining is a distance-based method, where two taxa with



(c) RHM

Figure 6.6: The obtained networks using different methods based on the "Notre besoin" tradition (Roos & Heikkila, 2009).

the shortest distance are joined by a newly added node. RHM is a method proposed by Roos et al. (2006) and is based on stochastic optimisation. The stemmata obtained through these methods are shown in Figure 6.6.

We will compare the triplet distance to other measures and metrics on these networks. The measures used in this comparison are the average sign, Robinson-Foulds, tripartition, and μ-distances. All distance functions are normalised and thus return values in $[0,1]$; where 0 means that the two trees are the same (with respect to the properties compared) and 1 means that the two trees do not have any similarities (with respect to the properties compared). We will define these measures below. Note that for these distances it has not been shown that they are in fact metrics for the networks as defined by Definition 4.1. Therefore, we simply refer to them as measures instead of metrics.

The average sign distance is based on comparing the distance between three labels in the two networks, N and N'. Suppose $a, b, c \in X$, then let $d_N(a,b)$ denote the length of the shortest path between a and b in the underlying undirected network of N. We define $u(a,b,c)$ as follows

$$u(a,b,c) = \frac{1}{2}\left|\operatorname{sign}\left(d_N(a,b) - d_N(a,c)\right) - \operatorname{sign}\left(d_{N'}(a,b) - d_{N'}(a,c)\right)\right|$$

where sign(x) is $+1$ if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$. The average sign distance is then defined as the average of all $u(a,b,c)$ for any three $a,b,c \in X$. Note that we look at all permutations of any three labels. The values for the average sign distance in Table 6.1 slightly differ from those in Roos and Heikkila (2009) due to the removal of the unlabelled node.

The Robinson-Foulds distance between two networks, $N = (V,E,l)$ and $N' = (V',E',l')$, is defined in Cardona, Llabres, et al. (2009) using the sets of leaf-descendants for any $v \in V$ in N including v itself if v is a leaf, denoted as $C_N(v)$. Using $C(N) = \{C_N(v) \mid v \in V\}$, the Robinson-Foulds distance is then defined as $|C(N)\Delta C(N')|$. This measure focuses on descendants rather than relative distances between labels, as the average sign distance does. To make it applicable for internal labels as well, we define $C_N(v)$ as the set of labels that are descendants of v, including the label of v itself in N. To normalize this measure we divide $|C(N)\Delta C(N')|$ by $|C(N) \cup C(N')|$ as described by Yianilos (2002).

Cardona, Llabres, et al. also redefine the tripartition distance, originally shown by Moret et al. (2004) as a metric for phylogenetic networks. It relies on the distinction between strict descendants and non-strict descendants. Strict descendants of a node $v \in V$ are all labelled descendants of v, including v itself, such that any path from the root to that labelled descendant contains v. For non-strict descendants, on the other hand, there exists a path from the root to that descendant that does not contain v. Let $A_N(v)$ denote the set of labels that are strict descendants of v in N and $B_N(v)$ denote the set of labels that are non-strict descendants of v in N. Then $\theta_N(v) = (A_N(v), B_N(v))$ is the tripartition of v. Moret et al. actually defined $\theta_N(v)$ as $(A_N(v), B_N(v), X \setminus C_N(v))$, hence the name tripartition. But as Cardona, Llabres, et al. point out, including $X \setminus C_N(v)$ in the tripartition is redundant. The tripartition representation of N is the set $\theta(N) = \{\theta_N(v) \mid v \in V\}$, and the tripartition distance is defined as $|\theta(N)\Delta\theta(N')|$. Again, we normalise this distance by dividing it by $|\theta(N) \cup \theta(N')|$.

Lastly, the μ-distance is originally defined by Cardona, Rossello, and Valiente (2009). It relies on the number of paths in the directed network N from any vertex to every label. Let $X = \{1,2,\dots,n\}$ and $m_i(v)$ denote the number of directed paths from v to i in N. We define the μ-vector, or path-multiplicity vector, of $v \in V$ as $\mu_N(v) = (m_1(v), m_2(v), \dots, m_n(v))$. Then the μ-representation of N is the multiset $\mu(N) = \{\mu_N(v) \mid v \in V\}$. The μ-distance is $|\mu(N)\Delta\mu(N')|$ where Δ is the symmetric difference of multisets; if a vector belongs to $\mu(N)$ with multiplicity a and belongs to $\mu(N')$ with multiplicity b, then that vector belongs to $\mu(N)\Delta\mu(N')$ with multiplicity $|a-b|$. We can again normalize the distance by dividing it by $|\mu(N) \cup \mu(N')|$, where the union of multisets A and B is defined such that the multiplicity of any element e from A or B in $A \cup B$ is the maximum multiplicity of e in A and B.

Note that the tripartition and μ-distances both give the Robinson-Foulds distance when limiting ourselves to trees (Cardona, Llabres, et al., 2009). Indeed, $B_N(v) = \emptyset$ for any v if N is a tree and thus $A_N(v) = C_N(v)$ in that case and $m_i(v) = 1$ if $i \in C_N(v)$ and 0 otherwise.

Comparing these metrics to one another is difficult, as they all return a score based on different properties of the two networks. For example, the average sign distance looks at the difference in path lengths from a node to any two different nodes in both networks. Meanwhile, the Robinson-Foulds distance looks at the sets of descendants of all nodes. Therefore, if one measure gives a lower distance than another, this does not tell us much about which measure is more accurate or useful. We can only compare them based on trends to see what one measure might weigh more heavily than another. Moreover, the average sign and triplet distances only look at labelled nodes, while the other distances also look at unlabelled nodes. However, since the average sign distance is based on the relative distances between three labels, just like the triplet distance, these two measures can be considered most alike in the properties they measure.

Table 6.1 shows the triplet distance between the networks in Figure 6.6 and the true stemma in Figure 6.5. Looking at these results, we can note that the triplet distance seems most conservative of all the measures looked at. It can be observed that the Robinson-Foulds, tripartition, and μ-distance are all fairly close, likely due to how they all concern the descendant sets of all vertices and are equivalent when only looking at trees. The average sign distance, however, gives the lowest scores among all obtained stemmata. As discussed, this distance focuses on the relative distances between three labels, and thus, we can expect that if the main clusters of a network are kept together, the distance will likely give good results. Indeed, we could, intuitively, divide the labels in Figure 6.5 into three clusters: $\{T1, T2\}$, $\{A, J, C, M, S, D\}$, and $\{U, F, V, B, L\}$. Although F could

Distance	Method		
	Classical Method (Figure 6.6a)	Neighbour Joining (Figure 6.6b)	RHM (Figure 6.6c)
Triplet	0.519	0.901	0.901
Average sign	0.145	0.229	0.235
Robinson-Foulds	0.235	0.500	0.520
Tripartition	0.235	0.560	0.577
μ	0.235	0.667	0.679

Table 6.1: The distances between the stemma considered to be true for this experiment and the obtained stemmata in Figure 6.6.

be placed in either of the last two clusters. When looking at the obtained networks in Figure 6.6, these same clusters could be identified where F is placed in between the last two clusters.

The high triplet distances are likely due to the lack of the additional triplets introduced in this thesis. Namely, the triplets of the forms $u/v \setminus w$, $u|v \setminus w$, and $u \setminus v \setminus w$. The trees in Figures 6.6b and 6.6c do not have any such triplets, while the network in Figure 6.6a has fewer triplets of those forms than the original network. For example, none of the networks in Figure 6.6 have the triplets $M/C \setminus S$, $V/U/A$, or $F/C|U$ which are all present in Figure 6.5.

To better understand which properties the triplet distance weighs more heavily, we will also examine the distances between the obtained networks. By doing so, we aim to discover more trends in the differences between the measures. These distances are shown in Table 6.2. Indeed, the tripartition and μ-distances yield the same results as the Robinson-Foulds distance for trees.

Again, the triplet distance is highest when comparing the classically obtained stemma to the ones in Figures 6.6b and 6.6c. Meanwhile, the triplet distance between the stemma obtained through neighbour joining with bootstrapping and RHM is the lowest distance except for the average sign distance. However, the average sign distance also gives the lowest distances of all the measures when comparing the classically obtained stemma with the ones obtained through Neighbour Joining and RHM. This further suggests that the lack of the additional triplets when comparing phylogenetic trees or networks with stemmata results in high distances when using the triplet distance. Or, equivalently, that incorrectly placing internal nodes with relatively many labelled descendants as leaf nodes causes a high triplet distance.

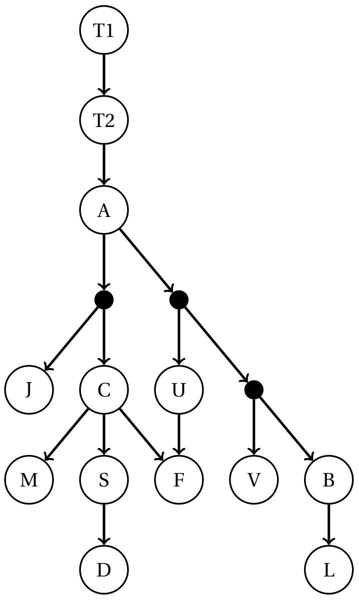
Distance	Networks		
	Classical Method vs Neighbour Joining (Figures 6.6a and 6.6b)	Classical Method vs RHM (Figures 6.6a and 6.6c)	Neighbour Joining vs RHM (Figures 6.6b and 6.6c)
Triplet	0.803	0.806	0.044
Average sign	0.240	0.247	0.037
Robinson-Foulds	0.519	0.423	0.120
Tripartition	0.571	0.536	0.120
μ	0.667	0.586	0.120

Table 6.2: The distances between the obtained stemmata in Figure 6.6.

