On Phylogenetic Encodings and Orchard Networks

Murakami, Yukihiro

DOI
10.4233/uuid:049932ab-4124-4639-a7e3-146ac4fd805d

Publication date
2021

Document Version
Final published version

Citation (APA)
Murakami, Y. (2021). *On Phylogenetic Encodings and Orchard Networks*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:049932ab-4124-4639-a7e3-146ac4fd805d

On Phylogenetic Encodings and Orchard Networks

On Phylogenetic Encodings and Orchard Networks

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op Maandag 29 November 2021 om 10:00 uur

door

Yukihiro Murakami

Master of Mathematics,
University of Oxford, Verenigd Koninkrijk,
geboren te Hiroshima, Japan.

Dit proefschrift is goedgekeurd door de

promotor: Dr.ir. L.J.J. van Iersel
promotor: Prof.dr.ir. K.I. Aardal

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Dr.ir. L.J.J. van Iersel,	Technische Universiteit Delft, promotor
Prof.dr.ir. K.I. Aardal,	Technische Universiteit Delft, promotor
Onafhankelijke leden:	
Prof.dr. M. Bordewich,	Durham University, Verenigd Koninkrijk
Prof.dr. D.C. Gijswijt,	Technische Universiteit Delft
Dr. J. Jansson,	Kyoto University, Japan
Dr. S. Linz,	The University of Auckland, Nieuw-Zeeland
Prof. dr. J.M.A.M. van Neerven	Technische Universiteit Delft, reservelid
Overig lid:	
Prof.dr. V. Moulton,	University of East Anglia, Verenigd Koninkrijk

Prof.dr. V. Moulton heeft in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Dit onderzoek is deels gefinancierd door de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Vidi-beurs 639.072.602).

Keywords: Phylogenetic Networks, Encodings, Orchard Networks

Printed by:

Front & Back:

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

CONTENTS

SUMMARY

Phylogenetic networks are a type of graph with vertices and edges, used to elucidate the evolutionary history of species. The fundamental goal of phylogenetic research is to infer the true phylogeny of species from raw data such as DNA sequences and morphological data. Most network inference methods require one to solve an NP-hard problem; furthermore, there is generally no guarantee of a unique network. One way of resolving this is to restrict our scope to networks within a certain class and to ask the following question.

What input data guarantees a unique network within this class?

Such a question brings us to the idea of *encodings*. A network class is encoded by a certain *building block*, such as displayed trees, splits, or induced inter-taxa distance matrices, if the building block distinguishes one network in the class from another. More precisely, no two networks in the same class may have the same set of building blocks. Often, encoding results give inspiration for polynomial-time algorithms for inferring networks within certain classes. Assuming to have data that corresponds to a network in that class, one may plausibly construct it as the unique network that is consistent with such information. We investigate encodings of different network classes in Chapters 3, 6 and 7.

The other half of the thesis, Chapters 4 and 5, introduces and investigates the class of *orchard networks*. Network classes are often computationally motivated to tackle an existing NP-hard problem, they are defined by their topological features, and they are usually accompanied by their biological relevance. Here, we give a recursive definition of orchards that make it an algorithmically interesting class, give a characterization based on edge coverings, and show that orchard networks are precisely those where every reticulate event is a horizontal transfer.

In addition to these results, we give a comprehensive analysis of how the recursive definition of orchards can be used to show when one network is contained in another. Biologically, stretches of DNA may evolve in a way that does not use the whole phylogenetic network of the species; being able to show that one network is contained in the other gives us a way to check if a gene tree or gene network is consistent with the species network. Algorithmically, we provide the first linear time algorithm for solving the NETWORK CONTAINMENT problem for tree-child networks, which is a prominent subclass of orchard networks.

SAMENVATTING

Fylogenetische netwerken zijn een type grafen met knopen en lijnen, die worden gebruikt om de evolutionaire geschiedenis van soorten te beschrijven. Het fundamentele doel van fylogenetisch onderzoek is om de ware fylogenie van soorten af te leiden uit ruwe data zoals DNA-sequenties en morfologische gegevens. Voor de meeste netwerkinferentiemethoden is het nodig om een NP-moeilijk probleem op te lossen; bovendien is er in het algemeen geen garantie op een uniek netwerk. Een manier om dit op te lossen is om ons domein te beperken tot netwerken binnen een bepaalde klasse en de volgende vraag te stellen.

Welk type data garandeert een uniek netwerk binnen deze klasse?

Een dergelijke vraag brengt ons bij het idee van *coderingen*. Een netwerkklasse wordt gecodeerd door een bepaalde *bouwsteen*, zoals weergegeven bomen, splitsingen of geïnduceerde afstandsmatrices tussen taxa, als de bouwstenen het ene netwerk in de klasse onderscheiden van het andere. Nauwkeuriger gezegd, geen twee netwerken in dezelfde klasse mogen dezelfde verzameling bouwstenen hebben. Vaak geven coderingsresultaten inspiratie voor polynomiale tijd algoritmen voor het genereren van netwerken binnen bepaalde klassen. Gegeven data die overeenkomen met een netwerk in een zekere klasse, kan men dit netwerk construeren als het unieke netwerk dat consistent is met die data. We bespreken coderingen van verschillende netwerkklassen in Hoofdstukken 3, 6, en 7.

De andere helft van het proefschrift, Hoofdstukken 4 en 5, introduceert en onderzoekt de klasse van *orchard networks*. Netwerkklassen zijn vaak computationeel gemotiveerd om een bestaand NP-moeilijk probleem aan te pakken, ze worden gedefinieerd door hun topologische kenmerken en hebben vaak een zekere biologische relevantie. Hier geven een recursieve definitie van orchard networks die het een algoritmisch interessante klasse maken, geven we een karakterisering op basis van edge coverings en laten we zien dat orchard networks precies de netwerken zijn die beschreven kunnen worden als een boom met horizontale overdracht.

Bovendien analyseren we hoe de recursieve definitie van orchard networks kan worden gebruikt om te beslissen of een netwerk zich in een ander netwerk bevindt. Biologisch gezien kunnen stukken DNA evolueren op een manier die niet het hele fylogenetische netwerk van de soort gebruikt; het kunnen aantonen dat het ene netwerk in het andere zit geeft ons een manier om te controleren of een genenboom of genennetwerk consistent is met het soortennetwerk. Algoritmisch bieden we het eerste lineaire tijd algoritme voor het oplossen van het NETWORK CONTAINMENT-probleem voor tree-child networks, een prominente subklasse van orchard networks.

1

INTRODUCTION

1

1.1. PHYLOGENETICS

PHYLOGENETICS is the study of the evolutionary history of species, and lies at the cross section of mathematics, biology, and computer science. The main goal of phylogenetic research is to infer the true phylogeny of species at hand from raw data such as DNA sequences and morphological data. Understanding the evolutionary history of species is a foundational undertaking within the field of biology. Studying how lineages have originated is not only vital in the classification of living things, but also important in interpreting immunological patterns and examining developments of harmful organisms. This can, for example, lead to fast and effective ways to deal with pathogen outbreaks by inferring the most likely source of transmission. Most recently, techniques from phylogenetics were employed in the COVID-19 outbreak to detect multiple variants of the virus and the evolutionary histories therein [1].

Traditionally, trees (graphs without undirected cycles) were (and still are) used to elucidate the evolutionary picture, and serve instrumental in depicting speciation events with vertical descent. Unfortunately, other evolutionary mechanisms can throw a wrench in the works. Reticulate events such as hybridization and horizontal gene transfer can give rise to signals that cannot be represented on a single tree [2, 3]. Such events, as it turns out, are widespread and important evolutionary drivers [4–8]. A tangible example is the evolutionary history of wheat. Triticum aestivum, most commonly known as the common wheat used to make bread, is a byproduct of at least three hybridization events from other wheat species (see Figure 1.1) [9]. Hybridization events also occur within the evolutionary history of animals. Figure 1.2 shows a phylogeny of common lizards [10].

1.2. PHYLOGENETIC NETWORKS

PHYLOGENETIC networks are a generalization of trees that allow for the representation of not only vertical descent, but also reticulate events. While trees have dominated the evolutionary scene, phylogenetic networks have gained increasing attention in recent years [2, 3].

Given a set of extant taxa X, a *(non-binary directed phylogenetic) network* on X is a directed acyclic graph with

- a single *root* of indegree-0 and outdegree-1;
- *tree vertices* of indegree-1 and outdegree at least 2;
- *reticulations* of indegree at least 2 and outdegree-1;
- *leaves* of indegree-1 and outdegree-0 that are labelled bijectively by elements of X. The leaf set will sometimes be denoted as $L(N)$ or as X.

Note that we do not allow parallel edges in phylogenetic networks[1]. We will allow for

[1] A quick note on parallel edges. Often in literature, they are seen as redundant and are excluded from most phylogenetic networks. Indeed, it is easy to see for example, that a network N with parallel edges does not provide any more information than one obtained by deleting all but one parallel edges from N, in terms of the trees that they may display. Further, Occam's Razor dictates that the most simple explanation should be taken. However, there are many cases for employing parallel edges; one is to represent intra-specific hybridization. For more reasons to allow for parallel edges within networks, see, for example, [11].

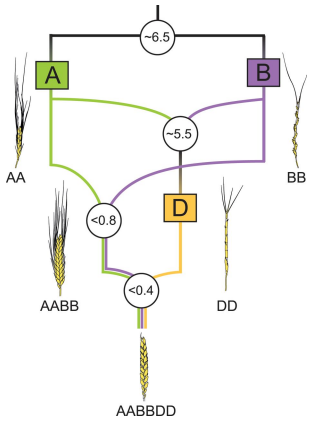


Figure 1.1: A directed phylogenetic network of wheat from [9]. The common wheat is indicated by the species at the very bottom, labelled AABBDD. The numbers indicate, in million year units, the approximate dates for divergence and the hybridization events.

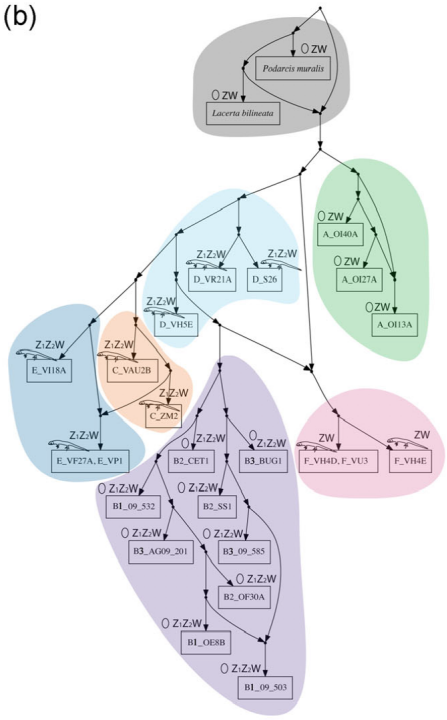


Figure 1.2: A directed phylogenetic network of common lizards from [10] containing three reticulation events. Each taxa is labelled by an egg or a lizard icon, which denotes the oviparous (egg-laying) or the viviparous (birth-giving) nature of the species. The network was used to illustrate the possible reversal of viviparity to oviparity within common lizards (Black and green clades are oviparous. The light blue clade contains viviparous species; we see the reversal within the purple clade, which contains oviparous species).

parallel edges in some of our graphs in Chapter 5.

The root is seen as the most recent common ancestor of the taxa set X; edges are directed, indicating the direction in which genetic material has passed. Tree vertices and reticulations represent speciation and hybridization events, though they are sometimes seen as ancestral species. The leaves, which are bijectively labelled by X, represent extant taxa. Edges directed into reticulations are called *reticulation edges*, and each non-reticulation edge is called a *tree edge*. A (non-binary directed phylogenetic) *tree* on X is a network on X with no reticulations.

Vertices of degree greater than 3 represent ambiguity in the order of how evolutionary events have unfolded. This occurs as a result of having insufficient data to accurately conclude which lineages bifurcated first (*soft polytomies*), but it could indicate a diver-

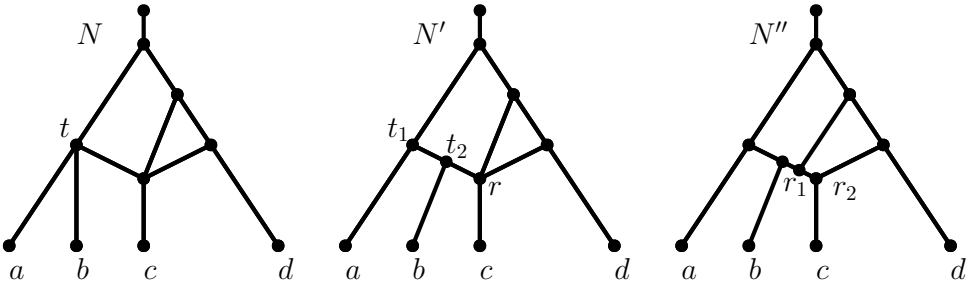


Figure 1.3: A non-binary network N, a semi-binary network N', and a binary network N'' on the taxa set $\{a,b,c,d\}$. The network N' is obtained from N by refining the vertex t into t_1 and t_2. The network N'' is obtained from N' by refining the vertex r into r into r_1 and r_2. The network N contains a *cherry* (a,b) and a *reticulated cherry* (c,d). The network N' *contains* N; the network N'' *contains* both N and N'. Cherries, reticulated cherries, and containment will be defined in Chapter 2.

gence of three or more lineages from a single lineage (*hard polytomies*). Most polytomies are soft, meaning that true phylogenies often contain dichotomous speciation events. It is possible also for more than two species to converge at the same time, which is called multispecies hybridization [12].

Such ambiguities are often used to classify networks. The following lists three classifications of networks in the order of most resolved / least ambiguous to most ambiguous. We say that a vertex in a network is *binary* if it is of degree at most 3. A tree vertex that is binary is called a *bifurcation*; a tree vertex that is not binary, i.e., one of degree greater than 3, is called a *multifurcation*. We say that a network is *binary* if all vertices are binary; it is *semi-binary* if all tree vertices are binary. We shall use the term *non-binary*, to mean 'not necessarily binary'. Therefore, all binary networks are semi-binary, and all networks are non-binary (see Figure 1.3).

Edges of phylogenetic networks either have directions or they do not. The direction represents the passage of genetic material and / or the passage of time. Directed networks are also referred to as 'evolutionary networks' or 'rooted networks', and aim to represent the explicit evolutionary history of the species [13]. On the other hand, undirected networks are often referred to as 'data-display networks' or 'unrooted networks'. As the pseudonym suggests, undirected networks are primarily used to exhibit the relatedness of the species as can be inferred from the data, although the direction in which genetic materials have passed is unknown. That said, unrooted networks are also used as evolutionary networks, just with the edge directions removed. In this thesis, the unrooted networks we consider in Chapter 7 are evolutionary networks. The main difference here is that an undirected network does not necessarily dictate the ancestor-descendant relationship, but rather the clades, clusters, and the splits that can be inferred from the input data. Figures 1.1 and 1.2 are two examples of directed phylogenetic networks. See Figure 1.4 for an example of a directed and an undirected network.

In addition to directed and undirected networks, researchers have looked into what are called *semi-directed networks*, in which only the reticulation edges are directed [14]. This gives information on all reticulations of the network, and they are often used in model-based methods, such as in pseudo-likelihood models [14], multispecies network

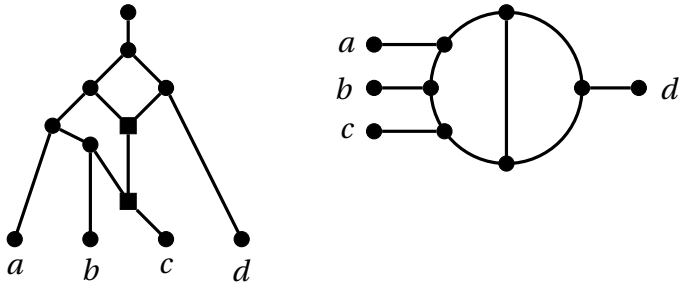


Figure 1.4: A directed (left) and an undirected (right) phylogenetic network on the leaf set $\{a, b, c, d\}$. In the directed network, the edges are directed downwards from the single root of indegree-0 to the leaves. The undirected network on the right was obtained by 'unrooting' the directed network, by removing the root vertex, suppressing degree-2 vertices, and removing directions from the edges.

coalescents [15], and Markov models [16]. We do not consider semi-directed networks in this thesis, and therefore we do not elaborate further on this topic.

1.2.1. EARLY USES OF PHYLOGENETIC NETWORKS

The field of phylogenetics has a rather young history. Charles Darwin is generally attributed the title 'the founding father of evolution' and rightfully so. He outlined the theory of natural selection in his book *On the Origin of Species* and was one of the first to use phylogenetic trees to depict (as a 'simile', in his words) the evolutionary relationship between different species (see Figure 1.5). There is however, evidence of phylogenetic networks being used long before Darwin was even born. A genealogical network of dogs was published in volume 8 of *Histoire Naturelle* in 1755 by Georges-Louis Leclerc, Comte de Buffon [17] (see Figure 1.6). The figure represents a geographical chart outlining how dog races have evolved differently under varying circumstances. We refer the interested reader to the following blog post on *The Genealogical World of Phylogenetic Networks* [18].

1.2.2. FROM SEQUENCE ALIGNMENTS TO PHYLOGENETIC NETWORKS

There are many ways of inferring or constructing a phylogenetic network from input data. For an overview on network building methods, see for example [2, 20]. Input data generally arrives in the form of a multiple sequence alignment, based on DNA sequences or morphological data.

A method based on parsimony searches for a network that minimizes the total number of mutations and / or the total number of reticulate events, which provides an explanation for the given input. The problem is known to be NP-hard even when the search space is restricted to phylogenetic trees [21]. A method based on maximum likelihood is based on computing the likelihood of a certain network given the input sequence alignment. This uses some model of DNA evolution, such as Jukes-Cantor [22], Kimura two parameter [23], or Kimura three parameter [24], which makes it possible to calculate the probabilities based on different transition rates. Inferring networks via maximum likelihood is also known to be NP-hard; in fact, even finding optimal branch lengths for a

1

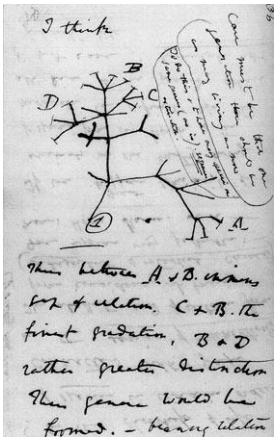


Figure 1.5: Darwin's famous sketch of a phylogenetic tree [19].