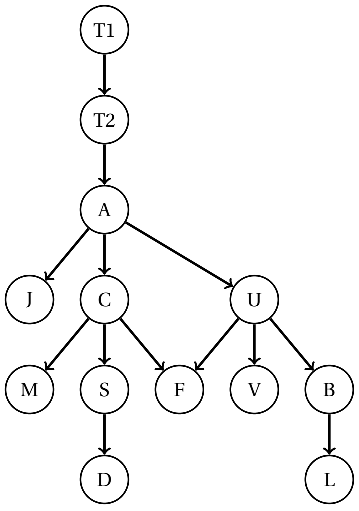
Indeed, we can slightly alter the classically obtained stemma to include C as a cycle vertex again by contracting the incoming edge of C (Figure 6.7a), which results in a network that more closely resembles the true stemma in Figures 6.5 and 6.7b. One might be prompted to do so based on additional research after creating the stemma, or because new facts have come to light. This gives us quite different results. The triplet distance between the true stemma and this newly created stemma is now 0.375, while the Robinson-Foulds, tripartition, and μ-distances are all 0.188, and the average sign distance is 0.111. That is a decrease of the distance of 27.8% for the triplet distance in comparison with the true stemma and the classically obtained stemma. Meanwhile, the average sign distance decreased by 23.4% and the other distances only by 20.0%.

Similarly, the triplet distance between the networks in Figures 6.6a and 6.7a is 0.212. While the average sign distance is 0.055 and the Robinson-Foulds, tripartition, and μ-distances are all 0.059. Again, the triplet distance is rather high due to only one contraction, while the other distances are all relatively low.

This all shows how the triplet distance punishes the incorrect placement of internal labels with relatively many labelled descendants more heavily than other distances. Or, in other words, that a high triplet distance may suggest that an internal node with relatively many labelled descendants has incorrectly been placed as a leaf node. Indeed, placing an internal label, $u \in X$, as a leaf child of its original vertex, $v \in V$, causes all triplets containing that label and one of its original descendants to be different, while the total number of triplets does not change much. Meanwhile, for the average sign distance, the distance from this label to any other label increases by one. Therefore, $u(a,b,u)$ only changes when the distance between a and b is one longer than the original distance between a and u. This can cause a smaller change in the score than for the triplet distance. Likewise, for the remaining distances only $C_N(v)$, $\theta_N(v)$, and $\mu_N(v)$ change and an additional set is added due to the new vertex. The properties for all the other vertices remain the same, and thus the score is not impacted as harshly.



(a) The manually improved stemma based on Figure 6.6a.



(b) The stemma we take to be true (the same as Figure 6.5).

Figure 6.7: The manually improved stemma and the stemma we take to be true.

7 Conclusion

7.1. Summary of results

In this thesis, we investigated the problem of encoding stemmata using triplets and explored the use of triplets as a distance metric. We addressed this question for three different classes of graphs with possible internal labels: multifurcating trees, general trees, and level-1 networks. In each case, we provided a formal proof that the full set of triplets uniquely determines the structure up to isomorphism. Building on these encoding theorems, we presented corresponding algorithms that reconstruct the full graph from its triplet set. These algorithms extend earlier work by Aho et al. (1981) and Ng and Wormald (1996), among others, by allowing for more extensive structures, including multifurcations, reticulations, and internal labels. Additionally, we evaluated these algorithms computationally and demonstrated that they efficiently reconstruct trees and level-1 networks based on triplet sets in practical instances.

Besides these foundational contributions, we examined the use of the triplet distance as a metric for comparing graphs. This distance, which counts the number of differing triplets between two graphs, was originally proposed for trees but can be naturally extended to networks. In Chapter 6, we applied this metric to both synthetic and real-world data, and our results show that it provides an intuitive and meaningful way to assess dissimilarity between reconstructed and ground truth structures.

In the case of level-1 networks, the encoding result is even more noteworthy, as it shows that a structure with reticulations — allowing for a single cycle in each biconnected component — can still be uniquely determined by its triplets. Previous work by Gambette and Huber (2012) emphasised the importance of characterising when such encodings are possible. As the level of networks increases, the difficulty of finding encodings increases too. This encoding result for level-1 networks allows for the use of triplets in cases where some limited horizontal transmission or contamination is expected, and serves as a stepping stone to extend the encoding results to even higher-level networks, although this may require more complex building blocks like trinets, quartets, or quarnets.

In terms of algorithmic development, our reconstruction methods generalise and extend classical algorithms in the literature. The multifurcating and general tree reconstruction algorithms build on Aho et al. (1981), while introducing new mechanisms to handle internal labels and out-degree one nodes. Compared to the algorithm of Ng and Wormald (1996), which is designed for multifurcating phylogenies, our method is more broadly applicable and offers more compact representations by exploiting internal labelling. For level-1 networks, our algorithm complements the works of Jansson and Sung (2006), Huber et al. (2011), Gambette et al. (2017), and van Iersel and Kelk (2008), who have come up with algorithms that find phylogenetic level-1 networks consistent with a set of triplets. Our contribution lies in providing a practical algorithm that can work with internal labels and nodes with arbitrary out-degree.

These results have immediate practical relevance in fields such as stemmatics and historical linguistics. Triplet-based methods provide a structured and theoretically justified way to model local textual relationships. Stemmatologists and linguists can use observations on textual witnesses to create triplets and combine those to form their stemma. The current difficulties of this approach are further discussed in Section 7.3.

7.2. Discussion

While the theoretical results presented in this thesis are strong, several limitations and open questions remain. The empirical evaluation of the triplet distance, for instance, was limited in scope. We tested it on only a small number of trees and applied a relatively modest number of Subtree Prune and Regraft (SPR) moves. Likewise, the distance was only applied to a single real dataset, which itself was of limited size and only compared to a handful of other metrics. Although initial results were promising, this narrow sample limits the conclusions that can be drawn about the general behaviour of the triplet distance under structural perturbations. A more systematic study is required to evaluate its sensitivity, robustness, and correlation with other measures across a broader range of topologies and perturbation models, which should also be applied on networks.

Another important observation concerns the performance of the level-1 reconstruction algorithm. Despite having a theoretical time complexity comparable to our general tree algorithm, its implementation was significantly slower in practice. This discrepancy suggests that the algorithmic design may not yet be optimally efficient. The slower performance may reflect non-optimal calculation workflows or a need for better data structures and implementation strategies. Given that performance is critical for working with large textual corpora or linguistic datasets in practice, improving this aspect of the algorithm is an important direction for future work.

In addition to computational considerations, some of the assumptions concerning the structure of our level-1 stemmatic networks may be unnecessarily restrictive. In particular, the requirement that the sinks of cycles in level-1 networks must be labelled could be questioned. Although this assumption is used in our proofs, we were not able to find a counterexample where omitting these labels resulted in two non-isomorphic graphs with the same triplet set. This raises the possibility that the assumption may be relaxed, at least in the case of level-1 networks. Clarifying whether those internal labels are necessary or if a weaker condition suffices is therefore a relevant theoretical question with clear practical implications.

7.3. Future work

Several promising directions for future research emerge from this work and the corresponding reflection on its results. First, we assume that triplet sets are given, while in practice, they must be inferred from textual data. Defining triplets directly from texts remains an open problem. One plausible approach involves aligning three manuscript texts and using a similarity or distance measure (such as Levenshtein or Hamming distance) to infer which two are most closely related. If two texts agree on a variant while the third differs, this may suggest a triplet of the form “AB|C”. Prior work in digital stemmatology has explored pairwise distance metrics between manuscripts (Roos & Heikkila, 2009), and extending these methods to robustly define triplets is a natural next step. Techniques from computational linguistics and natural language processing, including embedding-based similarity or edit operations weighted by textual features, may offer useful tools in this direction.

Second, our encoding result for level-1 networks raises the question of whether analogous results hold for more complex network classes. Level-2 networks, which allow for more intricate cycles, are a natural next step. However, existing literature shows that for phylogenetic level-2 networks, triplet sets alone are insufficient to determine the network structure. To and Habib (2009) and van Iersel et al. (2009), among others, have shown that polynomial algorithms exist to find a phylogenetic level-k network consistent with a given triplet set. However, these networks are not unique. Van Iersel et al. (2022) have shown that so-called “trinets”—subnetworks on three leaves—can encode broader network classes than triplets. Future research might focus on identifying the requirement for labelled internal vertices in level-2 networks such that they are still triplet-encodable, or extend the framework to include trinets or even larger subnetworks as the building blocks of the encoding. Proving encoding theorems in these broader classes would likely require new theoretical tools, but could considerably widen the scope of triplet-based methods.

Third, while our evaluation in Chapter 6 showed that the triplet distance behaves intuitively, a more systematic study is needed to understand its strengths and limitations as a performance metric for reconstruction algorithms. The triplet distance satisfies the properties of a metric and is sensitive to local topological differences, making it appealing for comparing inferred and reference structures. However, its practical effectiveness needs to be benchmarked more extensively, especially in comparison with other distance measures discussed in this thesis, such as the Robinson–Foulds distance, average sign distance, as well as other distances such as the quartet distance, or edit-based distances. Such a study could use simulated data with a known true stemma and controlled noise levels to evaluate how well the triplet distance correlates with reconstruction accuracy. An additional aspect to look at is the use of a weighted triplet distance, where specific triplet forms might be weighted more heavily than others to signify a higher importance. This could possibly give a more balanced measure.

Finally, an important direction is to design algorithms that reconstruct level-1 networks from incomplete triplet sets. In real-world scenarios, the triplet data may be partial or noisy, either because not all manuscript combinations are available or because the input data is ambiguous. While our current algorithm assumes the full triplet set is given, existing work on finding the simplest trees and networks given a triplet set provides a starting point for handling incomplete data (van Iersel & Kelk, 2008). The challenge is to develop algorithms that find a network satisfying as many input triplets as possible, or that optimise a weighted score when triplet reliability varies. This could be approached using heuristics, approximation algorithms, or constraint programming. The ultimate goal would be to build robust methods that still recover meaningful structure even from partial and imperfect input.

To conclude, this thesis has established a rigorous foundation for the use of triplets in the representation and reconstruction of stemmata. By extending triplet encoding to broader classes of trees and networks, and by grounding these methods in algorithms and computational experiments, we have opened the door to practical applications across disciplines. The outlined directions for future research offer a clear roadmap for extending these methods further, both in theory and in applied settings where textual and cultural transmission are studied.

Bibliography

Finlay, G. (2023). Evolution as history: Phylogenetics of genomes and manuscripts. *Christian Perspectives on Science and Technology*, *1*, 150–174. https://doi.org/10.58913/JJHH2131

Hall, A., & Parsons, K. (2013). Making stemmas with small samples, and digital approaches to publishing them: Testing the stemma of konráðs saga keisarasonar. *Digital Medievalist*, *9*(00). https://doi.org/10.16995/dm.51

Barbrook, A. C., Howe, C. J., Blake, N., & Robinson, P. (1998). The phylogeny of the canterbury tales. *Nature*, *394*(6696), 839. https://doi.org/10.1038/29667

Zammit, D. (2024). *Computational stemmatology: Reconstructing text phylogenies through computer assisted methods* [master]. Rijksuniversiteit Groningen. https://fse.studenttheses.ub.rug.nl/33326/

Francois, A. (2015). Trees, waves and linkages: Models of language diversification. In Claire Bowern and Bethwyn Evans (Ed.), *The routledge handbook of historical linguistics* (1st, pp. 161–189, Vol. 1). Routledge, Taylor & Francis Group.

Roelli, P. (2020, September). *Handbook of stemmatology: History, methodology, digital approaches*. De Gruyter. https://doi.org/10.1515/9783110684384

Heikkilä, T. (2022, May). Computer-assisted stemmatology. In *Computer-assisted stemmatology*. Routledge. https://doi.org/10.4324/9780415791182-RMEO364-1

Wallin, R., van Iersel, L., Kelk, S., & Stougie, L. (2021). Applicability of several rooted phylogenetic network algorithms for representing the evolutionary history of sars-cov-2. *BMC Ecology and Evolution*, *21*(1), 220. https://doi.org/10.1186/s12862-021-01946-y

Aho, A. V., Sagiv, Y., Szymanski, T. G., & Ullman, J. D. (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, *10*(3), 405–421. https://doi.org/10.1137/0210030

Ng, M. P., & Wormald, N. C. (1996). Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, *69*(1–2), 19–31. https://doi.org/10.1016/0166-218X(95)00074-2

Jansson, J., & Sung, W.-K. (2006). Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theoretical Computer Science*, *363*(1), 60–68. https://doi.org/10.1016/j.tcs.2006.06.022

van Iersel, L., Kole, S., Moulton, V., & Nipius, L. (2022). An algorithm for reconstructing level-2 phylogenetic networks from trinets. *Information Processing Letters*, *178*, 106300. https://doi.org/10.1016/j.ipl.2022.106300

To, T.-H., & Habib, M. (2009). Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time. In G. Kucherov & E. Ukkonen (Eds.), *Combinatorial pattern matching* (pp. 275–288). Springer Berlin Heidelberg.

Ciccolella, S., Bernardini, G., Denti, L., Bonizzoni, P., Previtali, M., & Della Vedova, G. (2021). Triplet-based similarity score for fully multilabeled trees with poly-occurring labels (A. Elofsson, Ed.). *Bioinformatics*, *37*(2), 178–184. https://doi.org/10.1093/bioinformatics/btaa676

Murakami, Y. (2021). *On phylogenetic encodings and orchard networks* [Doctoral dissertation, Delft University of Technology]. https://doi.org/10.4233/UUID:049932AB-4124-4639-A7E3-146AC4FD805D

Gusfield, D., & Bansal, V. (2005). A fundamental decomposition theory for phylogenetic networks and incompatible characters. In S. Miyano, J. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner, & M. Waterman (Eds.), *Research in computational molecular biology* (pp. 217–232). Springer. https://doi.org/10.1007/11415770_17

Dobson, A. J. (1975). Comparing the shapes of trees. In A. P. Street & W. D. Wallis (Eds.), *Combinatorial mathematics iii* (pp. 95–100). Springer Berlin Heidelberg.

Yianilos, P. N. (2002). *Normalized forms for two common metrics* (tech. rep.). NEC Research Institute.

Brynt, D. (1997). *Building trees, hunting for trees, and comparing trees: Theory and methods in phylogenetic analysis* [Doctoral dissertation, University of Canterburry].

Harvey, D. J., Jansson, J., Marciniak, M., & Murakami, Y. (2024). Resolving unresolved resolved and unresolved triplets consistency problems. In A. A. Rescigno & U. Vaccaro (Eds.), *Combinatorial algorithms* (pp. 193–205). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-63021-7_15

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (Third edition). MIT Press.

Harary, F., & Palmer, E. (1966). The reconstruction of a tree from its maximal subtrees. *Canadian Journal of Mathematics*, *18*, 803–810. https://doi.org/10.4153/CJM-1966-079-8

Gusfield, D., Eddhu, S., & Langley, C. (2004). Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *Journal of Bioinformatics and Computational Biology*, *02*(01), 173–213. https://doi.org/10.1142/S0219720004000521

Gambette, P., & Huber, K. (2012). On encodings of phylogenetic networks of bounded level. *Journal of Mathematical Biology*, *65*(1), 157–180. https://doi.org/10.1007/s00285-011-0456-y

van Iersel, L., Keijsper, J., Kelk, S., Stougie, L., Hagen, F., & Boekhout, T. (2009). Constructing level-2 phylogenetic networks from triplets. *IEEE/ACM transactions on computational biology and bioinformatics*, *6*(4), 667–681. https://doi.org/10.1109/TCBB.2009.22

Huber, K. T., van Iersel, L., Kelk, S., & Suchecki, R. (2011). A practical algorithm for reconstructing level-1 phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *8*(3), 635–649. https://doi.org/10.1109/TCBB.2010.17

Gambette, P., Huber, K. T., & Kelk, S. (2017). On the challenge of reconstructing level-1 phylogenetic networks from triplets and clusters. *Journal of Mathematical Biology*, *74*(7), 1729–1751. https://doi.org/10.1007/s00285-016-1068-3

van Iersel, L., & Kelk, S. (2008). Constructing the simplest possible phylogenetic network from triplets. In S.-H. Hong, H. Nagamochi, & T. Fukunaga (Eds.), *Algorithms and computation* (pp. 472–483). Springer. https://doi.org/10.1007/978-3-540-92182-0_43

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15).

Janssen, R. (2024, July). Phylox: A python package for complete phylogenetic network workflows. https://doi.org/10.5281/zenodo.12742473

Janssen, R., & Murakami, Y. (2020). Linear time algorithm for tree-child network containment. In C. Martín-Vide, M. A. Vega-Rodríguez, & T. Wheeler (Eds.), *Algorithms for computational biology* (pp. 93–107, Vol. 12099). Springer International Publishing. https://doi.org/10.1007/978-3-030-42266-0_8

Hordijk, W., & Gascuel, O. (2005). Improving the efficiency of spr moves in phylogenetic tree search methods based on maximum likelihood. *Bioinformatics*, *21*(24), 4338–4347. https://doi.org/10.1093/bioinformatics/bti713

Stamatakis, A., & Alachiotis, N. (2010). Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data. *Bioinformatics*, *26*(12), i132–i139. https://doi.org/10.1093/bioinformatics/btq205

Janssen, R. (2021). *Rearranging phylogenetic networks* [Doctoral dissertation, Delft University of Technology]. https://doi.org/10.4233/UUID:1B713961-4E6D-4BB5-A7D0-37279084EE57

Macé, C., Peersman, C., Mazza, R., Noret, J., Van Mulken, M., Wattel, E., Canettieri, P., Loreto, V., Lantin, A.-C., Baret, P. V., Robinson, P., Windram, H., Spencer, M., Howe, C., Albu, M., & Dress, A. (2006). Testing methods on an artificially created textual tradition. *Linguistica Computazionale*, *24–25*, 255–283.