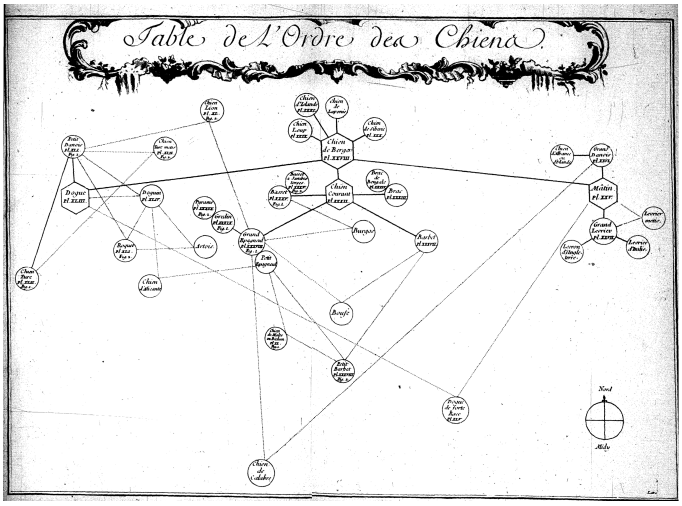


Figure 1.6: The first phylogenetic network (as far as we know) on the evolutionary history of dogs by Buffon [17, 18]

given tree that would maximize the likelihood is already NP-hard [25].

Other methods first convert the sequence alignment into a more digestible format. Two such formats include subnetworks and distance matrices. Inference methods using subnetworks are sometimes called *supernetwork methods*, and they seek a network with the least number of hybridization events that *display* the given subnetworks. Here, we say that a network *displays* another network if the latter can be obtained from the former by deleting reticulation edges and suppressing indegree-1 outdegree-1 vertices. The method again suffers from the issue of NP-hardness, even when the input consists of two trees [26] or a set of triplets and we look for a level-1 network (networks where the cycles are node-disjoint) [27]. On the other hand, distance matrices on the taxa can be obtained from alignments by simply taking the Hamming distance, or applying some model of DNA evolution along it [2]. A network that induces the same matrix (where every element is induced as a path or a shortest path of the network, or the average path length) is inferred, possibly with the fewest number of reticulations. Distance-based methods are noteworthy due to the polynomial running time of some of them [28, 29], which is attractive in relation to their NP-hard counter-part inference methods mentioned above. That said, they are less reliable when compared to other methods. Therefore, they are often used to obtain a first estimate network, after which methods such as *Bayesian inference* can be used to infer a more likely or a more parsimonious network [30]. Within a Bayesian framework, in particular through a Markov Chain Monte Carlo Metropolis-Hastings algorithm, for a given prior distribution, a specified phylogenetic network space is traversed via rearrangement moves to obtain a network with a higher likelihood for the given input [2, 31].

1.3. CONTRIBUTION OF THE THESIS

We first go over preliminary definitions and results in Chapter 2. The body of the thesis, Chapters 3 – 7 include results that either have been accepted in journals for publication, or those that are currently under peer review and are available on arXiv. Chapters 3, 4, 5, and 6 contain results on directed networks; Chapter 7 contains results on undirected networks. The thesis chapters complement one another on multiple themes. Chapters 3, 6, and 7 all explore results pertaining to encodings, which are unique characterizations of certain phylogenetic networks by certain building blocks. Encodings are particularly helpful when trying to infer a network; given perfect data, one can guarantee that a unique output solution that fits the data will be found. Chapters 4 and 5 both investigate characterizations of network classes. Phylogenetic networks have been categorized into many topological classes for both biological and computational incentives (for an overview of a few network classes, see Chapter 2). In Chapter 4, we introduce the class of orchard networks via means of a recursive characterization. This characterization also gives an encoding of orchard networks, as every orchard network is encoded by the smallest sequence that reduces them. In Chapter 5, we take inspiration from this recursive characterization to create a non-recursive characterization for orchard networks, and show that a similar characterization can be used for the class of tree-based networks. In the same chapter, we provide a third biologically meaningful characterization of orchards, by means of network vertex time labelling.

1.3.1. NETWORK CLASSES

As stated above, networks are categorized into different topological classes. Here, we give rough definitions for the network classes that appear in this thesis; formal definitions will be given in Chapter 2. The *level* of a network refers to the minimum number of reticulation edges that need to be removed from every biconnected component to obtain a tree [32]. A network of level-k is then called a *level-k network*. A network is *tree-child* if every non-leaf vertex has a child that is a leaf or a tree vertex. A network is *orchard* if it can be reduced to a network on a single leaf by applying a sequence of so-called cherry reductions.

1.3.2. ENCODINGS

A *building block* of a network is any information that can be obtained from a phylogenetic network, such as the set of displayed subnetworks or the induced distance matrix. *Encodings* refer to building blocks that uniquely characterize a network. We say that a class of networks $\mathscr{C}$ is *encoded* by a certain building block S, if for any network $N \in \mathscr{C}$ and for any network $N' \in \mathscr{C} \setminus \{N\}$, we have that $S(N) \neq S(N')$. This is sometimes called a *weak encoding* [33]. This is in contrast to another definition of encodings, where the second network does not need to be contained in the class $\mathscr{C}$ of networks. Note that this latter definition of encodings is stronger than the former definition of encodings. We shall call this a *strong encoding*. Most results that we reference and present in this thesis are weak encoding results. We reference one result in Chapter 6 that uses the notion of a strong encoding. There, we shall make this distinction clear. Elsewhere in the thesis, unless otherwise stated, we will use the term encoding to mean a weak encoding.

To see the bigger picture, we investigate encoding results to create algorithms to infer unique networks from plausible input sets. The input sets in this case would be the building blocks. By proving that a particular class of networks is encoded by some building block, we can guarantee a unique solution given that we have perfect data of a network in the given class.

To understand where encoding results fit in the grand scheme of phylogenetics, we must first understand the difference between *constructing* and *reconstructing* a phylogenetic network. One notion is motivated by the other. When *constructing* a network, one is often concerned with obtaining a network that agrees with the given input, whilst optimising certain criteria. Problems in this category are generally NP-hard, and exact solutions, as well as solutions obtained from heuristics are generally non-unique. A few of these inference method categories were mentioned in Section 1.2.2. Such uncertainty invites follow-up questions. What restrictions can we impose to make the problem easy? How does changing the input data affect this decision making? When can we expect a unique solution?

These questions are the primary motivators for investigating network *reconstruction*. When reconstructing a network, we start with the network, find the building blocks of the network (e.g., subnetworks, distance matrices) that we are interested in, and try to reconstruct the network from the building block. The aim is to show *reconstructibility*, that for a given set $\mathscr{C}$ of networks, if two networks in $\mathscr{C}$ have the same building blocks, then the networks must be isomorphic, thereby providing a unique solution when the space is limited to $\mathscr{C}$. The terms reconstructibility and encodings will be used interchangeably,

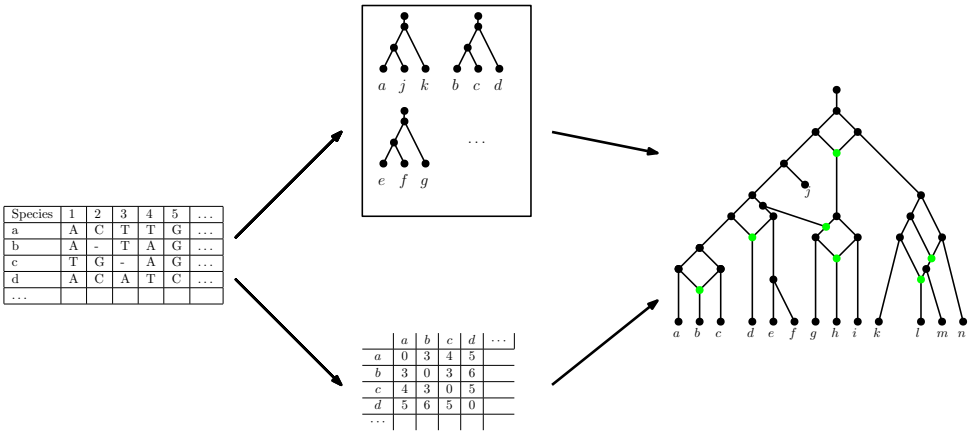


Figure 1.7: Visualization of the steps involved in a building block approach. The raw data is first converted to a building block (in this case either sets of triplets or a distance matrix), which is then used to construct a network that displays it.

through the thesis.

Therefore, the main difference in constructing and reconstructing a network comes down to the starting point (see Figure 1.7). The input for constructing a network is the raw data; the input for reconstructing a network is the network itself. When forging reconstruction proofs, we may theoretically assume to know what the network is. In practice, this is unfortunately not the case. Luckily, most reconstruction proofs may give intuitions for how a heuristic may be created. In presence of perfect data, where the data encodes some network, these methods will yield that network.

Results concerning reconstructibility are often shown in a constructive manner. Given the set of building blocks, it is often customary to identify key features of networks that would display such building blocks. The corresponding pendant structures are removed from the building blocks, and the proof recurses on the updated building blocks. Because of this constructive nature, inferring an algorithm follows quite naturally. This may be the most important aspect of reconstructibility results. The problem of finding a network that is consistent with the given input becomes 'easier' if we may assume that the output network is a member of a particular network class. Indeed, the assumption allows us to leverage our understanding of the network class. Another neat feature is that these algorithms often run in polynomial-time.

Another reason to consider encodings is to define a distance metric for the class. Let $\mathscr{C}$ be a class of networks that are encoded by some type of building block S, where the set of building blocks for a network $N \in \mathscr{C}$ is denoted by $S(N)$. Define a distance metric between two networks N, N' in $\mathscr{C}$ by taking the size of the symmetric difference between the two building blocks $|S(N) \triangle S(N')|$. Since no two distinct networks have the same building blocks, and because of the triangle inequality, it is easy to check that this yields a distance metric within the class.

We now describe three building blocks we consider in this thesis. The subnetworks (Chapter 3), the ancestral profiles (Chapter 6), and the distance matrices (Chapter 7).

SUBNETWORKS (CHAPTER 3)

Roughly speaking, a *subnetwork* of a network is obtained by deleting vertices and / or edges, and suppressing degree-2 vertices. In a *subnetwork* approach, we take as input a set $\mathscr{S}$ of networks, and seek to find a network N such that every network in $\mathscr{S}$ is a subnetwork of N. In a subnetwork reconstruction approach, we seek to show that networks in a particular class are encoded by their set of subnetworks.

We shall first visit some of the existing literature on construction methods that use subnetworks. Consider the case where our subnetworks are trees, all on the same set of taxa. Finding *a* network that displays all input trees is easy. In fact, there are notions of 'universal' tree-based and 'universal' tree-child networks that display all $(2n-3)!!$ possible (binary) directed trees on n leaves [34, 35]. That being said, finding a network with minimal hybridization number that displays a given set of trees, is NP-hard even for two input trees [36]. This problem is called the HYBRIDIZATION NUMBER problem, and is perhaps the most studied problem in combinatorial phylogenetics. Techniques including, but not limited to, agreement forests and cherry-picking sequences (the latter of which will be relevant in Chapter 4) have been developed to tackle the problem thus far [35, 37, 38]. With certain restrictions, however, solutions can be obtained rather efficiently. Huynh et al. proposed the first polynomial-time algortihm for combining a set of trees into a level-1 network with the fewest reticulations as possible [39].

Inferring so-called *consensus trees* / *networks* have also gained attention for sets of input trees. Such methods compare the induced clusters and splits[2] of the input trees, and seeks one that agrees with all or most input trees. For an overview, see [2]. Consensus trees / networks can be constructed in polynomial-time [40, 41], though in some instances, they will lead to outputs consisting of highly unresolved vertices.

Others have also considered trees on fewer leaves as building blocks. The classical BUILD algorithm by Aho et al. returns a tree that displays a set of given input triplets (trees on three leaves), should one exist, in polynomial-time [42]. In case a tree solution does not exist, others have created algorithms for outputting level-1 [43, 44], level-2 [45], and level-k networks [46] from triplet inputs.

Trinet inputs, which are networks on three leaves, have also been considered [47, 48]. The main reason for considering three leaved subnetworks is for simplicity and for scalability. The number of different possible networks on three leaves (given a few restrictions on, for example, the number of reticulations) is still at a manageable size. Constructing triplets or trinets from DNA data is not an obvious process, but in principle they are manageable since the size of the networks is highly limited. A few supernetwork methods with trinet inputs such as Lev1athan [44] and TriLoNet [47] also provide algorithms for obtaining a set of triplets and trinets, from sequence alignments.

The motivation for studying encoding results, as explained before, arises from the desire for a unique solution to be found, together with an algorithm to find one. In the language of supernetwork approaches, this means that if a set of subnetworks did not encode a particular network, there could be more than one network that displays the same set of subnetworks. When considering trees as subnetworks, Pardi and Scornavacca

[2] For directed trees, the clusters refer to the subset of leaves that can be reached from a certain internal vertex of the tree; for undirected trees, the splits are a partition of the leaf-set mirrored in the two components of the tree obtained by deleting an edge of the tree.

somewhat resolved this distinguishability issue by considering 'canonical forms' of networks with branch lengths; however the problem still persists in general [49]. This is, of course, one of the main reasons behind proving reconstructibility results on subnetworks.

In related literature regarding reconstructibility results, a certain class of networks called *regular networks* are encoded by their set of displayed trees [50]. It is also known from Gambette et al. [51] that level-1 networks with girth (shortest cycle in underlying graph) at least 5 are reconstructible from their triplets. It has been shown that tree-child networks are encoded by trinets [52] but not by trees. In the same paper, it was shown that level-2 networks are also encoded by their induced trinets [52]. Recently, it was shown that level-3 networks are encoded by their quarnets, but not by their trinets [33]. It was also shown that orchard networks are encoded by their trinets [53].

Results from Chapter 3 In Chapter 3, we consider *maximum lower-level subnetworks (MLLSs)*, which are subnetworks obtained by deleting a single reticulation edge from highest level biconnected component. These subnetworks have not been studied before; they are interesting to study as they provide the first step for constructing a complex network from simpler networks. Indeed, in principle, the presented method gives a way to reconstruct level-k networks from a set of level-$k-1$ networks.

We shall show that binary level-k tree-child networks, where $k \geq 2$, are encoded by their MLLSs. Following this, we also provide a polynomial-time algorithm for reconstructing a tree-child network from its full set of MLLSs.

The results of this chapter has been published in [54].

ANCESTRAL PROFILES / μ-REPRESENTATIONS (CHAPTER 6)

Unlike trees, networks allow for vertices to have more than one path to the same leaf. Given an n-leaf directed network, assign a vector of length n to each vertex where the ith element of the vector is the number of distinct paths to the ith leaf. We call the multiset of vectors of all vertices in the network the *μ-representation* of the network [55]. They are also called *ancestral profiles* [56].

Ancestral profiles have thus far been studied for encoding results on the class of semi-binary time-consistent tree-sibling networks [55] and tree-child networks [57]. For the class of binary stack-free orchard networks [56, 58], a stronger encoding result was shown: it was shown that given a binary stack-free orchard network N and a binary stack-free network N' (not necessarily orchard), N and N' have the same ancestral profiles if and only if they are isomorphic. These encodings were also used to give a natural metric on the network classes [55–57].

Unfortunately, there is not much of a biological interpretation behind ancestral profiles. The number of paths can, for example, denote the different ways in which genetic material has passed. Or, it could indicate some difference measure in the genome of the ancestral species and an extant species [56]. However, it is, simply put, a form of data that provides enough information to identify networks within certain classes and can therefore be used as basis for metrics, for example.

Results from Chapter 6 We give a counter-example to Theorem 3.1 of [58], to show that semi-binary stack-free orchard networks are *not* reconstructible from their ancestral profiles within the space of stack-free networks. We give intuition for why this is, and contribute by suggesting possible ways to prove a similar claim, that semi-binary stack-free orchards are reconstructible from their ancestral profiles. We also introduce the class of tree-clone-free networks, which are networks wherein every tree vertex has a different μ-vector. We show preliminary results for this class of networks, such as the relationship between two vertices with the same μ-vector, as well as the trivial automorphism group on tree-clone-free networks.

The results of this chapter have not yet been published.

DISTANCE MATRICES (CHAPTER 7)

The third building block we consider in this thesis for encodings is the distance matrix and generalizations thereof. A (metric) distance matrix on a set X is an $|X| \times |X|$ symmetric matrix with non-negative elements $d(i, j)$ for $i, j \in X$, such that

1. every element in the matrix entry is positive except for those on the diagonal, which are all zeroes;

2. the triangle inequality is satisfied. That is, for elements $a, b, c \in X$,

$$d(a, c) \leq d(a, b) + d(b, c).$$

We focus on undirected networks in Chapter 7. Every undirected network induces a shortest distance matrix, where the rows and columns of the matrix are indexed by the leaves of the network, and every element $d(i, j)$ of the matrix represents the shortest distance between leaves i and j. We say that a network *realizes* a distance matrix M if its induced shortest distance matrix is equal to M. In this thesis, we shall also consider the longest distance matrices, which can be obtained by taking each element $d(i, j)$ to be the length of a longest path between leaves i and j.

Encoding results in phylogenetics have also considered different variants of distance matrices, called the *sets* and *multisets* of distances induced by networks [29, 59]. Given a network N on X, the *multisets of distances* on a set X induced by N is the $|X| \times |X|$ symmetric matrix with multisets of distances of all paths between leaf pairs as elements. The *sets of distances* has the set of distance of all paths between leaf pairs as its matrix elements. The difference between sets and multisets of distances is that the latter takes into account the multiplicity of each path length, while the former does not. Observe that the shortest distance matrix and the longest distance matrix can be obtained from the sets of distances, and that the sets of distances can be obtained from the multisets of distances.

Finding a weighted undirected graph that realizes a distance matrix has applications not only in phylogenetics [2, 13], but also in psychology [60, 61], electricity networks [62, 63], information theory [64], and other areas. In their seminal paper, [62] showed that a necessary and sufficient condition for a distance matrix to be realizable on a graph is for it to be a metric space. While this gave existence for graph realizations on any metric spaces, such realizations were not necessarily unique.

In phylogenetics, distances are defined between pairs of taxa to denote the number of character changes (in terms of the nucleotide bases in DNA) or the evolutionary / genetic distance between them. Such distance data are often obtained from multiple sequence alignments by first taking the Hamming distance and applying a transformation based on a chosen model of evolution (e.g., Jukes-Cantor, Kimura 2-parameter, etc.) [2]. Two foundational phylogenetic tree distance-based methods are UPGMA [65] and Neighbor-Joining [66] (for an overview, see, for example, [67]). The main advantage of distance-based methods is that they are fast compared to other phylogeny construction methods such as maximum likelihood and maximum parsimony [68]. The focus now has shifted towards constructing phylogenetic networks exactly from distance matrices [28, 69].

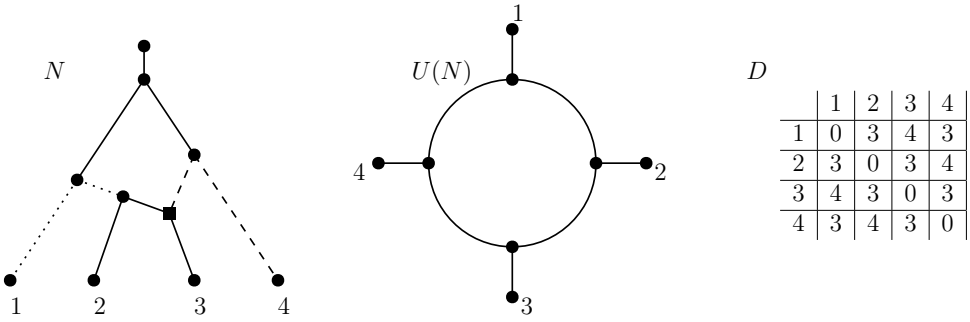
Refocusing back to the idea of encodings, we say that a network is *encoded* by their induced distance matrix if it is the only network that realizes the distance matrix. Not much is known about encoding results for undirected networks. It was shown in [70] that optimal *cactus graphs* are reconstructible from their shortest distances. Cactus graphs are connected graphs in which each edge belongs to at most one cycle. Here, optimal means that the sum of all edge weights is minimal over all cactus graphs. Another undirected network distance encoding result is over level-k networks. We showed in [71] that undirected unweighted binary level-2 networks are reconstructible from their multisets of distances. In the same paper, we showed that level-2 networks were not reconstructible in general from their shortest distance matrices. These results of [71] are not included in this thesis as they follow from the stronger results presented in Chapter 7. For directed networks, distance encoding results are relatively well studied. Reconstructibility results for tree-child networks from the sets of distances [29] and the multisets of distances [59], as well as reconstructibility results for so-called normal networks from shortest distance matrices [72, 73] have been shown. For more on reconstructibility results for directed networks, see Section 7.1.

Results from Chapter 7 In Chapter 7, we investigate reconstructibility results by considering shortest and longest distance matrices as building blocks. We first show that undirected networks with, in a certain sense, enough leaves, are reconstructible from their induced shortest distance matrix. Then, we show reconstructibility results for the class of undirected level-2 networks. We show that level-2 networks are reconstructible from their induced shortest and longest distance matrices. Since level-2 networks are not reconstructible from just their shortest distances [71], the additional longest distances give the necessary information for this encoding result. Following this, we give a characterization of level-2 networks that are not reconstructible from their induced shortest distance matrix.

The results of this chapter has been published in [74].

1.3.3. ORCHARD NETWORK CHARACTERIZATIONS

Orchard networks are a class of networks that can be reduced by a sequence of cherry reductions (called cherry-picking sequences). Cherry reductions are graph operations where subgraphs on two leaves and their parents are altered to obtain a new network. We introduced the network class as *cherry-picking networks* in [75]; the same network



	1	2	3	4
1	0	3	4	3
2	3	0	3	4
3	4	3	0	3
4	3	4	3	0

Figure 1.8: An orchard network N with a cherry-picking sequence $(3,2)(1,2)(3,4)(2,4)$, with its corresponding cherry cover indicated by the colors. The undirected version $U(N)$ of N obtained by removing edge directions from N. The shortest distance matrix D induced by $U(N)$, given that every edge of $U(N)$ is of distance 1.

class for the binary case was independently discovered in [56], where it the class was called the class of orchard networks. We shall call the class orchard networks instead of cherry-picking networks throughout this thesis. Chapters 4 and 5 will go over the current state of the art for orchard networks, with results from [75] and [76], respectively; a rigorous definition of the network class, results pertaining to cherry reductions and network containment, a recursive encoding of orchard networks using smallest cherry-picking sequences, and two non-recursive characterizations of orchards, and more, shall be explored.

Cherry picking sequences were developed to tackle variations on the HYBRIDIZATION NUMBER problem, of finding a network with a minimum number of reticulations that displays a given set of trees [35, 38, 77, 78]. Two leaves of a tree form a *cherry* if they share a common parent—by successively 'picking' cherries (removing one of the leaves in a cherry) from the set of input trees, we obtain a sequence of cherries that ultimately reduce each input tree to a tree on a single leaf. This sequence of cherries corresponds to some network that contains the set of all input trees. Therefore, the actions of these sequences were until now defined only for trees; we start by extending this definition to networks.

ORCHARD NETWORKS AND NETWORK CONTAINMENT (CHAPTER 4)

We investigate the correspondence between orchard networks and the sequences that reduce them, and ultimately show that some orchard networks are encoded by their *smallest* cherry-picking sequences (Theorem 10). Although [35] and [77] mention how one can construct some network from a cherry-picking sequence, no further characterizations for the type of networks that can be obtained from such sequences have been investigated. Here, we reintroduce these constructions as a way to reverse the reduction. As different networks can be reduced by the same cherry-picking sequence, this reversal cannot be unique. Hence, we investigate several constructions corresponding to the different reversals of the reductions. Each construction then leads to a reconstructible[3]

[3]The word choice here is intentional, with respect to the reconstructibility / encoding theme that we have discussed in Section 1.3.2. We shall see that the networks in the reconstructible classes are uniquely encoded / characterized by the smallest cherry-picking sequences that reduce them.

class of networks, for which we can prove relations between containment and reduction by cherry-picking sequences.

These relations depend on a careful definition of what it means for a network to be a *subnetwork* of and to be *contained* in another network. A network N' is *contained* in another network N if N' can be obtained from N by deleting reticulation edges, suppressing degree-2 vertices, and by contracting edges. We show that within particular classes of orchard networks, if a sequence for a network N reduces another network N', then N' is contained in N (Lemma 33). The converse holds when the two input networks are tree-child, but not for orchard inputs (Theorem 12).

It turns out that the class of tree-child networks is contained in the class of orchard networks, as each tree-child network has a special type of cherry-picking sequence—a *tree-child sequence*—that reduces it. We examine how these sequences can be used to solve NETWORK CONTAINMENT for tree-child networks. NETWORK CONTAINMENT is the following problem: for given networks N and N' on the same set of species, decide whether N contains N'. In particular, we show that a tree-child sequence for a tree-child network N reduces another tree-child network N' if and only if N' is contained in N (Theorem 13). Following this, we provide a linear-time algorithm for NETWORK CONTAINMENT for inputs of tree-child networks (Algorithm 7). The algorithm has been implemented in Python, which shows that the linear theoretical running time is achievable in practice.

The results of this chapter has been published in [75].

CHERRY COVERS AND HGT-TIME CONSISTENT LABELLING (CHAPTER 5)

In Chapter 5, we give two non-recursive characterizations of orchard networks. The first characterization is based on covering the edges of the network using so-called *cherry shapes*. Such a characterization of a network class is often called a *structural characterization*, i.e., one that is not recursive, but static. Structural characterizations are important, as they can be used to prove numerous results about the network class. Indeed, they provide a tangible property of the network class that can be easy to work with in some situations. The second characterization is based on a vertex time labelling, called *HGT-time consistent labelling*. The labelling requires that exactly one parent of each reticulation vertex (hybridization event) is contemporaneous with the reticulation vertex, thereby showing that binary orchard networks are merely trees with horizontal edges added. This gives the first biologically relevant motivation for considering orchard networks.

Structural characterizations already have been discovered for tree-based networks. Binary tree-based networks have been characterized in the form of forbidden substructures [79], matchings [80], and using antichains and path-partitions [81]. Non-binary tree-based networks have been characterized by extending the matching notion [80]. For the class of orchard networks, a structural characterization has not been found before. Therefore in Chapter 5, we provide the first structural characterization of orchard networks, using cherry covers. We also show that the class of tree-based networks can be characterized using cherry covers. One consequence of these results will be that the class of orchard networks is contained in the class of tree-based networks.

1

Part of the presented results of this chapter has been published in [76], while the results of Section 5.5 have not yet been published.

REFERENCES

[1] P. Forster, L. Forster, C. Renfrew, and M. Forster, *Phylogenetic network analysis of sars-cov-2 genomes,* Proceedings of the National Academy of Sciences **117**, 9241 (2020).

[2] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications* (Cambridge University Press, 2010).

[3] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L. Stougie, and J. Whitfield, *Networks: expanding evolutionary thinking,* Trends in Genetics **29**, 439 (2013).

[4] R. Abbott, D. Albach, S. Ansell, J. W. Arntzen, S. J. Baird, N. Bierne, J. Boughman, A. Brelsford, C. A. Buerkle, R. Buggs, *et al.*, *Hybridization and speciation,* Journal of evolutionary biology **26**, 229 (2013).

[5] B. E. Goulet, F. Roda, and R. Hopkins, *Hybridization in plants: old ideas, new techniques,* Plant physiology **173**, 65 (2017).

[6] D. A. Wickell and F.-W. Li, *On the evolutionary significance of horizontal gene transfers in plants,* New Phytologist **225**, 113 (2020).

[7] P. J. Keeling and J. D. Palmer, *Horizontal gene transfer in eukaryotic evolution,* Nature Reviews Genetics **9**, 605 (2008).

[8] A. Wagner, R. J. Whitaker, D. J. Krause, J.-H. Heilers, M. Van Wolferen, C. Van Der Does, and S.-V. Albers, *Mechanisms of gene flow in archaea,* Nature Reviews Microbiology **15**, 492 (2017).

[9] T. Marcussen, S. R. Sandve, L. Heier, M. Spannagl, M. Pfeifer, K. S. Jakobsen, B. B. Wulff, B. Steuernagel, K. F. Mayer, O.-A. Olsen, *et al.*, *Ancient hybridizations among the ancestral genomes of bread wheat,* science **345** (2014).

[10] J. L. Horreo, T. Suarez, and P. S. Fitze, *Reversals in complex traits uncovered as reticulation events: Lessons from the evolution of parity-mode, chromosome morphology, and maternal resource transfer,* Journal of Experimental Zoology Part B: Molecular and Developmental Evolution **334**, 5 (2020).

[11] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *Polynomial-time algorithms for phylogenetic inference problems,* in *International Conference on Algorithms for Computational Biology* (Springer, 2018) pp. 37–49.

[12] J. Ottenburghs, *Multispecies hybridization in birds,* Avian Research **10**, 1 (2019).

[13] D. A. Morrison, *An introduction to phylogenetic networks* (RJR productions, 2011).

[14] C. Solís-Lemus and C. Ané, *Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting,* PLoS genetics **12**, e1005896 (2016).

[15] H. Baños, *Identifying species network features from gene tree quartets under the coalescent model,* Bulletin of mathematical biology **81**, 494 (2019).

[16] E. Gross and C. Long, *Distinguishing phylogenetic networks,* SIAM Journal on Applied Algebra and Geometry **2**, 72 (2018).

[17] G.-L. L. Buffon Comte de, *Histoire Naturelle, générale et particulière, avec la description du Cabinet du Roi*, Vol. 5 (1755).

[18] D. Morrison, *The first phylogenetic network (1755),* (February 26, 2012), http://phylonetworks.blogspot.com/2012/02/first-phylogenetic-network-1755.html.

[19] C. Darwin, *On the Origin of Species by Means of Natural Selection* (Murray, London, 1859).

[20] V. Makarenkov, D. Kevorkov, and P. Legendre, *Phylogenetic network construction approaches,* Applied mycology and biotechnology **6**, 61 (2006).

[21] L. R. Foulds and R. L. Graham, *The steiner problem in phylogeny is np-complete,* Advances in Applied mathematics **3**, 43 (1982).

[22] T. H. Jukes, C. R. Cantor, *et al.*, *Evolution of protein molecules,* Mammalian protein metabolism **3**, 21 (1969).

[23] M. Kimura, *A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences,* Journal of molecular evolution **16**, 111 (1980).

[24] M. Kimura, *Estimation of evolutionary distances between homologous nucleotide sequences,* Proceedings of the National Academy of Sciences **78**, 454 (1981).

[25] G. Jin, L. Nakhleh, S. Snir, and T. Tuller, *Maximum likelihood of phylogenetic networks,* Bioinformatics **22**, 2604 (2006).

[26] M. Bordewich and C. Semple, *On the computational complexity of the rooted subtree prune and regraft distance,* Annals of combinatorics **8**, 409 (2005).

[27] J. Jansson, N. B. Nguyen, and W.-K. Sung, *Algorithms for combining rooted triplets into a galled phylogenetic network,* SIAM Journal on Computing **35**, 1098 (2006).

[28] D. Bryant and V. Moulton, *Neighbor-net: an agglomerative method for the construction of phylogenetic networks,* Molecular biology and evolution **21**, 255 (2004).

[29] M. Bordewich and N. Tokac, *An algorithm for reconstructing ultrametric tree-child networks from inter-taxa distances,* Discrete applied mathematics **213**, 47 (2016).

[30] S. Guindon and O. Gascuel, *A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood,* Systematic biology **52**, 696 (2003).

[31] D. Wen, Y. Yu, and L. Nakhleh, *Bayesian inference of reticulate phylogenies under the multispecies network coalescent,* PLoS genetics **12**, e1006006 (2016).

[32] C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung, *Computing the maximum agreement of phylogenetic networks,* Theoretical Computer Science **335**, 93 (2005).

[33] L. Nipius, *Rooted binary level-3 phylogenetic networks are encoded by quarnets*, Bachelor's thesis, Delft University of Technology (2020).

[34] M. Hayamizu, *On the existence of infinitely many universal tree-based networks,* Journal of Theoretical Biology **396**, 204 (2016).

[35] S. Linz and C. Semple, *Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies,* Advances in Applied Mathematics **105**, 102 (2019).

[36] M. Bordewich and C. Semple, *Computing the minimum number of hybridization events for a consistent evolutionary history,* Discrete Applied Mathematics **155**, 914 (2007).

[37] M. Baroni, S. Grünewald, V. Moulton, and C. Semple, *Bounding the number of hybridisation events for a consistent evolutionary history,* Journal of mathematical biology **51**, 171 (2005).

[38] P. J. Humphries, S. Linz, and C. Semple, *Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies,* Bulletin of mathematical biology **75**, 1879 (2013).

[39] T. N. Huynh, J. Jansson, N. B. Nguyen, and W.-K. Sung, *Constructing a smallest refining galled phylogenetic network,* in *Annual International Conference on Research in Computational Molecular Biology* (Springer, 2005) pp. 265–280.

[40] J. Jansson, C. Shen, and W.-K. Sung, *Improved algorithms for constructing consensus trees,* Journal of the ACM (JACM) **63**, 1 (2016).

[41] N. Tahiri, M. Willems, and V. Makarenkov, *A new fast method for inferring multiple consensus trees using k-medoids,* BMC evolutionary biology **18**, 1 (2018).

[42] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions,* SIAM Journal on Computing **10**, 405 (1981).

[43] J. Jansson and W.-K. Sung, *Inferring a level-1 phylogenetic network from a dense set of rooted triplets,* Theoretical Computer Science **363**, 60 (2006).

[44] K. T. Huber, L. Van Iersel, S. Kelk, and R. Suchecki, *A practical algorithm for reconstructing level-1 phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 635 (2010).

[45] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout, *Constructing level-2 phylogenetic networks from triplets,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 667 (2009).

[46] T.-H. To and M. Habib, *Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time,* in *Annual Symposium on Combinatorial Pattern Matching* (Springer, 2009) pp. 275–288.

[47] J. Oldman, T. Wu, L. van Iersel, and V. Moulton, *Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories,* Molecular biology and evolution **33**, 2151 (2016).

[48] S. Kole, *Constructing level-2 phylogenetic networks from trinets*, Master's thesis, Technische Universiteit Delft, the Netherlands (2020).

[49] F. Pardi and C. Scornavacca, *Reconstructible phylogenetic networks: do not distinguish the indistinguishable,* PLoS computational biology **11**, e1004135 (2015).

[50] S. J. Willson, *Regular networks can be uniquely constructed from their trees,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 785 (2011).

[51] P. Gambette, K. T. Huber, and S. Kelk, *On the challenge of reconstructing level-1 phylogenetic networks from triplets and clusters,* Journal of Mathematical Biology **74**, 1729 (2017).

[52] L. van Iersel and V. Moulton, *Trinets encode tree-child and level-2 phylogenetic networks,* Journal of Mathematical Biology **68**, 1707 (2014).

[53] C. Semple and G. Toft, *Trinets encode orchard phylogenetic networks,* Journal of Mathematical Biology **83**, 1 (2021).

[54] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing tree-child networks from reticulate-edge-deleted subnetworks,* Bulletin of mathematical biology **81**, 3823 (2019).

[55] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, *A distance metric for a class of tree-sibling phylogenetic networks,* Bioinformatics **24**, 1481 (2008).

[56] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[57] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[58] A. Bai, P. L. Erdős, C. Semple, and M. Steel, *Defining phylogenetic networks using ancestral profiles,* Mathematical Biosciences **332**, 108537 (2021).