Roos, T., & Heikkila, T. (2009). Evaluating methods for computer-assisted stemmatology using artificial benchmark data sets. *Literary and Linguistic Computing*, *24*(4), 417–433. https://doi.org/10.1093/llc/fqp002

Roos, T., Heikkilä, T., & Myllymäki, P. (2006). A compression-based method for stemmatic analysis. *European Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:858094

Cardona, G., Llabres, M., Rossello, F., & Valiente, G. (2009). Metrics for phylogenetic networks i: Generalizations of the robinson-foulds metric. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *6*(1), 46–61. https://doi.org/10.1109/TCBB.2008.70

Moret, B., Nakhleh, L., Warnow, T., Linder, C., Tholse, A., Padolina, A., Sun, J., & Timme, R. (2004). Phylogenetic networks: Modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *1*(1), 13–23. https://doi.org/10.1109/TCBB.2004.10

Cardona, G., Rossello, F., & Valiente, G. (2009). Comparison of tree-child phylogenetic networks. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, *6*(4), 552–569. https://doi.org/10.1109/TCBB.2007.70270

A

Appendix

A.1. Trees

A.1.1. Multifurcating trees

Algorithm 4: Process resolved triplet

```
1  Function ProcessResolvedTriplet($t$, branches, placed_nodes):
       Input: $t = uv|w$ — a resolved triplet induced by $T$
       Input: branches — the list of current branches
       Input: placed_nodes — the list of already placed nodes
       Output: The updated branches and placed_nodes
2      if $u, v$ not in placed_nodes then
3          branches.append($\{u, v\}$)
4      else if $u$ not in placed_nodes then
5          $B_j$ = branch containing $v$
6          $B_j$.add($u$)
7      else if $v$ not in placed_nodes then
8          $B_i$ = branch containing $u$
9          $B_i$.add($v$)
10     else
11         $B_i$ = branch containing $u$
12         $B_j$ = branch containing $v$
13         branches.remove($B_i, B_j$)
14         branches.append($B_i \cup B_j$)
15     placed_nodes.add($u, v$)
16     return branches, placed_nodes
```

Algorithm 4 processes resolved triplets by ensuring that their labels are put in the right branches depending on their current placement in the branches. It is a direct implementation of Lemma 3.7 and runs in constant time.

Algorithm 5: Process fanned triplets

```
1  Function ProcessFannedTriplet(t,branches,placed_nodes,fanned_triplets,C):
       Input: $t = u|v|w$ — the fanned triplet induced by $T$
       Input: branches — the list of current branches
       Input: placed_nodes — the list of already placed nodes
       Input: fanned_triplets — the list of fanned triplets
       Input: $C$ — the set of possible children of the root
       Output: The updated branches and placed_nodes
2      $n_p = |\{u, v, w\} \cap$ placed_nodes$|$          // The number of placed nodes of $t$
3      branches_containing_nodes = $\{B_i \mid B_i \cap t \neq \emptyset\}$
4      $n_b$ = |branches_containing_nodes|
5      if $n_b = 3$ or ($n_p = 3$ and $n_b = 1$) then
6          return branches, placed_nodes
7      else if $n_p > n_b$ then
           // Two nodes are already in the same branch, so all nodes must be in the
              same branch
8          for $B_i \in$ branches_containing_nodes do
9              branches.remove($B_i$)
10         new_branch = ($\bigcup$branches_containing_nodes) $\cup \{u, v, w\}$
11         branches.append(new_branch)
12         placed_nodes.append($u, v, w$)
13         for triplet $\in$ fanned_triplets, triplet $\cap$ new_branch $\neq \emptyset$ do
14             branches, placed_nodes =
                ProcessFannedTriplet(triplet,branches,placed_nodes,fanned_triplets,C)
15     else if $n_p = 2$ then
           // If we can, we put the last node as a child of the root somewhere, or
              create a new branch for it
16         for node $\in \{u, v, w\} \setminus$ placed_nodes do
17             if node $\in C$ then
18                 for $B_i$ in branches do
19                     if $|B_i| \geq 2$ and $D_{\text{node}} \cap B_i = B_i$ then
20                         $B_i$.add(node)
21                         placed_nodes.append(node)
22                         for triplet $\in$ fanned_triplets, triplet $\cap B_i \neq \emptyset$ do
23                             branches,placed_nodes =
                                ProcessFannedTriplet(triplet,branches,placed_nodes,fanned_triplets,C)
24             if node $\notin$ placed_nodes then branches.append({node})
25             placed_nodes.append(node)
26     return branches, placed_nodes
```

Algorithm 5 correctly places the labels of a fanned triplet in their corresponding branches. It is a direct implementation of Lemma 3.7 and runs in $\mathcal{O}(1+p)$, where p is the number of fanned triplets checked after checking the initial one. Note that this is bounded by $\mathcal{O}(|t(T)|)$.

Algorithm 6 divides the labels in their corresponding branches based on the triplets. It is an implementation of Lemma 3.7 and thus its correctness is not proven directly. Note that Algorithm 6 runs in $\mathcal{O}(R + F(|t(T)| + |X|))$, where R is the number of resolved triplets and F is the number of fanned triplets.

Algorithm 6: Divide nodes in branches

```
Function DivideBranches($t'(T), X, C$):
    Input: $t(T)$ — a set of triplets
    Input: $X$ — the labelled nodes of $T$
    Input: $C$ — the set of possible children of the root
    Output: branches — the list of branches from the root
    placed_nodes = [ ]
    branches = [ ]
    fanned_triplets = the subset of fanned triplets
    resolved_triplets = the subset of resolved triplets
    for $t = uv|w \in$ resolved_triplets do
        branches, placed_nodes = ProcessResolvedTriplet($t$,branches,placed_nodes) using
         Algorithm 4
    for $t = u|v|w \in$ fanned_triplets do
        branches, placed_nodes =
         ProcessFannedTriplet($t$,branches,placed_nodes,fanned_triplets,$C$) using
         Algorithm 5
    for $x \in X, x \notin$placed_nodes do
        for $t \in$ fanned_triplets, $x \in t$ do
            branches, placed_nodes =
             ProcessFannedTriplet($t$,branches,placed_nodes,fanned_triplets,$C$) using
             Algorithm 5
        if $x \notin$ placed_nodes then
            forall $B_i$ do
                if $(|B_i| \geq 2$ or $x \notin C)$ and $D_x \setminus B_i = \emptyset$ then
                    $B_i$.add($x$)
                    break
        else
            branches.append($\{x\}$)
        placed_nodes.append($x$)
    return branches
```

A.1.2. General trees

Lemma A.1. Let $T = (V, E, l)$ be a tree on X, and $u, v \in X$ be such that a directed path from u to v exists in T. Given any $w \in X \setminus \{u, v\}$, there still exists a directed path from u to v in $T \setminus w$.

Moreover, if a path from u to v exists in $T \setminus w$, a path from u to v also exists in T.

Proof.
Let $u, v, w \in X$ be such that a directed path from u to v exists. Suppose, without loss of generality, $(l(u) = n_0, n_1, n_2, \ldots, n_k, l(v) = n_{k+1})$ is this path with $n_0, \ldots, n_{k+1} \in V$. Either (i) $l(w) \in \{n_1, \ldots, n_k\}$ or (ii) $l(w) \notin \{n_1, \ldots, n_k\}$.

(i) Suppose $l(w) = n_i$. Two possibilities arise: (a) n_i has out-degree one, or (b) n_i has out-degree two or larger.

  (a) The edge $n_i n_{i+1}$ will be contracted according to Definition 2.1. By that definition $n_{i-1} n_{i+1}$ will be added in $T \setminus w$ as an edge. Thus, the path $(n_0, n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k, n_{k+1})$ exists in $T \setminus w$.

  Or the if n_{i-1} has out-degree one $n_{i-1} n_i$ is contracted in which case the path $(n_0, n_1, \ldots, n_{i-2}, n_i, \ldots, n_k, n_{k+1})$ exists in $T \setminus w$.

  (b) If n_{i-1} has out-degree one, then $n_{i-1} n_i$ will be contracted according to Definition 2.1. A new edge will be included in $T \setminus w$, namely $n_{i-2} n_i$. So the path $(n_0, n_1, \ldots, n_{i-2}, n_i, \ldots, n_k, n_{k+1})$ exists in $T \setminus w$. Note that if $i = 1$, then in $T \setminus w$ $l(u) = n_1$ and thus the path would be $(n_1, n_2, \ldots, n_k, n_{k+1})$.

  If n_{i-1} has out-degree two or larger, the path remains the same as no contraction takes place.

(ii) If w is not a descendant of u it is clear that $T \setminus w$ will still contain the path. The same holds if w is a descendant of v. Also, if w is not a direct descendant of any vertex of the path, the path will remain in $T \setminus w$. Namely, removing w can only cause a contraction of an edge containing w, and therefore cannot include any vertex in the path.

  Therefore, assume w is a child of a vertex of the path except v, say n_i, but not in $\{n_1, \ldots, n_{k+1}\}$ itself. In two scenarios, a contraction can take place.

  (a) If w is a leaf of T, n_i has out-degree two in T, and either n_i and/or n_{i+1} is unlabelled then edge $n_i n_{i+1}$ will be contracted. Thus the path $(n_0, n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k, n_{k+1})$ exists. If $n_i = n_0$, then the path becomes $(n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k, n_{k+1})$ and $l(u) = n_1$.

  (b) If w is not a leaf of T, let $c \in X$ be the child of w in T. We may assume w has out-degree one, as otherwise no contractions would take place. The edge wc is contracted, in which case the path remains the same.

In all cases, we have shown that a path from u to v still exists in $T \setminus w$.

For the second statement, suppose a path from u to v exists in $T \setminus w$. Clearly, a path from u to v still exists in T as removing w from T only contracts edges or removes edges that point to a leaf. Therefore, a path still exists in T from u to v. □

Corollary A.2. If there is a path from $x \in V$ to $u \in X$ and $v \in X$. Then, after removing a label different from u and v, there is still a path from $x' \in V$ to both labels.