[59] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[60] J. P. Cunningham, *Free trees and bidirectional trees as representations of psychological distance,* Journal of mathematical psychology **17**, 165 (1978).

[61] R. W. Schvaneveldt, F. T. Durso, and D. W. Dearholt, *Network structures in proximity data,* in *Psychology of learning and motivation*, Vol. 24 (Elsevier, 1989) pp. 249–284.

[62] S. L. Hakimi and S. S. Yau, *Distance matrix of a graph and its realizability,* Quarterly of applied mathematics **22**, 305 (1965).

[63] S. Forcey and D. Scalzo, *Phylogenetic networks as circuits with resistance distance,* Frontiers in Genetics **11** (2020).

[64] A. Dewdney, *Diagonal tree codes,* Information and Control **40**, 234 (1979).

[65] R. R. Sokal and C. D. Michener, *A statistical method for evaluating systematic relationships.* Univ. Kansas, Sci. Bull. **38**, 1409 (1958).

[66] N. Saitou and M. Nei, *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Molecular biology and evolution **4**, 406 (1987).

[67] F. Pardi and O. Gascuel, *Distance-based methods in phylogenetics,* in *Encyclopedia of Evolutionary Biology* (Elsevier, 2016) pp. 458–465, 1st ed.

[68] M. K. Kuhner and J. Felsenstein, *A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates.* Molecular biology and evolution **11**, 459 (1994).

[69] H.-L. Chan, J. Jansson, T.-W. Lam, and S.-M. Yiu, *Reconstructing an ultrametric galled phylogenetic network from a distance matrix,* Journal of bioinformatics and computational biology **4**, 807 (2006).

[70] M. Hayamizu, K. T. Huber, V. Moulton, and Y. Murakami, *Recognizing and realizing cactus metrics,* Information Processing Letters , 105916 (2020).

[71] L. van Iersel, V. Moulton, and Y. Murakami, *Reconstructibility of unrooted level-k phylogenetic networks from distances,* Advances in Applied Mathematics **120**, 102075 (2020).

[72] M. Bordewich, C. Semple, and N. Tokac, *Constructing tree-child networks from distance matrices,* Algorithmica **80**, 2240 (2018).

[73] M. Bordewich, K. T. Huber, V. Moulton, and C. Semple, *Recovering normal networks from shortest inter-taxa distance information,* Journal of mathematical biology **77**, 571 (2018).

[74] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, and Y. Murakami, *Level-2 networks from shortest and longest distances,* Discrete Applied Mathematics **306**, 138 (2022).

[75] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[76] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A unifying characterization of tree-based networks and orchard networks using cherry covers,* Advances in Applied Mathematics **129**, 102222 (2021).

[77] J. Döcker, L. van Iersel, S. Kelk, and S. Linz, *Deciding the existence of a cherry-picking sequence is hard on two trees,* Discrete Applied Mathematics **260**, 131 (2019).

[78] S. Borst, L. van Iersel, M. Jones, and S. Kelk, *New fpt algorithms for finding the temporal hybridization number for sets of phylogenetic trees,* arXiv preprint arXiv:2007.13615 (2020).

[79] L. Zhang, *On tree-based phylogenetic networks,* Journal of Computational Biology **23**, 553 (2016).

[80] L. Jetten and L. van Iersel, *Nonbinary tree-based phylogenetic networks,* IEEE/ACM transactions on computational biology and bioinformatics **15**, 205 (2018).

[81] A. Francis, C. Semple, and M. Steel, *New characterisations of tree-based networks and proximity measures,* Advances in Applied Mathematics **93**, 93 (2018).

2

PRELIMINARIES

We start with elementary definitions from graph theory. A *graph* $G = (V, E)$ contains a set of vertices V and a set of edges $E \subseteq V \times V$. Depending on whether edges are directed or undirected, we call the graphs directed or undirected, respectively. *Directed acyclic graphs* are directed graphs with no directed cycles. Note that edge notation may sometimes differ in different chapters. In particular, an edge between two vertices u and v will be denoted (u, v) in Chapter 3, and uv in all chapters thereafter ((u, v) will carry a different meaning in Chapters 4 and 5). Let v be a vertex of a directed graph G. We say that v is a *cut-vertex* if deleting v from G results in a disconnected graph. We say that u is an *incoming neighbour* of v if uv is an edge in G. Conversely, we say that u is an *outgoing neighbour* of v if vu is an edge in G. We shall denote the set of incoming neighbours and the set of outgoing neighbours of v by $\Gamma^-(v)$ and $\Gamma^+(v)$, respectively. The *indegree* and *outdegree* of a vertex v are $|\Gamma^-(v)|$ and $|\Gamma^+(v)|$, respectively. The *degree* of a vertex is the sum of its indegree and outdegree. Let v be a vertex of an undirected graph H. We say that u is a *neighbour* of v if uv is an edge in H.

Algorithms take data as input and return an output solution. The performance of an algorithm is measured by its running time with regards to the input size. For an overview of polynomial-time algorithms and complexity theory in general, we refer the interested reader to [1].

We give all general definitions in this chapter, and define more specific terms in each of the respective chapters.

2.1. DIRECTED PHYLOGENETIC NETWORKS

In Chapters 3, 6 and 7, we consider only binary networks. In Chapters 4 and 5, we will always mention in the statement of a claim whether a network has to be binary, semi-binary, or non-binary, to make the assumptions on the degrees of the network vertices clear.

A few of the terms used to describe relationships between network vertices are biologically motivated. If there is a directed path from a vertex u to a vertex v, we say that u is an *ancestor* of v and that v is a *descendant* of u. We also say that u is *above* v and that v is *below* u. If the path consists only of a single edge, that is, uv is an edge, then u is a *parent* of v and v is a *child* of v. We also call u and v the *tail* and *head* of the edge uv, respectively. If the tail of an edge uv is the root of the network, then uv is called the *root edge*. We say that u is a *lowest common ancestor (LCA)* of two vertices x and y if u is above x and y, and no other vertices below u has this property. Within trees, the LCA of any two vertices is unique; this property does not hold in networks. Two vertices are *incomparable* if both vertices cannot be reached from the other via a directed path.

An important subgraph that will appear many times in this thesis is the *crown*. We say that a network contains a crown if there exists a set of vertices $\{u_1, \ldots, u_k, v_1, \ldots, v_k\}$ with edges $\{u_i v_i, u_i v_{i+1} : i \in [k]\}$ where $[k] = \{1, \ldots, k\}$, where the indices are taken modulo k (the network N_5 of Figure 2.1 on page 29 contains a crown on four vertices).

2.1.1. CHERRIES AND RETICULATED CHERRIES

Let (x, y) be an ordered pair of leaves of a network N, and let p_x, p_y denote the parents of x, y, respectively. We call (x, y) a *cherry* if $p_x = p_y$, that is, if x and y share a common

parent. Note that if (x, y) is a cherry, then (y, x) is also a cherry. We call (x, y) a *reticulated cherry* if p_x is a reticulation, p_y is a tree vertex, and p_y is a parent of p_x. If (x, y) is a cherry or a reticulated cherry in N, we say (x, y) is a *reducible pair*. See Figure 1.3 for an example of a cherry and a reticulated cherry.

2.1.2. SIMPLE GRAPH OPERATIONS

Many of the results presented in this thesis involve applying simple graph operations to our networks. We start with the notion of cleaning up a graph, which is generally applied after applying some graph operation. This is to ensure that the resulting graph remains a network. *Cleaning up* a directed acyclic graph is the act of applying the following operations until none is applicable:

1. delete an unlabelled outdegree-0 vertex;
2. *suppress* an indegree-1 outdegree-1 vertex (i.e., if uvw is a directed path where v is an indegree-1 outdegree-1 vertex, we *suppress* v by deleting the vertex v and adding an edge uw);
3. replace a pair of parallel edges by a single edge.

Observe that the above three operations do not give a way of dealing with indegree-0 and outdegree-2 vertices. In this thesis, we shall apply cleaning up after deleting reticulation edges – this means in particular that such vertices will not be created.

Edge Deletion *Deleting an edge* uv from a network N is the action of removing uv from N and cleaning up. Deleting a reticulation edge from a network and cleaning up always returns a network.

Vertex Deletion *Deleting a vertex* x from a network N is the action of removing x, deleting all its incident edges from N, and cleaning up.

Edge Contraction (for directed networks) Let u, v be adjacent vertices of a network N. Then *contracting the edge* uv is the action of deleting u and v, adding a vertex w, and adding edges xw for each $x \in \Gamma^-(u) \cup \Gamma^-(v) \setminus \{u\}$ and edges wy for each $y \in \Gamma^+(u) \setminus \{v\} \cup \Gamma^+(v)$.

We say that a path is *contracted* if every edge in the path is contracted, and *partially contracted* if some of the edges in the path are contracted.

Vertex Refinement (for directed networks) The reverse operation of edge contraction is *vertex refinement* (the term *splitting vertices* is also used in graph theory). Let N be a directed network, and let w be a tree vertex of degree at least 4 in N, and let p_w be the parent of w in N. Then *refining the tree vertex* w is the action of deleting w, adding tree vertices u, v, adding edges $p_w u, uv$, and adding edges (at least one) from u to some of the vertices in the set $\Gamma^+(w)$, and adding edges (at least one) from v to the rest of the vertices in $\Gamma^+(w)$. This increases the number of tree vertices by exactly one. Now let r be a reticulation of degree at least 4 in N with child c_r. Then *refining the reticulation* r is the

action of deleting r, adding reticulations s, t, adding edges st, tc_r, adding edges (at least one) from some of the vertices in the set $\Gamma^-(r)$ to s, and adding edges (at least one) from the rest of the vertices in $\Gamma^-(r)$ to t. This increases the number of reticulation vertices by exactly one. Observe that contracting the newly introduced edge upon refining a vertex returns the original network.

Cherry Reduction Let (x, y) be a cherry on the leaves x and y in a network N. *Reducing a cherry* (x, y) from N is the action of deleting the leaf x and cleaning up. Note that a cherry reduction from a network always returns a network.

Cutting a Reticulated Cherry / Reticulated Cherry Reduction Let (x, y) be a reticulated cherry on the leaves x and y in a network, where p_x, p_y are parents of x, y, respectively. *Cutting* (x, y) is the action of deleting the edge $p_x p_y$ and cleaning up. We also refer to this action as *reducing* the network by a reticulated cherry (x, y). Reticulated cherry reduction and cherry reduction will be grouped together to be called *reducible pair reduction*.

Isolating a Reticulated Cherry (only for binary networks) Let (x, y) be a reticulated cherry of a binary network. Let g_x denote the parent of p_x that is not p_y. *Isolating* (x, y) is the action of deleting the edge $g_x p_x$ and cleaning up. Note that the resulting network contains a cherry (x, y).

Adding vertices Let x be a leaf in a network N with a parent vertex p. *Adding a vertex q directly above x* is the action of deleting the edge px and adding two edges pq and qx.

Upon adding a vertex to a network, the resulting graph is no longer a network since there is a vertex of indegree-1 and outdegree-1. We note here that adding a vertex to a network is a technical operation that is always succeeded by another graph operation, in which we add an edge incident to the recently added indegree-1 and outdegree-1 vertex. While the intermediate graphs are sometimes not networks, this ensures that the resulting graph obtained from our graph operations is a network.

2.1.3. SUBNETWORKS AND CONTAINMENT

A network can be consistent with another network in many ways. The following defines precise notions of when a network *displays* another network as a *subnetwork*, or when a network *contains* another network. One is a generalization of the other: if a network is a subnetwork of another network, then it is contained by that network.

Let N and N' be non-binary networks on X and X', respectively. We say that we *clean up a DAG with respect to X'* if in addition to the normal cleaning up definition, we remove all outdegree-0 vertices that are not labelled by X'. We say that N' is a *subnetwork* of N if N' can be obtained from N by

1. deleting reticulation edges;

2. cleaning up with respect to X'.

Equivalently, N' is a subnetwork of N if there exists an embedding of N' into N: an injective mapping $f : V(N') \to V(N)$ of the vertices of N' to a subset of the vertices of N, where each leaf of N' is mapped to the leaf in N of the same label, and each edge uv in N' is mapped to a path from $f(u)$ to $f(v)$ in N. If there exists an embedding of N' into N, then we say that N' *can be embedded into* N.

Before we define *containment*, we require the following definition. Let N and N' be non-binary networks on X. Then N is a *refinement* of N' if N' can be obtained from N by contracting some edges. A non-binary network N *contains* another non-binary network N' if some refinement N'_b of N' is a subnetwork of N, i.e., if N' can be obtained from a subnetwork of N by contracting edges (see Figure 1.3).

The difference between being a subnetwork of a network and being contained in a network can be seen in Figure 4.8 on page 87. As seen, a subnetwork of a network can be defined by deleting reticulation edges and cleaning up, but also with embeddings. The equivalence of these two definitions has been shown for when the two networks are binary and on the same leaf-sets (Lemma 1 of [2]). It is easy to extend this equivalence to non-binary networks on different leaf-sets (in which one leaf-set is a subset of the other).

Lemma 1 ([3])**.** *Let N and N' be non-binary networks on X and $X' \subseteq X$. N' can be embedded into N if and only if N' can be obtained from N by deleting a set of reticulation edges and then cleaning up with respect to X'.*

Proof. First suppose there is an embedding of N' into N. Note that in this embedding, we may assume that the root of N' is mapped to the root of N. The image of this map (a subgraph of N) is a subdivision of N'. Let R be the set of reticulation edges of N not used in the embedding. We will show that N' can be obtained from N by removing R and cleaning up w.r.t X'. To show this, we prove that the edges that are removed in the clean-up are exactly the edges not used by the embedding of N' into N.

Let M be the network obtained from N by removing R and cleaning up w.r.t. X'. First note that no edges used by the embedding are removed in the process of removing R and cleaning up: indeed, for each such edge, there is a path to a leaf of X' using only edges of the embedding, which cannot be removed by cleaning up. Now suppose M has an edge that is not used in the embedding of N' into N. Consider a lowest such edge xy.

Node y cannot be a leaf of N, because all leaves of N are in the embedding of N' into N; or they are removed in the clean-up because they are not in X', in which case they cannot be part of M.

Now suppose that y is a tree node of N. It is impossible for an outgoing edge of y to be in the embedding, because the root of the embedding is the root of N. Hence, the outgoing edges of y are not in the embedding. At least one of these outgoing edges of y is in M, because, otherwise, y would have been deleted by cleaning up outdegree-0 nodes. Hence, at least one outgoing edge of y is in M but not in the embedding of N' into N, contradicting the assumption that xy is a lowest such edge.

Lastly, suppose that y is a reticulation. If none of the other incoming edges of the reticulation are in the embedding, it follows, similarly to the previous case, that the outgoing edge of y is in M but not in the embedding. This contradicts the assumption that xy is a lowest such edge. Hence, at least one incoming edge of y is used by the

embedding. This implies xy is an element of R, and has been deleted, contradicting the assumption that xy is an edge of M.

For the other direction, suppose N' can be obtained from N by removing a set of reticulation edges R and cleaning up. By reversing the operations used to clean up, we get an embedding of N' into N. Indeed, this only introduces reticulation edges not used by the embedding, and subdivides edges. When subdividing an edge used by the embedding, adapt the embedding accordingly, by mapping the edge of N' to the resulting path. □

Two networks N, N' on X are *isomorphic* if N is a subnetwork of N' and if N' is a subnetwork of N. Equivalently, N and N' are isomorphic if there exists a bijection f between the vertices of N and the vertices of N' such that uv is an edge of N if and only if $f(u)f(v)$ is an edge of N' and each leaf of N is mapped to a leaf of N' with the same label.

2.1.4. CLASSES OF DIRECTED PHYLOGENETIC NETWORKS

Phylogenetic networks are often categorized into different classes depending on their topological nature. While biological interpretations of such classes are offered, the reason for studying such classes are often computational. They are used as stepping stones to solve many NP-hard problems - by restricting the search space to particular network classes, the problem can sometimes become tractable, with some being solvable via an algorithm with a theoretical linear running time. See Figure 2.1.

STACK-FREE NETWORKS

A network is *stack-free* if no two reticulations are adjacent.

TREE-CHILD NETWORKS

A network is *tree-child* if every non-leaf vertex has a child that is either a tree vertex or a leaf [4]. This property inherently implies that every vertex in a tree-child network has a *tree path*, which is a path to a leaf having only tree-vertices as internal vertices. By definition, tree-child networks are stack-free.

Observation 1. *Every vertex in a tree-child network has a tree path to a leaf.*

Tree-child networks have gained some popularity within the mathematical phylogenetics field over the last decade. This is due to its convenient structure and nice properties, including the following.

Lemma 2 (Lemma 4.1 of [5])**.** *Tree-child networks on at least two leaves contain a reducible pair. The network obtained by reducing a cherry or a reticulated cherry in a tree-child network and cleaning up is tree-child.*

TREE-SIBLING NETWORKS

Let N be a network, and let u be a vertex of N. We say that v is a *sibling* of u if u and v share a common parent. A network is *tree-sibling* if every reticulation has a sibling that is a tree vertex or a leaf [6].

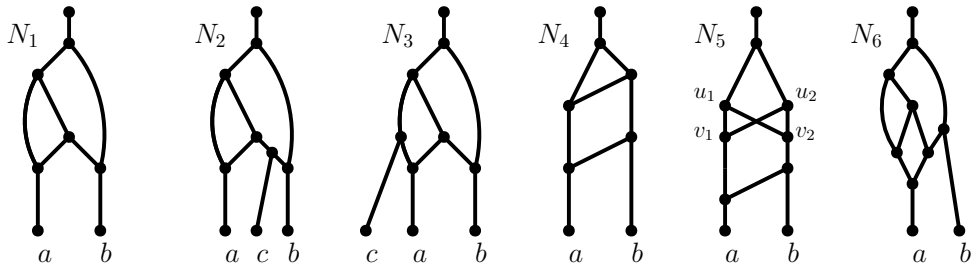


Figure 2.1: Six distinct networks N_i, $i \in [6]$ on either two or three taxa that are contained in different network classes. The networks N_5 and N_6 are level-3 networks; all other networks are level-2. The network N_1 is stack-free and tree-based, but not tree-child, valid, nor orchard. The network N_2 is stack-free, tree-child, valid, orchard, and tree-based. The network N_3 is stack-free, valid, orchard, and tree-based, but not tree-child. The network N_4 is orchard and tree-based, but not stack-free, tree-child, nor valid. The network N_5 is tree-based, but not stack-free, tree-child, valid, nor orchard. It contains a crown on the vertices $\{u_1, u_2, v_1, v_2\}$. The network N_6 is not stack-free, tree-child, valid, orchard, nor tree-based.