Proof.
If u and v are in different branches of the subtree rooted at x, then removing any other label will not remove x as a node. Thus, the paths from x to both labels remain by the reasoning of Lemma A.1.

If u and v are in the same branch of the subtree rooted at x, then there is some descendant of x, say x', such that u and v are in different branches of the subtree rooted at x'. Then, by the same reasoning as above, the paths from x' to u and v remain.

If no such descendant exists, then, without loss of generality, v is a descendant of u. Thus, after removing any other label will v will still be a descendant of u. Therefore, there is still a path from an $x' \in V$ (namely any of u's ancestors or u itself) to both u and v. □

Algorithm 7: Obtain descendants and separations from triplets

```
Function GetDescendantsAndSeperation($t'(T), X$):
    Input: $t'(T)$ — a set of triplets
    Input: $X$ — the labelled nodes of T
    Output: $D,\ S$ — two dictionaries showing the descendants of every label and which nodes are in
            different branches
    $D, S = \{x : \{\}$ for $x \in X\}$
    forall $t \in t'(T)$ do
        if $t = uv|w$ then
            $S[u]$.add$(v, w)$
            $S[v]$.add$(u, w)$
            $S[w]$.add$(u, v)$
        else if $t = u|v|w$ then
            $S[u]$.add$(v, w)$
            $S[v]$.add$(u, w)$
            $S[w]$.add$(u, v)$
        else if $t = u/v|w$ then
            $D[v]$.add$(u)$
            $S[u]$.add$(w)$
            $S[v]$.add$(w)$
            $S[w]$.add$(u, v)$
        else if $t = u/v/w$ then
            $D[w]$.add$(v, u)$
            $D[v]$.add$(u)$
        else if $t = u/v \setminus w$ then
            $D[v]$.add$(u, w)$
            $S[u]$.add$(w)$
            $S[w]$.add$(u)$
    Obtain the transitive closure of $D$ using depth-first search
    return $D, S$
```

Algorithm 7 creates the D and S hashmaps based on the triplets. The hashmap D contains all the descendants for a given label based on the triplets, and S contains all labels that at some point must be in a different branch than a given label. It runs in $\mathcal{O}(|t(T)| + |X|)$.

Algorithm 8: Get possible roots

```
1  Function PossibleRoots($t'(T), X, D, S$):
       Input: $t'(T)$ — a set of triplets
       Input: $X$ — the labelled nodes of T
       Input: $D$ — the dictionary showing each node's descendants obtained using Algorithm 7
       Input: $S$ — the dictionary showing separation between nodes obtained using Algorithm 7
       Output: $R$ — a list of labels that can be the root of the tree
2      $R = [\,]$
3      for $x \in X$ do
4          if $S[x] \cap X = \emptyset$ and $x \notin D[u]$ for all $u \in X$ then
5              $R$.append($x$)
6      forall $t \in t'(T)$ do
7          if $|R| = 0$ then
8              return $R$
9          if $t = u / v \setminus w$ then
10             $R$.remove($u, w$)
11             if $\{u, w\} \subseteq D[x]$ for some $x \in X \setminus \{v\}$ then
12                 $R$.remove($v$)
13             else if $\exists \tilde{t} = x, y|z \in t'(T)$ such that $x \in D[u] \cup u$ and $y \in D[w] \cup w$ then
14                 $R$.remove($v$)
15     return $R$
```

Algorithm 8 finds the set of possible roots based on Lemma 3.16. It runs in $\mathcal{O}(|X|^2 + |t(T)|^2)$.

Algorithm 9: Process Fanned Triplet

```
1  Function ProcessFannedTriplet($t$, branches, $t'(T), D$):
       Input: $t = u|v|w$ — a fanned triplet in $t'(T)$
       Input: branches — the list of current branches
       Input: $t'(T)$ — a set of triplets
       Input: $D$ — the dictionary showing each node's descendants obtained using Algorithm 7
       Output: branches, $D$ — the list of branches from the root and the updated descendant dictionary
2      branches_containing_nodes = $\{B_i \in$ branches $\mid B_i \cap \{u, v, w\} \neq \emptyset\}$
3      if |branches_containing_nodes| = 2 then
4          $B_1, B_2$ = branches_containing_nodes
5          branches.remove($B_1, B_2$)
6          branches.append($B_1 \cup B_2$)
7          for $\tilde{t} = x|y|z \in t'(T)$ such that $\{u, v, w\} \cap \{x, y, z\} \neq \emptyset$ do
8              branches, $D$ = ProcessFannedTriplet($\tilde{t}$, branches, $t'(T), D$)
9      for $x \in X$ such that $|D[x] \cap \{u, v, w\}| = 2$ do
10         $D[x]$.add($u, v, w$)
11     return branches, $D$
```

Algorithm 9 correctly places the labels of a fanned triplet in their corresponding branches. It is a direct implementation of Lemma 3.7. The runtime of Algorithm 9 is bounded by $\mathcal{O}(|X|(F+1))$.

Algorithm 10: Divide nodes in branches

```
Function DivideBranches($t'(T), X, D, \rho$):
    Input: $t'(T)$ — a set of triplets
    Input: $X$ — the labelled nodes of $T$
    Input: $D$ — the dictionary showing each node's descendants obtained using Algorithm 7
    Input: $\rho \in X$ — the labelled root of $T$ if it exists
    Output: branches — the list of branches from the root
    placed_nodes = []
    branches = []
    fanned_triplets = the subset of fanned triplets
    forall $x \in X \setminus \{\rho\}$ do
        if $(D[x] \cup \{x\}) \cap$ placed_nodes $\neq \emptyset$ then
            branches_containing_nodes = $\{B_i \in$ branches $\mid B_i \cap (D[x] \cup \{x\}) \neq \emptyset\}$
            for $B_i \in$ branches_containing_nodes do
                branches.remove($B_i$)
            new_branch = ($\bigcup$branches_containing_nodes) $\cup D[x] \cup \{x\}$
            branches.append(new_branch)
        else branches.append($D[x] \cup \{x\}$)
        placed_nodes.append($D[x] \cup \{x\}$)
    for $t = uv|w \in t'(T)$ do
        branches_containing_nodes = $\{B_i \mid B_i \cap \{u, v\} \neq \emptyset\}$
        for $B_i \in$ branches_containing_nodes do
            branches.remove($B_i$)
        new_branch = $\bigcup$branches_containg_nodes
        branches.append(new_branch)
    for $t = u|v|w \in$ fanned_triplets do
        branches, $D$ = ProcessFannedTriplet($t$, branches, $t'(T), D$) using Algorithm 9
    Obtain the transitive closure of $D$ using depth-first search
    return branches
```

Algorithm 10 places all labels in their corresponding branches based on Lemmas 3.7 and 3.17. It runs in at most $\mathcal{O}(|X| + (|t(T)| - F) + F(|X|(F+1))) = \mathcal{O}(F^2|X| + F|X| - F + |t(T)| + |X|)$, where F is the number of fanned triplets.

A.2. Networks

Algorithm 11: Create SN sets (based on Figure 4 of Jansson and Sung (2006))

```
1  Function GetSNSets($t(N), X$):
       Input: $t(N)$ — the set of triplets
       Input: $X$ — the labelled nodes of $N$
       Output: $SN$ — the list of non-trivial $SN$ sets of $N$
2      $SN$ = a list of $|X|$ sets, each containing a different label
3      forall $x, y \in X$ do
4          $S = \{x\}$ and $Z = \{y\}$
5          while $Z \neq \emptyset$ do
6              Let $z$ be any element in $Z$
7              forall $a \in S$ do
8                  if there is some $c \in X \setminus (S \cup Z)$ such that a triplet exists in $t(N)$ on $a, c$ and $z$ where $c$ is a
                     descendant of LCA$(a, z)$ in that triplet then $Z = Z \cup \{c\}$
9              $S = S \cup \{z\}$ and $Z = Z \setminus \{z\}$
10         if $S \neq X$ then append $S$ to $SN$
11     return $SN$
```

Algorithm 12: Process Fanned Triplet for a network

```
1  Function ProcessFannedTriplet($t$, branches, $t(N)$, MaxSN):
       Input: $t = u|v|w$ — a fanned triplet in $t(N)$
       Input: branches — the list of current branches
       Input: $t(N)$ — the set of triplets
       Input: MaxSN — the maximal $SN$ sets of $N$
       Output: branches — the updated list of branches from the root
2      if $\exists S_1, S_2, S_3 \in$ MaxSN such that $S_i \cap \{u, v, w\} \neq \emptyset \; \forall i \in \{1, 2, 3\}$ then
3          return branches
4      branches_containing_nodes = $\{B_i \mid B_i \cap \{u, v, w\} \neq \emptyset\}$
5      if |branches_containing_nodes| = 2 then
6          $B_1, B_2$ = branches_containing_nodes
7          branches.remove($B_1, B_2$)
8          branches.append($B_1 \cup B_2$)
9          for $\tilde{t} = x|y|z \in t'(T), \{u, v, w\} \cap \{x, y, z\} \neq \emptyset$ do
10             branches, $D$ = ProcessFannedTriplet($\tilde{t}$, branches, $t(N)$, MaxSN)
11     return branches
```

Lemma A.3. Let t be a fanned triplet on u, v, and w, B_i the current branches from the root of a network, $t(N)$ the triplet set, and $SN_{max}$the maximal SN sets of N. Then Algorithm 12, with these inputs, correctly updates the branches in $\mathcal{O}(|X||t(T)|)$ time.

Proof.
If the labels of a fanned triplet are all in the same branch and the root is not the source of a cycle, they must also be in the same maximal SN set. If the labels are in the same branch (note that a cycle is considered to be one branch) but the root is the source of a cycle, then the labels can be part of different SN sets. For example, consider u to be the sink of the cycle, and v and w to be both children of a cycle vertex but not cycle vertices. Then each label is in a different SN set. However, they will be placed in the same branch by Algorithm 10. Likewise, if the labels of a fanned triplet are spread in two distinct branches, then two of these labels must be part of a cycle whose source is the root, and the other is a descendant of the root but not part of the cycle. So the root has out-degree three or higher. However, the labels that are part of the sink will both be in their own maximal SN sets, so each label still has a distinct SN set, and thus the branches will not be merged. Lastly, if all labels are in their own branches, then they are also contained in distinct maximal SN sets, so no branches need to be merged.

For the time complexity, Line 2 loops over all maximal SN sets and thus runs in $\mathcal{O}(|SN_{max}|)$ time. Likewise, Line 4 loops over all current branches and thus runs in $\mathcal{O}(|\texttt{branches}|)$ time. Lastly, the algorithm calls itself at most $|t(T)|$ times. In total Algorithm 12 runs in $\mathcal{O}(|t(T)|(|SN_{max}| + |\texttt{branches}|)$ time which is capped by $\mathcal{O}(|X||t(T)|)$. □

Algorithm 13: FindSinkOfCycle

```
1  Function FindSinkOfCycle($\rho, t(N), X,$ MaxSN$, D$):
       Input: $\rho$ — a source label
       Input: $t(N)$ — the set of triplets of $N$
       Input: $X$ — the labels of $N$
       Input: MaxSN — the maximal $SN$ sets of $N$
       Input: $D$ — the dictionary showing each node's descendants obtained using Algorithm 7
       Output: $SD$ — a list of sets of sinks and their descendants
2      $SD = \emptyset$
3      if $\rho \in X$ then
4          foreach triplet $t \in t(N)$ such that $t = n_1/\rho \setminus n_2$ do
5              common_descendants $= D[n_1] \cap D[n_2]$
6              if $n_1 \in D[n_2]$ then add $n_1$ to common_descendants
7              else if $n_2 \in D[n_1]$ then add $n_2$ to common_descendants
8              if common_descendants $\neq \emptyset$ then
9                  Add common_descendants to $SD$
10     $R = \emptyset$
11     foreach $\{n_1, n_2, n_3\} \subseteq X$ do
12         if $|N|_{n_1,n_2,n_3}| = 2$ then
13             if one triplet is $n_1|n_2\backslash n_3$, and the other not of the form $1/2\backslash 3$ or $1\backslash 2\backslash 3$ then
14                 $SD = SD \cup \{n_3\}$
15             else if one triplet is $n_1|n_2\backslash n_3$ and the other is $n_2 \setminus n_3 \setminus n_1$ then
16                 $SD = SD \cup \{n_1\}$
17             else if one triplet is $n_1|n_2\backslash n_3$ and the other is $n_1 \setminus n_2 \setminus n_3$ then
18                 $SD = SD \cup \{n_2, n_3\}$
19             else if one triplet is $n_1|n_2\backslash n_3$ and the other is $n_1/n_2 \setminus n_3$ then
20                 $SD = SD \cup \{n_1\}$
21             else if both triplets of the form $1|2,3$ then
22                 $R = R \cup N|_{n_1,n_2,n_3}$
23     foreach $t = u|v,w \in R$ do
24         if $\{u,v,w\} \cap SD = \emptyset$ then
25             foreach $t' = x|y,z \in R$ sharing exactly two labels with $\{u,v,w\}$ do
26                 $d = \{x,y,z\} \setminus \{u,v,w\}$
27                 if $d \not\subseteq SD$ then
28                     if exactly one of $|N|_{a,b,c} \cap R|$, $|N|_{a,b,d} \cap R|$, $|N|_{a,c,d} \cap R|$, $|N|_{b,c,d} \cap R|$ equals 0 then
29                         Add the label not used for that empty set to $SD$
30     $SD = \{S \in$ MaxSN $| S \subseteq SD\}$
31     return $SD$
```

Lemma A.4. Let the root of N be the source of a cycle and take $\rho \in X$ to be this source if it is labelled, $t(N)$ the triplet set of N, X the labels of N, MaxSN the maximal SN sets of N, and D_x the descendants of any label $x \in X$ as obtained by Algorithm 7. Given this input, Algorithm 13 finds any highest sink and its descendants with no cut-arcs above it in $\mathcal{O}(|X|^3 + |t(N)|^2)$ time.

Proof.
If the source of the cycle is labelled and there is at least one labelled cycle vertex, then there is a triplet of the form $u/\rho \setminus s$ where u is this labelled cycle vertex and s is the sink of the cycle. Then the common descendants

of u and s are the descendants of the sink, and s is added in Line 6. Lastly, these labels are then added to the list of sinks in Line 9.

So suppose the source is not labelled, or no cycle vertices (other than the sink) are labelled, and let $l \geq 3$ be the size of the cycle. If $l = 3$, then the other cycle vertex, u, must be labelled, so we may assume the source is not labelled. In this case, since $|X| \geq 3$, either (i) the sink or the cycle vertex has another child, or ii) the source has out-degree three or higher.

(i) The two triplets will be $v/s/u$ and $v/s|u$ or $s/u \setminus v$ and $s|u \setminus v$. The first case would be handled by Line 17 such that v and s will be added to the sink and its descendants. Line 19 handles the second case by adding s to the sink and its descendants.

(ii) Now the two triplets would be of the form $s|u|v$ and $s/u|v$. Line 13 would ensure s is added to the sink and its descendants.

So we see for $l = 3$ the sink is found properly.

Now, if $l = 4$, a non-sink, non-source cycle vertex, u, must be labelled. The cases as described above are properly handled by the same reasoning. However, since $|X| \geq 3$ three more cases remain where u does not have any non-cycle children: (i) the other cycle vertex is an ancestor of u, (ii) the other cycle vertex is a descendant of u, or (iii) the other cycle vertex is a sibling of u.

(i) If this other cycle vertex is labelled with v we have $s/u/v$ and $s|v \setminus u$ as triplets. Then Line 15 would add s as a sink. If this cycle vertex is not labelled, it must have another child. So let v be this child (or one of its labelled descendants). Then $s/u|v$ and $s|u,v$ would be the triplets and Line 13 would identify s as the sink.

(ii) If the cycle vertex is labelled with v we have $s/v/u$ and $s|u \setminus v$ as triplets. Then Line 15 would add s as a sink. If this cycle vertex is not labelled, then u must have another child. So let v be this child (or one of its labelled descendants). Then $s|u \setminus v$ and $s/u \setminus v$ would be the two triplets. So Line 19 would correctly add s as the sink.

(iii) Assuming the cycle vertex is labelled with v the two triplets would be $s/u|v$ and $s/v|u$. If the vertex is not labelled, the triplets would be $s/u|v$ and $s,v|u$. In both cases, Line 13 would again add s as the sink.

So also for $l = 4$ the sink is correctly identified.

Lastly, if $l \geq 5$, the only remaining case would be if no cycle vertices were labelled. In which case, for any three nodes on which at least two triplets exist, the triplets would both be of the form $s,u|v$ and $s,v|u$. These triplets are added to a list in Line 21 and handled separately afterwards.

Suppose u and v are in distinct cycle branches on different paths from the source to the sink, s. Then their triplets would be $u|v,s$ and $v|u,s$. Given that $l \geq 5$, there is another cycle branch containing a label d. Then d,u and s form either $u|d,s$ and $d|u,s$, $u,d|,s$ and $d|u,s$, or $u,d|s$ and $u|d,s$ as triplets. And d,v and s form either $v|d,s$ and $d|v,s$, $v,d|,s$ and $d|v,s$, or $v,d|s$ and $v|d,s$ as triplets. So, $N|_{u,d,s} \subset R$. Meanwhile, $N|_{u,v,d}$ is size one and contains either $u,v|d$, $u,d|v$, or $v,d|u$. Indeed, only $|N|_{u,v,d}| = 1$ holds and thus s is added as a sink by Line 29.

If, however, all cycle branches are on the same path from the source to the sink, the triplets on the sink s, and two labels in distinct cycle branches would be, without loss of generality, of the form $s,u|v$ and $s|u,v$. Given that $l \geq 5$, there is another cycle branch containing d. The triplet on u, v, and d would, without loss of generality, be $u,v|d$. And the triplets on u,d, and s would be $s|u,d$ and $s,u|d$, while the triplets on v,s, and d would be $s|v,d$ and $s,v|d$. Therefore, again, only $|N|_{u,v,d}| = 1$ holds and thus s is added as a sink by Line 29.

We have thus shown that any sink and its descendants can be detected. Also note that no labels outside of a sink and its descendants can be added, as no three such nodes can form two triplets. By Corollary 4.4, the highest sinks and descendants of cycles with no cut-arcs above them are maximal SN sets. Therefore, filtering the sinks and descendants for maximal SN sets in Line 30 gives us the highest sinks and descendants with no cut-arcs above it.