VALID NETWORKS

A reticulation edge deletion from a binary network is *valid* if the resulting subnetwork contains exactly 2 vertices and 3 edges fewer than the original network, i.e., only the reticulation edge is deleted and its endpoints suppressed. A reticulation edge deletion is *invalid* otherwise. Call a reticulation edge *valid* / *invalid* if its deletion is valid / invalid. A network is *valid* if all reticulation edges in the network are valid.

ORCHARD NETWORKS

A network is *orchard* if it can be reduced to a network on a single leaf by a sequence of cherry and reticulated cherry reductions. By Lemma 2 it is easy to see that tree-child networks are always orchard. We shall define orchard networks in more detail in Chapter 4.

Orchard networks are a relatively new class of networks within phylogenetics, introduced as *cherry-picking networks* by our paper [3] and also introduced independently by [7]. Throughout the thesis, we refer to this class of networks as orchard networks.

TREE-BASED NETWORKS

Tree-based networks were initially introduced and defined for binary networks by Francis and Steel [8]. This definition was extended for non-binary networks by Jetten and van Iersel [9] with two varying definitions: "tree-based" and "strictly tree-based". In this paper, we only focus on the "tree-based" definition.

A network N is *tree-based* with *base tree* T if N can be obtained from T via the following steps:

1. Replace some edges of T by paths, whose internal vertices are called *attachment points* of indegree-1 and outdegree-1.

2. Add edges, called *linking arcs*, between pairs of attachment points and from tree vertices to attachments points, so that N remains acyclic, attachment points have indegree or outdegree 1, and N has no parallel edges.

3. Clean up the acyclic directed graph.

The biological motivation for considering such networks hatched from an ongoing debate on whether or not evolutionary histories should be viewed as tree-like with reticulate events sprinkled in (e.g., in the context of horizontal gene transfer within prokaryotes [10]).

TEMPORAL NETWORKS

A network N is *temporal* or *time-consistent* if it admits a *temporal labelling* $t : V \to \mathbb{R}$ on its set of vertices V such that

1. For all tree edges uv, $t(u) < t(v)$.
2. For all reticulation edges uv, $t(u) = t(v)$.

Temporal networks are used to describe histories where both parents of reticulation vertices, as well as the reticulation vertex itself, are contemporaneous [11, 12]. Furthermore, speciation events always produce offsprings in a later generation. This has since been widely popular in defining network classes, for which results were obtained regarding metrics [13], applications for network class encodings (temporal tree-sibling) [13], and reduction of solution space in tackling the HYBRIDIZATION NUMBER problem (temporal tree-child) [14, 15].

HYBRIDIZATION NUMBER AND LEVEL-k NETWORKS

The *hybridization number* $r(N)$ of an acyclic directed graph N is the sum of all non-root vertex indegrees minus the number of vertices in N. Effectively, the reticulation number counts the number of hybridization edges that must be removed to obtain a tree, thereby acting as one measure of how much a network deviates from being a tree. Letting ρ denote the root of N, we have

$$r(N) = \sum_{v \in N \setminus \{\rho\}} (|\Gamma^-(v)| - 1)\,.$$

In a binary network, the hybridization number of a network is simply the number of reticulations in the network.

Another measure of non-treeness can be defined by a *level* of a network. A *biconnected component* refers to a maximal subgraph of a network with at least 3 vertices such that no vertex of the subgraph is a cut-vertex. The *level* of an acyclic directed graph is the maximum hybridization number across all biconnected components. A network is a *level-k network* if every biconnected component has hybridization number at most k, and at least one biconnected component has hybridization number exactly k.

2.2. UNDIRECTED PHYLOGENETIC NETWORKS

THE definitions listed hereafter are the undirected analogues of terms defined in the previous section. Given a non-empty set of extant taxa X, a *(non-binary) undirected (phylogenetic) network* on X is an undirected connected simple graph with

- unlabelled internal vertices of degree at least 3;

- leaves of degree-1 that are labelled bijectively by the elements of X.

A (non-binary undirected phylogenetic) *tree* on X is a network on X with no cycles.

Edges of undirected networks are undirected, which is the biggest distinction between directed and undirected networks. Because of this, undirected networks, although they can be seen as representations of evolutionary histories, are used more often as a data-display graph [16]. Undirected networks can be used to represent data that cannot be represented on a single tree due to conflicting signals. However, they can be used to represent evolutionary histories of which the location of the root and reticulations is unknown [17].

We call an undirected network *binary* if all internal vertices are of degree-3. We call it *non-binary* otherwise. We call an edge uv a *cut-edge* if deleting uv returns a graph on two components.

Due to the lack of a root and direction on the edges, internal vertices of undirected networks cannot be distinguished from one another; there is no distinction between tree-vertices and reticulation vertices as in the directed case. However, cycles within networks are formed as a result of non-treelike evolution being detected in the input data. Through this, we may analogously define the *level* of a network as a measure of how much a network deviates from being a tree. As before, a *biconnected component* is a biconnected maximal subgraph on at least three vertices. The *hybridization number* of an undirected network is the minimum number of edges that need to be removed from the graph to break all of its cycles. A network is of *level-k* if every biconnected component has hybridization number at most k, and at least one biconnected component has hybridization number exactly k.

2.2.1. CHERRIES AND CHAINS

Let N be an undirected network. A set of two leaves $\{x, y\}$ of N forms a *cherry* if they share a common neighbour. Let $a = (a_1, \ldots, a_k)$ be an ordered sequence of k leaves, and let $p_a = (v_1, \ldots, v_k)$, where v_i is the neighbour of a_i for each $i \in [k]$. We allow for $v_1 = v_2$ and $v_{k-1} = v_k$, in which case p_a contains only one of v_1, v_2 and / or v_{k-1}, v_k. If p_a is a path in N then a is called a *chain* of length $k \geq 0$. We call chains of length 0 empty chains. Assume that all chains are non-empty unless stated otherwise. Letting chains be of empty length is to generalize some statements later on in the thesis. We say that a is a *maximal chain* if a does not form a subsequence of some other chain. We assume all chains to be maximal, unless stated otherwise. If a is a chain, then the vertices of p_a are called the *spine vertices*, and the path p_a is called the *spine*. The vertices a_1, a_k are called the *end-leaves* of the chain a, and the vertices v_1, v_k are called the *end-spine vertices* of the chain. For brevity, given a set S, we shall write $S \cup a$ to denote the set $S \cup \{a_1, a_2, \ldots, a_k\}$.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms* (MIT press, 2009).

[2] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing*

tree-child networks from reticulate-edge-deleted subnetworks, Bulletin of mathematical biology **81**, 3823 (2019).

[3] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[4] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[5] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[6] L. Nakhleh, *Phylogenetic Networks,* Ph.D. thesis, The University of Texas at Austin (2004).

[7] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[8] A. R. Francis and M. Steel, *Which phylogenetic networks are merely trees with additional arcs?* Systematic biology **64**, 768 (2015).

[9] L. Jetten and L. van Iersel, *Nonbinary tree-based phylogenetic networks,* IEEE/ACM transactions on computational biology and bioinformatics **15**, 205 (2016).

[10] W. F. Martin, *Early evolution without a tree of life,* Biology direct **6**, 36 (2011).

[11] B. M. Moret, L. Nakhleh, T. Warnow, C. R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme, *Phylogenetic networks: modeling, reconstructibility, and accuracy,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **1**, 13 (2004).

[12] M. Baroni, C. Semple, and M. Steel, *Hybrids in real time,* Systematic biology **55**, 46 (2006).

[13] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, *A distance metric for a class of tree-sibling phylogenetic networks,* Bioinformatics **24**, 1481 (2008).

[14] P. J. Humphries, S. Linz, and C. Semple, *Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies,* Bulletin of mathematical biology **75**, 1879 (2013).

[15] S. Borst, L. van Iersel, M. Jones, and S. Kelk, *New fpt algorithms for finding the temporal hybridization number for sets of phylogenetic trees,* arXiv preprint arXiv:2007.13615 (2020).

[16] D. A. Morrison, *An introduction to phylogenetic networks* (RJR productions, 2011).

[17] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, and C. Semple, *Rooting for phylogenetic networks,* arXiv preprint arXiv:1906.07430 (2019).

3

RECONSTRUCTING LEVEL-k TREE-CHILD NETWORKS FROM RETICULATE EDGE-DELETED SUBNETWORKS

In this chapter, we show that directed level-k tree-child networks are encoded by their reticulate-edge-deleted subnetworks, which are subnetworks obtained by deleting a single reticulation edge, if $k \geq 2$. Following this, we provide a polynomial-time algorithm for uniquely reconstructing such networks from their reticulate-edge-deleted subnetworks. Moreover, we show that this can even be done when considering subnetworks obtained by deleting one reticulation edge from each biconnected component with k reticulations.

3.1. INTRODUCTION

It has been shown by Huber et al. in [2] that there exist networks which are not encoded by all subnetworks (called *subnets*) induced on proper subsets of the taxa. This is not to say that subnets do not encode many networks; in fact, it has been shown time and time again that considering topologically restricted classes of networks can help bypass this complication [3–6]. Two of the more prominent network classes are the *tree-child networks* [7] and the *level-k networks* [8].

In this chapter, we show that binary level-k tree-child networks, where $k \geq 2$ are encoded by *reticulate-edge-deleted subnetworks*, which are subnetworks obtained by deleting a single reticulation edge. In fact, we prove an even stronger result that this network class is encoded by its *Maximum Lower-Level Subnetworks (MLLSs)*, the subnetworks obtained by deleting a reticulation edge from every level-k biconnected component. We

The contents of this chapter have been published in Bulletin of Mathematical Biology **81**, 3823-3863 (2019) [1].

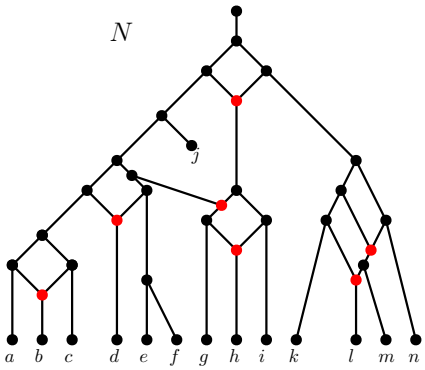


Figure 3.1: A level-4 tree-child network N on the set of species $X = \{a, \ldots, n\}$. Though N is a directed acyclic graph, the edge directions are omitted to avoid cluttering. The arcs are directed downwards. The leaf pair $\{e, f\}$ is a *cherry* since they share a common parent. The leaf pair $\{a, b\}$ is a *reticulated cherry* since the parent of a is also the parent of the parent of b.

do so by exploiting the fact that tree-child networks contain either a cherry or a reticulated cherry [9]. Cherries need not be reconstructed, since they stay intact in every MLLS; therefore we focus on reconstructing reticulated cherries and show that they are uniquely reconstructible through an exhaustive case study. In proving this result, we explore 'blob trees', an underlying tree of a network, introduced initially by Gusfield and Bansal [10]. These labelled trees are obtained from networks by collapsing every biconnected component to a single node, labelling the vertex by its set of leaf-descendants, and removing the leaves. In the case of valid networks, a class of networks more general than the tree-child networks, we show that we can reconstruct the blob tree of the original network from the blob trees of all MLLSs (Theorem 3).

The motivation behind studying reconstructibility results from MLLSs lies in the fact that not much is yet known regarding construction of networks of higher level. Indeed, deleting reticulation edges from each highest level blob ensures that the level of the resulting network is of lower level. We combine these lower level subnetworks to construct the original network of higher level. Until now, trees [11], level-1 networks [5, 12] and level-2 networks [4, 13] have garnered much attention in the reconstructibility literature. Our results from this chapter provide the first steps in the endeavour of finding higher level networks.

In related literature, it has been shown that tree-child networks are encoded by trinets [4] but not by trees (see Figure 3.2). Gambette et al. [5] showed that level-1 networks (which are necessarily tree-child) with girth (shortest cycle in underlying graph) at least 5 are reconstructible from their triplets. The triplets are phylogenetic trees on 3 leaves; as the set of triplets can be computed from the set of all displayed trees of the network, level-1 networks of girth at least 5 are encoded by trees and therefore by their MLLSs. Apart from this result by Gambette et al., reconstructing networks from MLLSs, or reticulation-edge-deleted subnetworks, has not been studied before.

The chapter is organized as follows. In the next section we define essential terms relevant to this chapter, including *MLLSs* and the notion of *encoding / reconstructibility*.

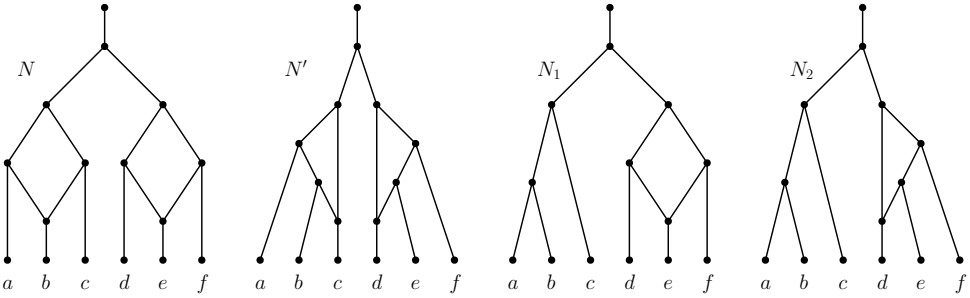


Figure 3.2: Networks N and N' are non-isomorphic but have the same lower-level subnetworks. Hence, any class containing N and N' is not level-reconstructible. However, these networks have different subnetworks: N_1 is a subnetwork of N but not of N'; N_2 is a subnetwork of N' but not of N. So $\{N, N'\}$ is subnetwork-reconstructible.

Section 3.3 presents the definitions and the key results on blob trees. In Section 3.4 we investigate the possible topologies for each leaf pair. Per our definition, there are 5 possibilities for each leaf pair up to isomorphism, and we develop a method for reconstructing a blob containing a particular leaf pair topology. In Section 3.5, we show our main result for this chapter, that binary level-$k \geq 2$ tree-child networks are reconstructible from their MLLSs (Theorem 6). A polynomial-time (in the size of the leaf set and the MLLS set) algorithm for reconstructing tree-child networks from their MLLSs follows naturally from our proof, and we present this in Section 3.6. In the last section, we conclude with some discussion of potential future directions.

3.2. PRELIMINARIES

In this chapter, we assume all networks to be directed and binary. Also in this chapter only, we let (u, v) denote an edge from u to v instead of writing uv. Let $\mathcal{N}(N)$ denote the set of all subnetworks on X, excluding N itself. A class $\mathscr{C}$ of networks is called *subnetwork-reconstructible* if for any two networks $N, N' \in \mathscr{C}$ with $\mathcal{N}(N) = \mathcal{N}(N')$, we have that N and N' are isomorphic. Recall that two networks N, N' on X are *isomorphic* if there exists a bijection f between the vertices of N and the vertices of N' such that (u, v) is an edge of N if and only if $(f(u), f(v))$ is an edge of N' and each leaf of N is mapped to a leaf of N' with the same label.

A related but subtly different notion is the following. Let N be a level-k network. Then $\mathcal{N}^{k-1}(N)$ denotes the set of subnetworks of N that is of level at most $k-1$. The networks in $\mathcal{N}^{k-1}(N)$ are called the *lower-level subnetworks* of N. Then a class $\mathscr{C}$ of networks is called *level-reconstructible* if for any two networks $N, N' \in \mathscr{C}$ of level-k and $\mathcal{N}^{k-1}(N) = \mathcal{N}^{k-1}(N')$, we have that N and N' are isomorphic. Note that if a network is level-reconstructible then it is subnetwork-reconstructible. The converse is not true in general, and an example of this is shown in Figure 3.2.

In this chapter we prove a result that is stronger than level-reconstructibility. Recall that a valid network is one where every reticulation edge deletion, upon suppressing degree-2 vertices, yields a network. An example of a valid reticulation edge is shown in

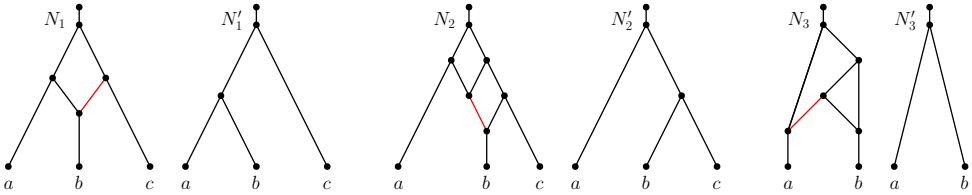


Figure 3.3: Three networks N_1, N_2, N_3 with their respective maximum subnetworks N_1', N_2', N_3' obtained by deleting the red reticulation edge and subsequently cleaning up. The red reticulation edge in N_1 is valid, however the red reticulation edges in N_2 and N_3 are invalid. The subnetwork N_2' contains 4 fewer vertices and 6 fewer edges than N_2, and N_3' contains 3 fewer vertices and 5 fewer edges than N_3.

Figure 3.3.

Lemma 3. *All reticulation edges in a tree-child network are valid.*

Proof. Let N be a tree-child network and suppose for a contradiction that deleting some reticulation edge $e = (u, v)$ is invalid. We note that v is a reticulation. As N is tree-child, we also have that u is a tree node. Therefore, after deleting e, u and v will each be indegree-1 outdegree-1 nodes, and will be suppressed by cleaning up. This removes a total of 2 vertices and 3 edges. Hence, to show that e is valid, it remains to show that no further cleaning up occurs after deleting e and suppressing u and v. As all remaining vertices have the same indegree-and outdegree-as before, there are no unlabelled outdegree 0 vertices and no remaining indegree-1 outdegree-1 nodes. So we just need to show that deleting e creates no parallel edges.