Looking at the computation time, we see that we loop through $\mathcal{O}(|t(N)|)$ triplets at Line 4. And at most $\mathcal{O}(|X|^3)$ times at line Line 11. Then, in the loop at Line 23 we loop over R twice, giving us $\mathcal{O}(|R|^2)$ theoretical running time, which is bounded by $\mathcal{O}(|t(N)|^2)$. Lastly, at Line 30 we loop over all the maximal SN sets which is at most

$\mathcal{O}(|X|)$ So in total the theoretical running time is $\mathcal{O}(|X|^3 + |t(N)|^2)$. Note that the time taken to find the triplets is not taken into account. This is because at the start of Algorithm 3 we can loop over all triplets once and make hashmaps that map any three or two labels to all the triplets that contain those labels. Then, finding the triplets can be done in constant time. □

Algorithm 14: RemoveOuterSinks

```
Function RemoveOuterSinks($SD, \rho, X, t(N)$):
    Input: $SD$ — the sets of sinks and their descendants
    Input: $\rho$ — a source label
    Input: $X$ — the labels of $N$
    Input: $t(N)$ — the set of triplets of $N$
    Output: $SD$ — the singular sink of the cycle whose source is the root of $N$
    foreach distinct pair $S_1, S_2 \in SD$ do
        Pick any $s_1 \in S_1$, $s_2 \in S_2$
        foreach $o \in X \setminus (S_1 \cup S_2 \cup \{\rho\})$ do
            $T_f$ = all fanned triplets and triplets of the form $1|2 \setminus 3$ on $s_1, s_2$, and $o$ in $t(N)$
            $T_r$ = all resolved triplets on $s_1, s_2$, and $o$ in $t(N)$
            if $|T_f| \geq 1$ and $|T_r| \geq 1$ then
                foreach $t_r \in T_r$ do
                    if $t_r = s_1|s_2, o$ and no $t'_r = s_2|s_1, o \in T_r$ then
                        $SD = SD \setminus \{S_2\}$
                        break
                    else if $t_r = s_2|s_1, o$ and no $t'_r = s_1|s_2, o \in T_r$ then
                        $SD = SD \setminus \{S_1\}$
                        break
    return $SD$
```

Lemma A.5. Let N be a network on X, s be the root of N and the source of exactly one cycle, SD be a list of size two or more, containing maximal SN sets that correspond to highest sinks and descendants with no cut-arcs above it obtained using Algorithm 13, and $t(N)$ be the set of triplets of N. Then given this input Algorithm 14 returns the sink and descendants of the cycle that has s as its source in $\mathcal{O}(|X|^3)$ time.

Proof.
Given that $|SD| \geq 2$, a cycle vertex is the source of another cycle. Thus, the sink and its descendants of that lower cycle should be removed from SD.

The for loop at Line 2 finds the sinks and descendants of cycles whose source is a cycle vertex of another cycle. Namely, take $S_1, S_2 \in SD$ such that S_1 is the sink of the cycle, C_1, whose source is the root, and S_2 is the sink of the cycle, C_2, whose source is a cycle vertex of C_1. Take $o \in X \setminus (S_1 \cup S_2 \cup \{s\})$, $s_1 \in S_1$, and $s_2 \in S_2$ arbitrarily. Assuming there is a triplet on s_1, s_2, and o of the form $u|v \setminus w$ or $u|v|w$ as well as a resolved triplet, then either o is a cycle vertex of C_1 on a different path from the source to the sink than C_2, o is a cycle vertex of C_2, or is in one of the cycle branches of C_2.

If o is a cycle vertex of C_1 on a different path from the source to the sink than C_2, then the resolved triplet is of the form $o|s_1, s_2$ and thus this label is not used to define which sink to remove.

If, however, o is a cycle vertex of C_2 or is in one of its cycle branches, any resolved triplet will be of the form $s_1|o, s_2$. And thus S_2 is removed from the list of sinks and descendants. We know such o exists as C_2 is a cycle and must therefore have a labelled cycle vertex or have cycle branches.

Thus Algorithm 14 returns only the sink of the cycle that has the root as its source.

Concerning the theoretical running time, we loop over each pair of elements in SD, after which we loop over all other labels. So the algorithm runs in $\mathcal{O}(|SD|^2|X|)$, where SD is again bounded by $|X|$. Therefore, the theoretical runtime of Algorithm 14 is $\mathcal{O}(|X|^3)$. Note that, again, looking up the triplets for any three labels can be done in constant time once a hashmap has been created that maps the labels to their triplets. □

Algorithm 15: ResolveCycle

```
1  Function ResolveCycle($SD$, $X$, $\rho$, $D$, $t(N)$):
       Input: $SD$ — the sink and its descendants
       Input: $X$ — set of labels
       Input: $\rho$ — optional source label
       Input: $D$ — descendant map (from Algorithm 7)
       Input: $t(N)$ — the set of all triplets of $N$
       Output: $CB$, $ICV$ — list of cycle branches, and internal cycle vertices
2      $ICV = \{\ell \in X \setminus \{\rho\} \mid D[\ell] \cap SD = SD \setminus \{\ell\}\}$    // cycle labels
3      $CB$ = a list where each label $x \in X \setminus (SD \cup ICV \cup \{\rho\})$ is its own branch    // cycle branches
4      $IB = [\,]$    // internal-to-branch pairs
5      foreach distinct pair $n_1, n_2 \in X \setminus SD$ do
6          foreach $z \in SD$ do
7              if $|N|_{n_1,n_2,z}| = 1$ or $\exists t_1, t_2 \in N|_{n_1,n_2,z}$ such that $t_1$ is of the form $1|2\backslash 3$ and $t_2$ of the form $1/2\backslash 3$
                 or $\exists t \in N|_{n_1,n_2,z}$ of the form $1|2|3$ then
8                  $B = \{b \in CB \mid (n_1 \in b \setminus ICV) \vee (n_2 \in b \setminus ICV)\}$
9                  if $|B| \geq 1$ then
10                     Remove all branches in $B$ from $CB$
11                     $NB = \left(\bigcup B\right) \cup \left(\{n_1, n_2\} \setminus (\{\rho\} \cup ICV)\right)$
12                     Append $NB$ to $CB$
13                 else append $\{n_1, n_2\} \setminus (\{\rho\} \cup ICV)$ to $CB$
14                 if $\exists z/n_1 \setminus n_2 \in N|_{n_1,n_2,z}$ then
15                     Append $(n_1, n_2)$ to $IB$
16                 else if $\exists z/n_2 \setminus n_1 \in N|_{n_1,n_2,z}$ then
17                     Append $(n_2, n_1)$ to $IB$
18     foreach pair $(c, b) \in IB$ do
19         Let $B_i$ be the (unique) branch in $CB$ containing $b$
20         $B_i = B_i \cup \{c\}$
21     foreach $x \in ICV \setminus (\bigcup CB)$ do
22         Append $\{x\}$ to $CB$
23     return $CB$, $ICV$
```

Lemma A.6. Let N be a network whose root, ρ, has out-degree two and is the source of exactly one cycle, SD be the sink and its descendants of the cycle, D_x the descendants of any label $x \in X$ as obtained by Algorithm 7, and $t(N)$ the set of triplets. Given this input, Algorithm 15 returns a list of the labelled cycle vertices and a list of all cycle branches (including their corresponding labelled cycle vertex) in $\mathcal{O}(|X|^3)$ time.

Proof.
In Line 2, all labelled cycle vertices are stored. Namely, each such vertex must have the entire sink and its descendants as part of its descendants. Then, in Line 3 all labels not in SD, ICV or $\{\rho\}$ are placed in their own branches. Therefore, branches only need to be merged when two labels are supposed to be in the same cycle branch.

Let $z \in SD$ and $n_1, n_2 \in X \setminus SD$.

If $|N|_{n_1,n_2,z}| = 1$ clearly n_1 and n_2 must be in the same cycle branch or the triplet is, without loss of generality, of the form $z|n_1 \setminus n_2$, where n_1 is a cycle vertex and n_2 is not a cycle vertex and is in a lower cycle branch. In the first case, they are placed in the same branch in Line 12 or Line 13 if they were not yet placed. In the second case, since $n_1 \in ICV$, only n_2 will be placed in a branch if it was not already.

Likewise, if $z|n_1 \setminus n_2, z/n_1 \setminus n_2 \in N|_{n_1,n_2,z}$, then n_1 is a cycle vertex and n_2 is in the cycle branch of n_1. Again, either Line 12 or Line 13 places only n_2 in a branch. Moreover, (n_1, n_2) is added to IB in Line 15 to later add n_1 to the same branch, which is done in Line 20.

If $n_1|n_2|z \in N|_{n_1,n_2,z}$, then n_1 and n_2 are also in the same cycle branch. Indeed, either Line 12 or Line 13 places them in the same branch.

Lastly, any labelled cycle vertices that have out-degree one are added as their own branches in Line 22.

So Algorithm 15 returns the cycle branches and the internal cycle vertices of a cycle given the input.

Looking at the time complexity, the algorithm loops over a subset of each combination of length three of the labels. So this loop is at most $\mathcal{O}(|X|^3)$. The algorithm also loops over the pairs of internal cycle vertices and their corresponding branches, and afterwards over the remaining labelled cycle vertices, which are both bounded by $\mathcal{O}(|X|)$. Therefore, the theoretical running time of the algorithm is $\mathcal{O}(|X|^3)$. By using a hashmap that maps any three labels to the triplets on these labels, looking up the triplets can be done in constant time. □

Algorithm 16: FindCycleOrder

```
1  Function FindCycleOrder(branches, $SB$, $CV$, $\rho$, $t(N)$):
       Input: branches — current list of cycle branches except the sink branch obtained using
              Algorithm 15
       Input: $SB$ — the sink branch
       Input: $ICV$ — the labelled cycle vertices
       Input: $\rho$ — the source label
       Input: $t(N)$ — the triplet set of $N$
       Output: $L$, $R$ — two ordered lists of branches (left and right)
2      $L = [\,]$, $R = [\,]$                                    // left/right branch ordering
3      foreach distinct $b_1, b_2 \in$ branches do
4          foreach $n_1 \in b_1$, $n_2 \in b_2$ do
5              foreach $z \in SB \cup \{\rho | \text{ if } \rho \in X\}$ do
6                  foreach $t \in N|_{n_1,n_2,z}$ do
7                      if $t$ of the form $1/2/3$ then
8                          place_together ($b_1, b_2$)
9                      else if $t$ of the form $1|2\backslash 3$ then
10                         if $t = z|n_1 \setminus n_2$ or $t = z|n_2 \setminus n_1$ then
11                             place_together ($b_1, b_2$)
12                         else if $n_1, n_2 \in CV$ then
13                             place_apart ($b_1, b_2$)
14                         else
15                             Let $t' \in N|_{n_1,n_2,z} \setminus \{t\}$
16                             if $t' = z, n_1|n_2$ or $t' = z, n_2|n_1$ then
17                                 place_apart ($b_1, b_2$)
18                                 break
19                             place_together ($b_1, b_2$)
20                     else if $t$ of the form $1|2,3$ then
21                         if $t = z|n_1, n_2$ then
22                             place_together ($b_1, b_2$)
23                         else
24                             Let $t' \in N|_{n_1,n_2,z} \setminus \{t\}$
25                             if $t' = n_1|n_2, z$ or $t' = n_2|n_2, z$ then
26                                 place_apart ($b_1, b_2$)
27                     else if $t$ is of the form $1/2\backslash 3$ then
28                         place_apart ($b_1, b_2$)
29     if |branches| = 1 then $L$ =branches
30     return $L$, $R$
```

Algorithm 17: Helper functions for Algorithm 16

```
1 Function place_together($b_1$, $b_2$):
2     Place $b_1$ and $b_2$ such that they are either both in $L$ or in $R$.
3     If neither is placed yet and $L = [] = R$, then place them both in either $L$ or $R$.
4 Function place_apart($b_1$, $b_2$):
5     Place $b_1$ and $b_2$ such that they are not both in $L$ or $R$.
6     If neither is placed yet and $L = [] = R$, then place one in $L$ and the other in $R$ arbitrarily.
```

Lemma A.7. Let N be a network whose root, ρ, has out-degree two and is the source of exactly one cycle, B be the list of cycle branches as obtained by Algorithm 15, SB be the sink and its descendants, ICV the labelled cycle vertices, $t(N)$ the set of triplets. Given this input, Algorithm 16 properly separates the branches in two sets, each corresponding to one of the paths from the source to the sink in $\mathcal{O}(|X|^4)$ time.

Proof.
Given any two labels from two distinct branches, and a label from the sink and its descendants, z, it can be deduced whether or not these branches are on the same path from the source to the sink. For the remainder of this proof, we will refer to such a path as a "side" of the cycle.

Suppose $b_1, b_2 \in B$ are on the same side and $n_1 \in b_1, n_2 \in b_2$, and that b_2 is part of the descendants of the cycle vertex corresponding to b_1 (i.e. b_2 is "below" b_1). Then the triplets they can have are: $z|n_1 \setminus n_2$ and $z/n_2/n_1$, $z|n_1 \setminus n_2$, $z|n_1, n_2$ and $z/n_2|n_1$, or $z|n_1, n_2$ and $z, n_2|n_1$. While if b_1 and b_2 are on different sides, then the triplets on n_1, n_2, and z can be: $z/n_1|n_2$ and $z/n_2|n_1$, $z/n_1|n_2$ and $z, n_2|n_1$, $z, n_1|n_2$ and $z/n_2|n_1$, or $z, n_1|n_2$ and $z, n_2|n_1$. Note that for both scenarios the different combinations correspond to $n_1, n_2 \in ICV$, $n_1 \in ICV$ and $n_2 \notin ICV$, $n_2 \in ICV$ and $n_1 \notin ICV$, and $n_1, n_2 \notin ICV$, respectively. Lastly, if a triplet on n_1, n_2, and ρ exists of the form $n_1/\rho \setminus n_2$, then clearly b_1 and b_2 are on different sides.

It can be easily checked that for any of the above cases, the branches are placed correctly in L or R.

For the time complexity, the algorithm loops over a subset of each combination of length three of the labels. This this has time complexity $\mathcal{O}(|X|^3)$. Moreover, to correctly place the branches in L and/or R, it has to loop over at least one of the lists every time, which can be done in $\mathcal{O}(|X|)$ time. So in total, the algorithm runs in $\mathcal{O}(|X|^4)$ time. Again, by using a hashmap that maps any three labels to the triplets on these labels, looking up the triplets can be done in constant time. □

Algorithm 18: FilterTriplets

```
1  Function FilterTriplets($t$, branches, $SD$, $ICV$, $t(N)$):
       Input: $t$ — a triplet on $u, v,$ and $w$
       Input: branches — the branches of the cycle on one side
       Input: $SD$ — the sink and its descendants
       Input: $ICV$ — all labelled cycle vertices
       Input: $t(N)$ — the triplet set of $N$
       Output: True or False whether or not the triplet should be included
2      $B$ = set of all $b \in$ branches $\cup \{SD\}$ such that $b$ contains at least one label of $\{u, v, w\}$
3      if $|B| = 1$ then
4          return True
5      if $t$ is of the form $1/2 \setminus 3$ or of the form $1 \setminus 2 \setminus 3$ then
6          return True
7      if $t$ contains no labels from $SD$ then
8          return True
9      $b_s$ = the branch of $t$ containing a label from $SD$
10     if $b_s$ contains two labels from $SD$ then
11         if the remaining label of $t$ is not in $ICV$ then
12             return True
13         else return False
14     else if $b_s$ contains one label from $SD$ then
15         if $t$ is of the form $1|2,3$ then
16             if $|N|_{u,v,w}| \geq 2$ and $\exists t' \in N|_{u,v,w}$ such that $t'$ is of the form $1|2|3$ then
17                 return False
18         $B$ = set of all $b \in$ branches such that $b$ contains at least one label of $\{u, v, w\} \setminus b_s$
19         if $|B| = 1$ then
20             if no label in $\{u, v, w\} \setminus b_s$ is in $ICV$ then
21                 return True
22             else return False
23         else return False
```

Lemma A.8. Let N be a network whose root, ρ, has out-degree two and is the source of exactly one cycle, B be the list of cycle branches on one path from the source to the sink as obtained by Algorithm 16, t be a triplet containing only labels from B, SD be the sink and its descendants, ICV be the labelled cycle vertices, and $t(N)$ the set of triplets. Given this input, Algorithm 18 returns True if for every edge in t the corresponding path in N (as described in Definition 2.6) does not use any edges from the other path from the source to the sink, and False otherwise in $\mathcal{O}(|X|)$ time.

Proof.
Let t be a triplet as described on $u, v, w \in X$. Suppose there is a branch $b \in B \cup SD$ such that $u, v, w \in b$. Then clearly this triplet did not use any cycle edges and thus True should be returned as is done in Line 4.

Likewise, if $t = u / v \setminus w$ and not all labels are from one branch, then v must be a cycle vertex. Then, this triplet also did not use cycle edges from the other path, as that triplet would have to be, without loss of generality, of the form $u/v|w$. Similarly, if $t = u \setminus v \setminus w$, then no edges from the other path could have been used, as that would require a label to be separated from the others by the root. These cases are handled in Line 6.

Moreover, if none of $u, v,$ and w is an element of SD, then no cycle edges from the other path could have been used. Indeed, Line 8 returns True.

If the above cases do not hold, then t must have exactly one branch that contains a label from SD. If t contains two labels from SD, z_1 and z_2, the triplet could be of the following forms: $z_1/z_2/n$, $z_1/z_2|n$, or $z_1, z_2|n$. The first case is already covered by Line 6. In the other cases, these triplets did use edges from the other path if $n \in ICV$. So True can be returned if $n \notin ICV$, and False otherwise. These cases are correctly handled by Lines 12 and 13.

The remaining case is if t contains only one label from SD, say z. Let n_1 and n_2 be the other remaining labels. If t is of the form $1|2,3$ and there is another triplet on n_1, n_2, and z of the form $1|2|3$, then n_1 and n_2 must have been from the same cycle branch such that the corresponding cycle vertex had out-degree three or higher. In this case, t did use edges from the other path and thus `False` is returned in Line 17.

Suppose z is in its own branch in t, then t is of the form $z/n_1 \setminus n_2$, $z|n_1, n_2$, or $z|n_1 \setminus n_2$. The first two cases are handled by Lines 6 and 17, respectively. The last case can only be a valid triplet if n_1 and n_2 are in the same cycle branch. If they are not, `False` is returned in Line 23. However, if n_1 and n_2 are in the same branch, then neither can be in ICV. So `True` is returned in Line 21.

Now, without loss of generality, suppose z shares its branch in t with n_2. Then the triplet could be of the following forms: $n_1|n_2, z$, $n_1|n_2 \setminus z$. Both triplets require $n_1, n_2 \notin ICV$ and so `True` is returned in Line 21.

Thus Algorithm 18 properly determines if t used any edges from the other path of the cycle.

Looking at the time complexity, since we can use a hashmap to find the triplets on any three labels, this algorithm only loops over the number of branches. So this can be done in $\mathcal{O}(|X|)$ time. $\square$