We split into three sub-cases. First assume that suppressing u results in the creation of parallel edges. Then we must have that u is contained in a 'triangle' with nodes x, y and edges $(x, u), (x, y), (u, y)$. But then y is a reticulation, implying that u is the parent of two reticulations y and v. Thus u has no child that is a tree vertex or a leaf, contradicting the tree-child property of N. Next assume that suppressing v results in the creation of parallel edges. Then we must have that v is contained in a triangle with nodes x, y and edges $(x, v), (x, y), (v, y)$. But then y is a reticulation, implying that v is the parent of a reticulation y. Thus v has no child that is a tree vertex or a leaf, contradicting the tree-child property of N. Finally assume that suppressing both u and v results in the creation of parallel edges. Then we must have that e formed the central edge of a 'diamond' with nodes x, y and edges $(x, u), (x, v), (u, v), (u, y), (v, y)$. However this cannot occur since the child of v, y, would be a reticulation, which again contradicts the tree-child property of N.

Therefore, every reticulation edge of a tree-child network is valid. □

The above lemma does not hold for general networks (see Figure 3.3). Intuitively, Lemma 3 states that removing any reticulation edge from a tree-child network is self-contained, and it does not affect any other reticulations within the network. No additional information is 'lost' when deleting valid reticulation edges. In particular, Lemma 3 implies that tree-child networks are valid.

From here onwards, it is implicitly assumed that the network N' obtained by deleting some reticulation edges from N undergoes cleaning up.

Definition 1. A *maximum subnetwork* of a network N is a subnetwork obtained by a single reticulation edge deletion from N.

Note in particular, that maximum subnetworks can be obtained both from valid and invalid reticulation edge deletions.

Lemma 4. *Every maximum subnetwork of a tree-child network is tree-child.*

Proof. Suppose that there exists a tree-child network N with a maximum subnetwork N' that is not tree-child. Then there exists a node t in N' such that all of its children are reticulations. Let (u,v) be the reticulation edge deleted from N to obtain N'. Since t has a tree vertex as a child in N, node u must be a child of t in N. Hence, (t,u) and (u,r) are edges in N, for some child r of t in N'. But then N is not tree-child as u is the parent of only reticulations v and r, a contradiction. □

Definition 2. A *maximum lower-level subnetwork (MLLS)* of a level-k network N is a subnetwork obtained by deleting exactly one valid reticulation edge from every level-k blob in N. Let $\mathscr{N}^{mlls}(N)$ denote the set of all MLLSs of N.

Observe that, as long as $\mathscr{N}^{mlls}(N)$ is a non-empty set, it is equal to the set of all subnetworks of N with level at most $k-1$ and a maximum number of edges.

By considering each reticulation edge deletion separately, it follows from Lemma 4 that the MLLSs of a tree-child network are tree-child.

A class $\mathscr{C}$ of networks is called *MLLS-reconstructible* if for any two networks $N, N' \in \mathscr{C}$ with $\mathscr{N}^{mlls}(N) = \mathscr{N}^{mlls}(N')$, we have that N and N' are isomorphic. Because all MLLSs are lower level subnetworks of N, we have $\mathscr{N}^{mlls}(N) \subseteq \mathscr{N}^{k-1}(N)$. Therefore, if a class of networks is MLLS-reconstructible, then it is level-reconstructible. The converse also holds for valid networks: the set of lower-level subnetworks of a valid network can be obtained by deleting further edges from the MLLSs. Therefore we have that if a class containing only valid networks is level-reconstructible, then it is also MLLS-reconstructible.

Observation 2. *Let $\mathscr{C}$ be a class of networks. If $\mathscr{C}$ is MLLS-reconstructible, then $\mathscr{C}$ is also level-reconstructible. If $\mathscr{C}$ is level-reconstructible, then $\mathscr{C}$ is also subnetwork-reconstructible.*

We will henceforth assume that all considered networks are binary tree-child networks on a non-empty set of taxa X, unless stated otherwise.

3.3. BLOB TREES

In this section we show how to reconstruct a *blob tree*, the underlying tree of a network. The tree has a similar construction as the 'blobbed trees' in [10] with further modifications.

Definition 3. The *blob tree* of a network N, denoted $BT(N)$, is the labelled tree obtained by applying the following:

1. contract every blob into a single node, and label each node, except for the root node, by the leaf-descendant set of the top vertex of the blob;

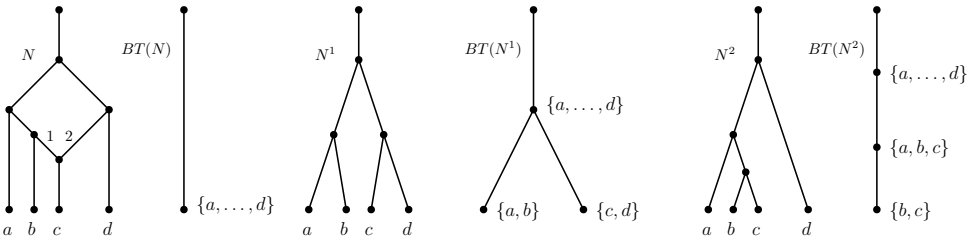


Figure 3.4: A tree-child network N, its maximum subnetworks N^1, N^2 obtained from deleting edges 1 and 2 respectively, together with their blob trees.

3

2. delete all leaf nodes.

We call the vertices in $BT(N)$ *blob nodes*.

An example of a blob tree is illustrated in Figure 3.4.

We refer to the top vertices of blobs as *pure nodes*. In the case of a level-0 blob, this top vertex is simply the tree vertex itself. Let x denote the pure vertex of some blob B of some network N. Then $desc_N(x) = desc_N(B)$ denotes the set of leaf-descendants of x (and thus B) in N.

For a general network N, it is possible for $BT(N)$ to contain two vertices with the same label if there is a blob in N of indegree-1 and outdegree-1. However the same cannot occur in tree-child networks, due to the following lemma.

Lemma 5. *Let N be a tree-child network on X, let $A \subseteq X$ and let x be a highest tree vertex with $desc_N(x) = A$. If a tree node $y \neq x$ also has $desc_N(y) = A$ then one child of x is a reticulation r such that y is below x and y is above r. Hence, x is the unique highest tree vertex with $desc_N(x) = A$ and all other tree nodes y with $desc_N(y) = A$ are in the same blob.*

Proof. Let $y \neq x$ be a tree vertex with $desc_N(y) = A$. To begin, note that y must be either above or below x. To see this, note that by the tree-child property of N, there exists a leaf l that is reached by x via a tree path. Then for y to be an ancestor of l, y must be either above or below x. Hence, x is the unique highest tree vertex with $desc_N(x) = A$ and y is below x.

By the tree-child property of N, either x can have two children that are tree vertices or leaves, or x can have one tree vertex or leaf child and one reticulation child. Let c_1, c_2 denote the children of x, and by the tree-child property of N, there exist leaves l_1, l_2 that are reached by c_1, c_2 via tree paths respectively.

First suppose that the two children c_1, c_2 of x are tree vertices or leaves. Then for y to be an ancestor of both l_1 and l_2, y must be an ancestor of both c_1 and c_2, contradicting that $y \neq x$ is below x.

Hence, one of the two children of x, is a reticulation r. Without loss of generality, $r = c_1$. It remains to show that y is above c_1. Since y is an ancestor of l_1, and there is a tree path from c_1 to l_1, node y is either above or below c_1. Suppose for contradiction that y is below c_1. Since y is also an ancestor of l_2, there exists a directed path from y to l_2. This path must pass through x since the path from x to l_2 is a tree path. This is a directed path

from y to x. However, since there is also a directed path from x to y (via c_1), and $y \neq x$, it follows that there exists a directed cycle, a contradiction. □

The following corollary follows immediately from Lemma 5.

Corollary 1. *Let N be a tree-child network. Then its blob tree $BT(N)$ contains vertices with unique labels.*

Due to this, we identify blob vertices by their vertex labels, e.g., for a blob B in N with $desc_N(B) = A$, the corresponding blob vertex in $BT(N)$ is A.

3.3.1. ON RETICULATED CHERRIES

In a phylogenetic network N, non-reticulation nodes $\{x, y\}$ form a *reticulated cherry shape* if p_x, p_y are parents of x, y respectively, where p_y is a reticulation and p_x is a parent of p_y (see Figure 3.5 (a)). In this case we say that the reticulation is on y and that the reticulation p_y is in the reticulated cherry shape $\{x, y\}$. This notion is a generalization of the reticulated cherries defined by Bordewich et al. in [14], in which both x and y are leaves.

Lemma 6. *In a tree-child network, all reticulations are in a* reticulated cherry shape. *Moreover, for $k \geq 1$, there is at least one reticulation in each level-k blob that is in a reticulated cherry shape formed by two vertices outside of the blob.*

Proof. Let N be a tree-child network and consider a reticulation r in an arbitrarily chosen blob B. By the tree-child property, r must have a non-reticulation child y and two tree vertex parents t_1, t_2. The child of t_1 that is not r must be a non-reticulation x. Then r is in a reticulated cherry shape formed by $\{x, y\}$.

Now consider a lowest tree node a in B. If both children of a were to be non-reticulations then at least one of the children would also be contained in B, contradicting our choice of a. If both children of a were to be reticulations then the network would no longer be tree-child, a contradiction. Thus, one child of a is a reticulation, say c, and the other a non-reticulation, say x. The child of c, say y, must be a non-reticulation as the network is tree-child, and thus B contains a reticulated cherry shape formed by two nodes x, y. Moreover, x and y are outside of B because they are either leaves or tree nodes, and below a lowest tree vertex in B. □

A reticulated cherry shape $\{x, y\}$ is called a *lowest reticulated cherry shape of a blob B*, if the parent p_x of x is a lowest tree vertex of B. This implies that x and y are not contained in B, as shown in the proof of Lemma 6.

Let N' be a maximum subnetwork of a tree-child network N obtained by isolating a lowest reticulated cherry shape of a blob B. Then there is a pure vertex in N' that is not a pure vertex in N (Figure 3.5). Moreover, if blob B is of level at least 2, the leaf-descendant set of the new pure vertex is not equal to the leaf-descendant set of any vertex in N. This leads to the following observation.

Observation 3. *For a tree-child network N and B a level-k blob, with $k \geq 2$, there is always a reticulation edge we can delete from B such that the blob tree of the resulting subnetwork is not equal to $BT(N)$.*

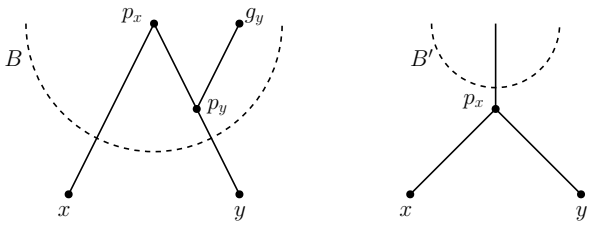


Figure 3.5: (a) A portion of the network showing a lowest reticulated cherry shape in a blob B. (b) The same portion of the network after isolating the reticulate cherry shape $\{x, y\}$. Note here that p_x is a pure vertex in the subnetwork, but p_x in not a pure vertex in the original network.

Now suppose x and y are both leaves. If x and y share a common parent, then they form a *cherry*. If $\{x, y\}$ forms a reticulated cherry shape then it is a *reticulated cherry*. The following Lemma from Bordewich and Semple [9] is essential for our results and will be used extensively throughout the text.

Lemma 7 ([9])**.** *If N is a tree-child network on at least two leaves, then N contains either a cherry or a reticulated cherry.*

3.3.2. RECONSTRUCTING THE BLOB TREE OF A TREE-CHILD NETWORK

Lemma 8. *For a valid network N, if the blob tree $BT(N)$ contains a blob node A then $BT(N')$ contains the blob node A for all maximum subnetworks N' of N.*

Proof. First suppose that A is a blob vertex corresponding to a level-0 blob in N. The corresponding node t in N is not incident to any reticulation edges, so it is not possible to suppress t via a reticulation edge deletion. Note that a reticulation edge deletion from a blob above or below t would not change the leaf-descendant set of t. Hence t remains a level-0 blob in all maximum subnetworks of N with leaf-descendant set A. Thus A is a blob vertex in all $BT(N')$ for all maximum subnetworks N' of N.

Now suppose that A is a blob vertex corresponding to a blob of level at least 1. Suppose t is the corresponding pure vertex in N. If t is not incident to a reticulation edge then there is no way of suppressing t by means of edge deletions and any reticulation edge deletion will not change the leaf-descendant set of t. Hence, t is a pure vertex with leaf-descendant set A in all maximum subnetworks of N. If on the other hand there is a reticulation r with edges $(t, r), (s, r)$ then let c be the child of t that is a tree vertex (it is possible that $c = s$). Because t is the top vertex of the blob, there is a directed path from t to s, which must include c. Hence there is a directed path from c to s and to r. Therefore, we have $desc_N(r) \subseteq desc_N(c)$. So $desc_N(c) = desc_N(c) \cup desc_N(r) = desc_N(t) = A$. We now use the fact that, after a valid edge deletion, only the endpoints of the edge are suppressed in the resultant maximum subnetwork. The maximum subnetwork where (t, r) is deleted contains c as a pure node, and hence A is a blob vertex in its blob tree. The maximum subnetwork where (s, r) is deleted contains t as a pure node, and hence A is a blob vertex in its blob tree. The maximum subnetwork where some other reticulation edge is deleted contains t as a pure node, and hence A is a blob vertex in its blob tree.

Thus A is a blob vertex in $BT(N')$ for all maximum subnetworks N' of N.

□

Lemma 9. *For a valid network N, if $BT(N')$ contains a blob node A for all maximum subnetworks N' of N then $BT(N)$ also contains the blob node A.*

Proof. Consider some lowest reticulation r in N such that r is the ancestor of some $a \in A$. Let c be the child of r in N. Since r is of outdegree-1, we have $desc_N(r) = desc_N(c)$. We may assume $desc_N(r) \neq A$, as otherwise c is the root of a pendant subtree spanning A in N, and consequently A is a blob vertex in $BT(N)$. Let $(u,r),(v,r)$ be the edges leading into r. Let N', N'' be the maximum subnetworks of N obtained by deleting $(u,r),(v,r)$ respectively. Note here that every node x in N' or N'' is also a vertex in N. We now examine the relations between $desc_N(r)$ and A exhaustively.

- Suppose $desc_N(r) \not\subset A$ and $A \not\subset desc_N(r)$. We show that there is no vertex in N' that has leaf-descendant set A. By assumption, there exists a node $a' \in desc_N(r)$ such that $a' \notin A$. Then, $a' \in desc_{N'}(c)$. Let x be a vertex in N' (which is also a vertex in N). We examine the relations between x and c in N' exhaustively.
  - If x is an ancestor of c in N' then $desc_{N'}(x) \neq A$ since $a' \in desc_{N'}(c) \subseteq desc_{N'}(x)$.
  - If x is a descendant of c in N' then $desc_{N'}(x) \neq A$ since $A \not\subset desc_{N'}(c)$.
  - If x is incomparable to c in N' then $desc_{N'}(x) \neq A$ since $a \notin desc_{N'}(x)$ by assumption that r was the lowest reticulation above a.

  It follows that A is not in $BT(N')$, and this case is not possible. The only possibilities then, are either $A \subsetneq desc_N(r)$ or $desc_N(r) \subsetneq A$.

By assumption, $BT(N')$ and $BT(N'')$ both contain A. Because of this, there are corresponding pure nodes x', x'' in N', N'' (also in N) respectively with $desc_{N'}(x') = desc_{N''}(x'') = A$.

- Suppose $A \subsetneq desc_N(r)$. Then x' must be a descendant of c in N', implying that x' must be a descendant of r in N. We claim that x' is a pure vertex in N with $desc_N(x') = A$. If x' is not a pure vertex in N then there exists a reticulation $s \neq r$ below x' where s and x' are contained in the same blob in which x' is not the top-node, in N. The edge deletion does not suppress or delete the node s, since s is a descendant of r, and any directed path from r to s is of length at least 2. Then, s is a reticulation that is below r such that the leaf-descendant set of s contains an element of A. This contradicts our choice of r, so x' must be a pure vertex in N. Furthermore, we must have $desc_N(x') = desc_{N'}(x') = A$ where the first equality holds as deleting a reticulation edge from above a vertex does not change its leaf-descendant set in the resultant subnetwork. Then x' must be a pure vertex in N with $desc_N(x') = A$ and we are done.
- So we may assume $desc_N(r) \subsetneq A$. We now claim that $desc_N(v) \subseteq A$. Suppose not. Noting that v is not suppressed in N' (since N is a valid network), and since $desc_{N'}(v) = desc_N(v)$, we split into the three possible cases for the relation between x' and v in N'.

- If x' is an ancestor of v then it is also an ancestor of $b \notin A$ in N' for some $b \in desc_{N'}(v)$, a contradiction.
- If x' is incomparable to v then x' is also incomparable to c in N'. Then, since $a \in A$ is a leaf-descendant of x' in N', there is a reticulation s below r in N such that s is an ancestor of a, which contradicts our choice of r.
- If x' is a descendant of v then it must either be incomparable to or be a descendant of c in N'.
  - If x' is incomparable to c in N', then we reach a contradiction by the same argument as above.
  - If x' is a descendant of c in N', then as $desc_{N'}(c) \subsetneq A$ (since $desc_N(r) = desc_{N'}(c)$) we have that $desc_{N'}(x') \subsetneq A$, a contradiction.

Thus we have that $desc_N(v) \subseteq A$. By an analogous reasoning on x'' in N'', we have that $desc_N(u) \subseteq A$. It follows that x' must be an ancestor of v in N', and so x' must be an ancestor of v in N. It also follows that x' must be an ancestor of u in N to ensure that there is a path from x' to the leaf-descendants of u in N'.

We now claim that x' is also a pure vertex in N with leaf-descendant set A. Indeed, adding the edge (u, r) to N' (after undoing any cleaning up) only joins descendants of x', implying x' has leaf-descendant set A in N. Furthermore, it cannot add any vertices that are not descended from x' to the blob containing x'. It follows that x' remains a pure vertex in N with leaf-descendant set A.

□

By combining the previous two lemmas, we see that the blob trees of valid networks are reconstructible from their maximum subnetworks.

Theorem 1. *For a valid network N, given a set $A \subseteq X$, the blob tree $BT(N)$ contains a blob node A if and only if $BT(N')$ contains a blob node A for all maximum subnetworks N' of N.*

Proof. Follows from Lemmas 8 and 9. □

We can prove a similar result for MLLSs.

Theorem 2. *Let N be a level-k valid network, with $k \geq 2$. Given a set $A \subseteq X$, the blob tree $BT(N)$ contains the blob node A if and only if $BT(N^{mlls})$ contains A for all MLLSs N^{mlls} of N.*

Proof. Suppose first that the blob tree $BT(N)$ contains the node A, and let N^{mlls} be an MLLS of N obtained by deleting the edges in the set $E = \{e_1, \ldots, e_m\}$. Consider the maximum subnetwork N' of N obtained by deleting the reticulation edge e_1. By Theorem 1, $BT(N')$ contains the blob node A. Now consider the maximum subnetwork N'' of N' obtained by deleting the reticulation edge e_2. Then $BT(N'')$ contains the blob node A by Theorem 1. Continuing in this fashion for all edges in E shows that $BT(N^{mlls})$ contains the blob node A.

Now suppose that A is not a blob vertex of $BT(N)$. We prove the contrapositive that there exists an MLLS N^{mlls} of N such that $BT(N^{mlls})$ does not contain the blob node A. Since A is not a blob vertex of $BT(N)$, there exist blob nodes C and D such that $C \subseteq A \subseteq D$. Let B denote the blob in N with leaf-descendant set D. Consequently, if there exists a pure vertex in an MLLS N^{mlls} of N with leaf-descendant set A, then it must be a vertex that was originally in the blob B. Now observe that deleting reticulation edges from blobs that are not B do not affect the leaf-descendant set of all vertices in B. Then we may assume, without loss of generality, that N is a single blob network. But then by Theorem 1, we have that A is not a blob vertex in $BT(N^{mlls})$, for all MLLSs N^{mlls} of N. □

We call a set $A \subseteq X$ a *foundation node* of N if $BT(N)$ contains the node A. Let $\mathscr{F}(N)$ be the set of all foundation vertices of N.

Theorem 3. *For a level-k valid network N, with $k \geq 1$, its blob tree $BT(N)$ is reconstructible from its MLLSs.*

Proof. By Theorem 2, the set of all foundation nodes $\mathscr{F}(N)$ consists of the blob vertices that appear in every MLLS blob tree. Then, the blob tree $BT(N)$ is the tree with vertex set $\mathscr{F}(N)$ and an edge (A, B) precisely if $B \subsetneq A$ and there is no $C \in \mathscr{F}(N)$ with $B \subsetneq C \subsetneq A$. □

3.3.3. MINIMUM NUMBER OF MLLSS TO RECONSTRUCT THE BLOB TREE OF A TREE-CHILD NETWORK

We consider the minimum number of MLLSs required to reconstruct the blob tree of a tree-child network. Let r be some reticulation in a blob B. We call the node s a *pseudo pure vertex of r* if it is the lowest vertex in B such that there are two edge disjoint directed paths from s to r.

Lemma 10. *Let N be a level-k tree-child network where $k \geq 1$. Two maximum subnetworks N' and N'' of N suffice to reconstruct $BT(N)$.*

Proof. Let r be a lowest reticulation in some blob B. Let $\{x, y\}$ denote a reticulated cherry shape corresponding to r. Let p_x and $p_y = r$ be the parents of x and y, respectively, and let g_y be the parent of p_y that is not p_x. Let N' and N'' be the maximum subnetworks of N derived by cutting and isolating $\{x, y\}$ respectively. Let P', P'' denote the set of all pure vertices in N', N'' respectively, that are not pure vertices in N. We claim that the intersection of P' and P'' is empty, from which it follows that the intersection of the vertex sets of $BT(N')$ and $BT(N'')$ contains the foundation vertices of N. Since by Lemma 8 each foundation vertex of N is a foundation vertex of each maximum subnetwork, it follows that the intersection of $BT(N')$ and $BT(N'')$ is precisely the set of foundation vertices of N.

Claim 1. *Let $p \in P'$ ($p \in P''$). Then p is an ancestor of r in N.*

Proof. Suppose not. First suppose that p is a descendant of r in N. As r is a lowest reticulation in N, p is a tree vertex or a leaf in N' (N''). If p is a tree vertex then p must have

been a pure vertex in N to begin with: the pendant subnetwork rooted at the child of r is an invariant upon obtaining maximum subnetworks of N, since r is a lowest reticulation. This contradicts the fact that p is an element of P' (P''). If p is a leaf then p cannot be a pure vertex in N' (N''), a contradiction.

Now suppose that p is incomparable to r in N. Let p be the pure vertex of a blob B' in N' (B'' in N''). As p is not an ancestor of r in N, p must also not be an ancestor of p_x nor g_y in N. We see that B' (B'') remains a blob after adding the edge (p_x, r) to N' ((g_y, r) to N''), and so p remains a pure vertex in N, a contradiction. □

3

Claim 2. *Let $p \in P'$ ($p \in P''$). Then p is a descendant of the pseudo pure node s of r in N and $p \neq s$.*

Proof 1: Suppose not.

If p is equal to or strictly above s then p is an ancestor of both p_x and g_y. Adding the edge (p_x, r) to N' ((g_y, r) to N'') only joins descendants of p. Furthermore, it cannot add any vertices that are not descended from p to the blob in N' (N'') containing p. It follows that p remains a pure vertex in N, a contradiction.

Now suppose that p is incomparable to s. If p is not in the blob B, then as reticulation edge deletions do not affect other blobs, we have that p must have been a pure vertex in N. This contradicts our assumption on p. So p must be in the blob B. Since p is incomparable to s, but p must still be an ancestor of r by Claim 1, p must be an ancestor of a reticulation r' such that s is an ancestor of r' and r' is an ancestor of r. The parent of s, denoted p_s, is not suppressed in both N' and N''. Now p_s is either above or incomparable to p, and the two vertices belong to the blob which contains r' in N' (N''). It follows that p cannot be a pure vertex in N' (N''), which contradicts our assumption. ■

It remains to show that $P' \cap P'' = \emptyset$. Let $P'_x = \{p \in P' : p \text{ is an ancestor of } p_x \text{ in } N\}$ and let $P'_y = \{p \in P' : p \text{ is an ancestor of } g_y \text{ in } N\}$. By Claim 1, we have $P' = P'_x \cup P'_y$. By Claim 2, we have $P'_x \cap P'_y = \emptyset$. Define P''_x and P''_y analogously. Let $a' \in P'_x$, $a'' \in P''_x$ and $b'' \in P''_y$. Let $u \in desc_N(x)$ and let $v \in desc_N(y)$. Clearly, $u \in desc_{N'}(a')$ and $u, v \in desc_{N''}(a'')$. By Claim 2, $v \notin desc_{N'}(a')$ and $u, v \notin desc_{N''}(b'')$. We now see that $a' \neq a''$ because one has leaf v as descendant and the other one not. So $P'_x \cap P''_x = \emptyset$. Similarly, $a' \neq b''$ because one has leaf u as descendant and the other one not. So $P'_x \cap P''_y = \emptyset$. It follows that P'_x and P'' are disjoint. By an analogous argument, P'_y and P'' are disjoint. Therefore P' and P'' are disjoint. □

Let N be a network and let N_A be a pendant subnetwork of N rooted by a vertex with leaf-descendant set A. *Collapsing* N_A from N means that we replace N_A by a leaf A. Let $N \setminus N_A$ denote the network obtained by collapsing N_A from N.

Lemma 11. *Let N be a tree-child network, and let N_A denote a pendant subnetwork of N rooted at a vertex with leaf-descendant set A. Then $BT(N \setminus N_A)$ is obtained from $BT(N)$ by deleting the pendant subtree rooted at A.*

Proof. By definition, there exists a blob node A in $BT(N)$. Note that pendant subnetworks of N uniquely correspond to a pendant subtree of $BT(N)$, by definition of blob trees and also because vertex labels of blob trees are unique for tree-child networks

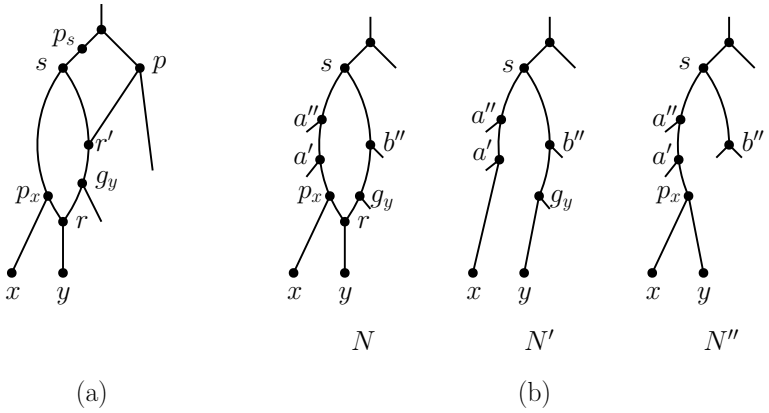


Figure 3.6: Visual aid for Lemma 10 proof. (a) Proof of Claim 2. Cutting or isolating $\{x, y\}$ results in subnetworks where p_s and p lie in the blob containing r'. (b) Proof of the paragraph after Claim 2, which shows that $P' \cap P'' = \emptyset$.

(Corollary 1). Then the pendant subtree of $BT(N)$ rooted at A is uniquely defined by N_A and vice versa: this implies the lemma. □

Lemma 12. *Let N be a level-k tree-child network with $k \geq 2$. Two MLLSs N_1^{mlls} and N_2^{mlls} of N suffice to reconstruct $BT(N)$. In particular, N_1^{mlls} is the MLLS obtained by cutting a lowest reticulated cherry shape in every level-k blob, and N_2^{mlls} is the MLLS obtained by isolating these reticulated cherry shapes. That is, the vertices of the blob tree of N correspond to the vertices that are a vertex of each blob tree of these two MLLSs.*

Proof. We prove the lemma by induction on the number of level-k blobs l in N. For the base case, there is only one level-k blob in N. By Lemma 10, we are done.

So suppose now that N contains $l \geq 2$ level-k blobs. Consider a lowest level-k blob B in N, and let A denote the leaf-descendant set of B. Let N_A and N_{iA}^{mlls} denote the pendant subnetwork of N and N_i^{mlls} rooted at the pure vertex with leaf-descendant set A, for $i = 1,2$. By Theorem 2, A is a blob vertex in $BT(N_i^{mlls})$ for $i = 1,2$, and therefore such pendant subnetworks exist. Note that the pendant subnetworks N_{iA}^{mlls} are maximum subnetworks of N_A obtained by cutting and isolating the reticulated cherry shape associated with some lowest reticulation r. By Lemma 10, we have that N_{1A}^{mlls} and N_{2A}^{mlls} suffice to reconstruct $BT(N_A)$. We now collapse N_{iA}^{mlls} from the MLLS N_i^{mlls} for $i = 1,2$. Furthermore we collapse N_A from the network N. Note that $N \setminus N_A$ is a level-k tree-child network with $l-1$ level-k blobs, and that $N_i^{mlls} \setminus N_{iA}^{mlls}$ are MLLSs of $N \setminus N_A$ obtained by cutting and isolating a lowest reticulated cherry shape from every level-k blob, for $i = 1,2$ respectively. By the induction hypothesis, these two MLLSs of $N \setminus N_A$ suffice to reconstruct $BT(N \setminus N_A)$. Now by Lemma 11, we have that $BT(N)$ is the blob tree obtained by appending $BT(N_A)$ to $BT(N \setminus N_A)$. We append $BT(N_A)$ to the node C in $BT(N \setminus N_A)$, such that $A \subseteq C$, and there exists no node $D \in BT(N \setminus N_A)$ where $A \subseteq D \subseteq C$. □

3.3.4. IDENTIFYING THE LEVEL-k BLOBS OF A TREE-CHILD NETWORK

We now show that given the MLLSs, it is possible to identify which foundation vertices correspond to a level-k blob in the original tree-child network. A *pendant subnetwork* of a network N is obtained by deleting a cut-edge (x, y) from N and taking the connected component containing y. A *pendant subtree* is a pendant subnetwork that is a tree.

Lemma 13. *Let N be a level-k tree child network with $k \geq 2$. A blob of N is level-$k' < k$ if and only if the set of outgoing neighbors of the corresponding blob vertex in $BT(N^{mlls})$, for every $N^{mlls} \in \mathcal{N}^{mlls}(N)$, is precisely the set of outgoing neighbors of the blob vertex in $BT(N)$.*

Proof. Suppose B is a level-$k' < k$ blob in N. Then B remains intact (no reticulation edges deleted) in all MLLSs of N. Let B' be a blob in N that is directly below B, and let e denote the outgoing edge from B to the pure vertex of B'. The edge e is not suppressed in any MLLS of N. And since edges are deleted to obtain MLLSs of N, we have that the number of leaves that are below the edge e (below the child of e) stays the same since blobs are biconnected. By Theorem 2, every vertex in $BT(N)$ is a vertex in $BT(N^{mlls})$ for all MLLSs N^{mlls} of N. Furthermore for tree-child networks, the vertex labels in blob trees are unique. Then the blob vertex of B must have the blob vertex of B' as one of its outgoing neighbors in the blob tree of all MLLSs. Since B' was chosen arbitrarily, this implies that the outgoing neighbors of the blob vertex in $BT(N^{mlls})$, for every $N^{mlls} \in \mathcal{N}^{mlls}(N)$, is precisely the outgoing neighbors of the blob vertex in $BT(N)$.

For the other direction, we prove the contrapositive. Suppose B is a level-k blob in N, and let $desc_N(B) = A$. By Observation 3, we can isolate a lowest reticulated cherry in B to obtain an MLLS N^{mlls} of N where $BT(N^{mlls})$ is different from $BT(N)$. In this construction of N^{mlls}, there exists a pure vertex in N^{mlls} which was not a pure vertex in N. Then the outgoing neighbors of A in $BT(N^{mlls})$ is not the same as the outgoing neighbors of A in $BT(N)$. □

Figure 3.7 illustrates Lemma 13 with a level-4 tree-child network N. The blob trees of its MLLSs are taken, from which the blob tree of N can be reconstructed (Theorem 2). Then, it can be seen that the set of outgoing neighbors of the blob node $\{a, \ldots, n\}$ in $BT(N_i)$ for $i = 1, 3, 5, 6, 7, 8$ differs from the the set of outgoing neighbors of $\{a, \ldots, n\}$ in $BT(N)$. Hence, the blob with leaf-descendant set $\{a, \ldots, n\}$ is of level-4. Since the outgoing neighbors of the other blob vertices do not change, the blobs with leaf-descendant sets $\{a, b, c\}$, $\{e, f\}$ and $\{k, \ldots, n\}$ are blobs of level lower than 4.

3.4. LEAF PAIR ANALYSIS

IN order to reconstruct a tree-child network from its MLLSs, we require a way of locating the position of the missing reticulation edges. In this section, we show that studying the topology of a leaf pair in the MLLSs gives enough information to infer the topology of those same leaves in the original network. The next section will show how we can use this to find the location of the missing reticulation edge of each blob by choosing the appropriate leaf pair.

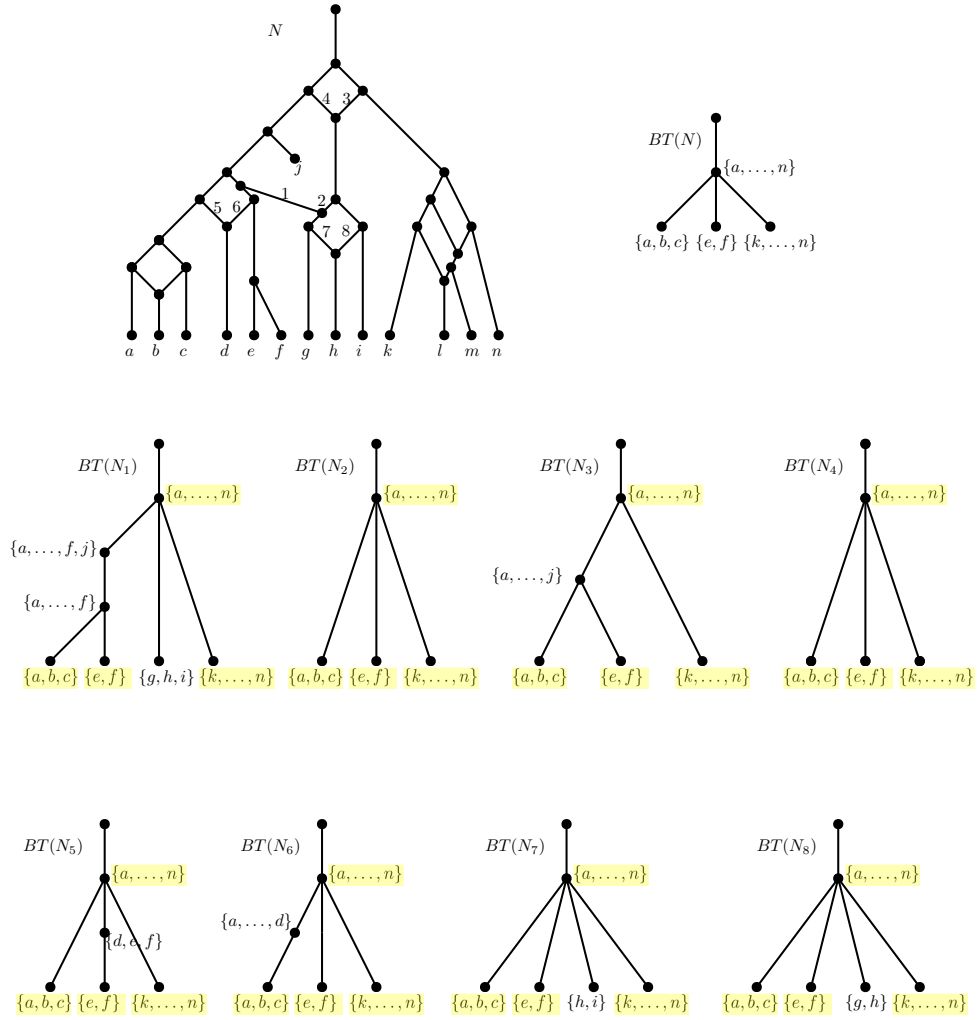


Figure 3.7: N_i for $i = 1,2,\ldots,8$ refers to the MLLS of a level-4 tree-child network N obtained by deleting the reticulation edge i. $BT(N)$ is the blob tree of the network N and $BT(N_i)$ is the blob tree of N_i for each i. The foundation vertices are highlighted in yellow.

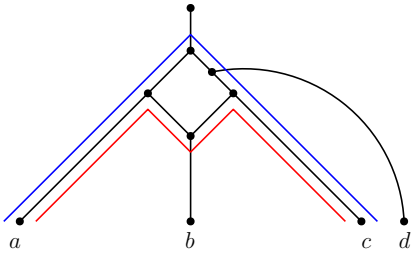


Figure 3.8: A network N on 4 leaves. The shortest ac up-down distance is 5 (blue path) however, the shortest ac distance in the underlying undirected graph of N is 4 (red path).

We use the inter-vertex distance as defined by Bordewich and Semple [9]. For our purposes, we slightly tweak the definition by allowing the endpoints to be non-leaf nodes.

Definition 4. Let N be a network and let $x, y \in N$. An *up-down path* of length p from x to y is a sequence of nodes $x = v_0, v_1, v_2, \ldots, v_{p-1}, v_p = y$ in N, such that for some $0 \leq i \leq p$, N contains the edges

$$(v_i, v_{i-1}), \ldots, (v_1, x)$$

and

$$(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{p-1}, y).$$

The node v_i is the *apex* of this up-down path. The length of a shortest xy up-down path P in N is denoted $d_N(x, y)$. If P is an up-down path in N and N^{mlls} an MLLS of N, $P_{N^{mlls}}$ is the xy up-down path P restricted on the vertex and edge sets of N^{mlls}, given it is still an up-down path in N^{mlls}.

Note that the shortest up-down distance $d_N(x, y)$ in a network N may not necessarily be the shortest distance in the underlying undirected graph of N (where the underlying undirected graph of N is obtained by replacing every directed edge by an undirected edge), see Figure 3.8.

Let Q be an up-down path between nodes u and v of length at least 2 in a tree-child network N. An edge (u, v), if it exists, is called a *shortcut*. In some papers, the notion of a shortcut (also known as a redundant arc) is defined on directed paths rather than on up-down paths [14, 15]. For the purposes of this paper and since a directed path is by definition an up-down path (without the 'up' portion), we define shortcuts on the up-down paths. Call an up-down path which has no shortcuts in N a *shortcut free* up-down path. Note that shortest up-down paths are necessarily shortcut free. Let N' be a maximum subnetwork of N obtained by deleting some reticulation edge (u, r). Let P' be an xy up-down path in N' for nodes x, y. Reinsert the edge (u, r) in N'. Then the xy up-down path P', together with any vertices in $\{u, r\}$ that bisect some edge of P' is called the *embedded path* of P' in N.

Lemma 14. *In a tree-child network, deleting a single reticulation edge can reduce the up-down distance between any two leaves by at most one.*

Proof. Let N be a tree-child network and let N' be a maximum subnetwork of N obtained by deleting some reticulation edge (u, r). Let v be the parent of r in N that is

not u. Take any xy up-down path P' in N', and let P be its embedded path in N. Let P^* be an up-down path in N derived from P by taking the shortcut (u,r) if it is a shortcut in P. We show that $|P^*| \leq |P'|+1$. Now compared to P', the up-down path P contains at most 2 additional vertices - the nodes u and r. If it contains:

- 0 additional vertices then (u,r) cannot be a shortcut of the embedded path P. So, $|P| = |P^*| = |P'|$;
- 1 additional vertex then again, (u,r) cannot be a shortcut of the embedded path P. So, $|P| = |P^*| = |P'|+1$;
- 2 additional vertices then (u,r) must be a shortcut in P, as otherwise deleting (u,r) disconnects P'. This implies that currently, P contains all three of the points $\{u,v,r\}$. Then $|P'| = |P|-2 \geq |P^*|-1$, where the inequality follows as taking a shortcut reduces the length of an up-down path by at least 1.

It then follows that a single reticulation edge deletion from N can reduce $d_N(x,y)$ for any two leaves $x,y \in N$ by at most 1. □

Lemma 15. *Let N be a tree-child network. There are four possible shapes on a pair of leaves $\{x,y\}$, depending on the shortest xy up-down path (see Figure 3.9):*

- *a* cherry $\Lambda(x,y)$ *with nodes a,x,y and edges $(a,x),(a,y)$;*
- *a* cherry subdivided by one tree node. *If this tree vertex is the parent of y, we call this $\lambda(x,y)$. The shape has nodes a,b,c,x,y and edges $(a,x),(a,b),(b,y),(b,c)$;*
- *a* reticulated cherry $K(x,y)$ *is a cherry subdivided by one reticulation, where the reticulation is on y. The shape has nodes a,b,c,i,x,y and edges $(a,x),(a,c),(b,c),(c,y),(i,a)$;*
  - *if $i=b$ then call this shape $A(x,y)$;*
  - *if $i \neq b$ then call this shape $H(x,y)$.*
- *if $d_N(x,y) \geq 4$, we say that N contains $\Pi(x,y)$. Note that, when N contains $\Pi(x,y)$, this does not mean just that there exists an xy up-down path of length at least 4, but also that there does not exist an xy up-down path of length at most 3. We call this a shape for brevity.*

Proof. We employ the following distance arguments.

- If $d_N(x,y) = 2$ then N must contain $\Lambda(x,y)$.
- If $d_N(x,y) = 3$ then there is at most one reticulation on the shortest xy up-down path. So if in addition we have that
  - there are no reticulations on the shortest xy up-down path. Then N must contain $\lambda(x,y)$ or $\lambda(y,x)$;
  - there is one reticulation on the shortest xy up-down path then N must contain a reticulated cherry $K(x,y)$ or $K(y,x)$.

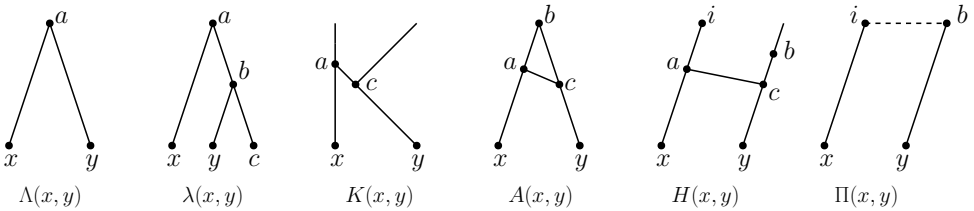


Figure 3.9: All possible shapes on two leaves $\{x, y\}$ (up to permuting x and y). The dashed line indicates that any ib up-down path has length at least 2.

- If $d_N(x, y) \geq 4$ then N must contain $\Pi(x, y)$.

□

We now show that the shape on leaves x and y in a tree-child network is identifiable from the shapes on x and y in its MLLSs. This is summarized in Table 3.1. We start with the following theorem, which shows that each shape is preserved in at least one MLLS.

Theorem 4. *Let N be a level-k tree-child network where $k \geq 2$, and let x, y be two leaves in N. If N contains $\Lambda(x, y)$, $\lambda(x, y)$, $A(x, y)$, $H(x, y)$ or $\Pi(x, y)$, then there is an MLLS of N containing $\Lambda(x, y)$, $\lambda(x, y)$, $A(x, y)$, $H(x, y)$ or $\Pi(x, y)$ respectively.*

Proof. In this proof, we refer to the vertex labels used in Lemma 15.

The case that N contains $\Lambda(x, y)$ is trivial.

Now suppose N contains $\lambda(x, y)$. If c, the sibling of y, is a reticulation then deleting the reticulation edge leading into c that is not (b, c) returns an MLLS containing $\lambda(x, y)$. If c is not a reticulation then deleting any reticulation edge will not affect the shortest xy up-down path. This results in an MLLS containing $\lambda(x, y)$.

Suppose N contains $A(x, y)$. As x, y are leaves, $A(x, y)$ is a level-1 blob and thus by definition, every MLLS of N contains $A(x, y)$.

Suppose N contains $H(x, y)$. If the blob containing the reticulation of $H(x, y)$ is of level lower than k, then every MLLS of N contains $H(x, y)$, and we are done. So suppose this blob is level-k. As $k \geq 2$, there exists a reticulation r, which is not c, with reticulation edges e and f. Let N' and N'' be the MLLSs of N obtained by deleting e and f (amongst other reticulation edges) respectively. We claim that at least one of N' or N'' contains $H(x, y)$. Indeed, if N' contains $A(x, y)$, then in N, either b or i must be incident to e, as otherwise a and c will still have different parents after deleting e and cleaning up. Now, b cannot be incident to e as it violates the tree-child property, regardless of whether b is the tree vertex or the reticulation incident to e. Then i must be incident to e. If i is r, then we note that b cannot be the parent of i due to the tree-child property. This implies that upon deleting e and cleaning up, a and c have different parents, and subsequently N' contains $H(x, y)$. Thus this case is impossible. If, on the other hand, i is the tree vertex of e, then neither i nor b are suppressed after deleting f and cleaning up. This implies that N'' contains $H(x, y)$.

Suppose N contains $\Pi(x,y)$. Suppose first that $d_N(x,y) \geq 5$. Take any xy up-down path in N, and consider $BT(N)$. Note that any up-down path in N can be mapped to an up-down path in $BT(N)$. The 'up' portion of the path passes through the blob vertices containing x in their label, until the first blob vertex containing y is reached. The 'down' portion of the path passes through the blob vertices containing y in their label, until a lowest blob vertex containing y is reached. In particular, the apex is contained in the lowest blob which contains both x and y in their leaf-descendant set. So every xy up-down path in N passes through the same set of blobs $\mathscr{B}$. Furthermore, every xy up-down paths enter and leave the blobs $B \in \mathscr{B}$ at the same nodes. Let t_B and h_B denote these vertices respectively.

We claim that there is a reticulation edge we can delete from any blob $B \in \mathscr{B}$ of level-k such that every xy up-down path uses at least one edge from B in the resultant subnetwork. We assume $lvl(B) = k$ as otherwise the claim holds trivially. At least one of t_B or h_B must be a reticulation, since we enter, pass through, and leave the blob B. We consider the cases when they are both reticulations and when t_B is a reticulation but h_B is not. Suppose first that t_B and h_B are both reticulations. Then B must contain the apex of any xy up-down path; furthermore because of the tree-child property, the shortest $t_B h_B$ up-down distance must be at least 3. Then deleting a reticulation edge incident to h_B either disconnects the xy up-down path or reduces the length by at most 1. In any case, at least one edge of B is still used in the xy up-down paths in the resultant subnetwork. Now suppose that t_B is the only reticulation. Suppose h_B is not incident to any reticulation edge. Since $lvl(B) = k \geq 2$, there exists a reticulation edge we can delete from B, such that neither t_B, h_B, nor the edge (t_B, h_B) are suppressed. Now suppose h_B is incident to a reticulation edge into a reticulation r. If this edge is also incident to t_B, then again since $lvl(B) = k \geq 2$, there exists a reticulation edge we can delete from B, such that neither t_B, h_B, nor the edge (t_B, h_B) are suppressed. Finally if the edge is not incident to t_B, then deleting the reticulation edge incident to r that is not (h_B, r) ensures that t_B, h_B, nor (t_B, h_B) are suppressed. In any case, deleting the chosen reticulation edge returns a subnetwork in which an edge of B is used in every xy up-down path.

So if $|\mathscr{B}| \geq 2$, then there exists an MLLS N^{mlls} in which all xy up-down paths use at least two edges from the blobs in $\mathscr{B}$ plus at least three edges connecting the two blobs, x, and y. Therefore, $d_{N^{mlls}}(x,y) \geq 5$. If $|\mathscr{B}| = 1$ then by Lemma 14, there exists an MLLS N^{mlls} with $d_{N^{mlls}}(x,y) \geq 4$. Thus, if $d_N(x,y) \geq 5$ then there is an MLLS of N containing $\Pi(x,y)$.

Suppose now that $d_N(x,y) = 4$. We first show that there are at most 2 shortest xy up-down paths in N. Let u, v be the parents of x, y respectively. Then any shortest xy up-down path is always of the form $(x,u),(u,w),(w,v),(v,y)$ (disregarding directions) where w is some vertex in N, and one of u, v, w is the apex of the shortest up-down path. Note that u and v are always included in any xy up-down path, since they are the parents of x and y respectively. Therefore, having two shortest xy up-down paths where u and v are the apex in each would create a cycle in N, contradicting the fact that N is a phylogenetic network. Therefore if u is the apex of a shortest xy up-down path in N then there cannot be a shortest xy up-down path where v is the apex. There can be, however, a second shortest xy up-down path in N where w is the apex.

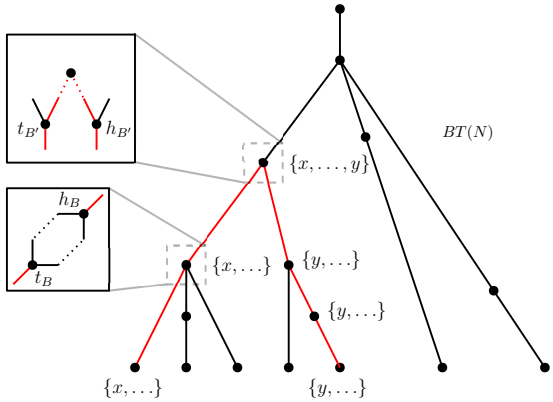


Figure 3.10: Proof visual of Theorem 4, $d_N(x,y) \geq 5$ case. The red up-down path in $BT(N)$ represents the trajectory of every xy up-down path in N, and consequently the set of blobs $\mathscr{B}$ through which every xy up-down path passes. A zoomed-in portion of the two particular blob vertices illustrates the entry point t_B and exit h_B in N, and the case for when both points can be reticulations.

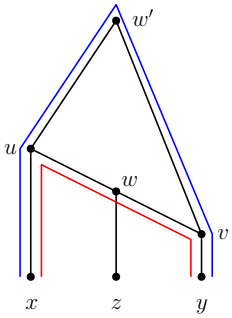


Figure 3.11: An example of 2 shortest xy up-down paths in N, whenever $d_N(x,y) = 4$. One up-down path (red) is $(x,u),(u,w),(w,v),(v,y)$ and the other (blue) $(x,u),(u,w'),(w',v),(v,y)$ (disregarding directions).

Since u, v are contained in all xy up-down paths, we have that if two shortest xy up-down paths have the same apex then they must be the same up-down paths. Otherwise the network would not be binary, or there would be parallel edges. If there were more than two shortest xy up-down paths then at least one of u or w would have degree greater than 3. This implies N is non-binary, so there can be at most two shortest xy up-down paths. This is shown in Figure 3.11. Note that if there are two shortest xy up-down paths in N then it must be isomorphic to the one shown in Figure 3.11, as otherwise the only other option would be to have w and w' be the apex, in which case w' would be a parent of 2 reticulations, deeming N to be not tree-child.

Now we show that if $d_N(x,y) = 4$ then there is always an MLLS of N containing $\Pi(x,y)$.

Suppose first that there are two shortest xy up-down paths. Then, as stated before, it is isomorphic to the diagram shown in Figure 3.11. There are no reticulation edges incident to either of the shortest paths other than on the reticulation at v. In particular (w,z) cannot be a reticulation edge because N is tree-child. As $lvl(N) \geq 2$, there is

another reticulation edge e incident to a reticulation that is not v. Indeed, parents of x and y remain different and non-adjacent in the MLLS obtained by deleting e. This particular MLLS contains $\Pi(x,y)$.

Now suppose there is only one shortest xy up-down path P. There are 5 vertices on P including x and y, and there are at most two reticulation edges incident to P and at most one on P by the tree-child property. Since $lvl(N) \geq 2$, there is at least one reticulation edge such that its deletion does not affect P. Deleting this reticulation edge and cleaning up ensures that the parents of x and y remain different and non-adjacent in the resultant MLLS. Therefore, there exists an MLLS of N which contains $\Pi(x,y)$.

Thus if $d_N(x,y) = 4$ then there exists an MLLS of N containing $\Pi(x,y)$. Therefore if N contains $\Pi(x,y)$, there exists an MLLS of N containing $\Pi(x,y)$.

3

□

Lemma 16. *For a tree-child network N, if N contains $\Pi(x,y)$ then no MLLS of N contains $\Lambda(x,y)$.*

Proof. We prove the contrapositive. Suppose one of the MLLSs N^{mlls} of N contains $\Lambda(x,y)$. Add the deleted reticulation edges back to N^{mlls}. Then every vertex on a shortest xy up-down path, excluding the apex, is incident to a reticulation edge. We first show that these vertices cannot be pure vertices in N.

Suppose for a contradiction that one of these nodes p is a pure vertex in N. Then p must be a tree node, and there must exist two disjoint paths from p to its reticulation child r. Without loss of generality, suppose that p is above x. Then there exists a node z that is above x and below p such that z is above r. When we delete the reticulation edges again to obtain N^{mlls}, we must delete two edges from the blob with pure node p, which is impossible. We have a contradiction.

Now suppose for a contradiction that there are two nodes u, v on a shortest xy up-down path in N excluding the apex. By our assumption, u and v are contained in a level-k blob. By the above claim, neither u nor v can be pure vertices in N, and we note that the blob containing u contains the apex, and the blob containing v also contains the apex. This implies that u and v are contained in the same level-k blob. To obtain N^{mlls}, only one of u or v can be suppressed. In particular, (u,v) cannot be an edge in N as otherwise, this blob would be a level-1 blob. This implies that N^{mlls} does not contain $\Lambda(x,y)$, a contradiction.

Therefore, there can only be one vertex on a shortest xy up-down path in N excluding the apex, and thus $d_N(x,y) \leq 3$. Hence N does not contain $\Pi(x,y)$. □

Theorem 5. *Let N be a level-k tree-child network where $k \geq 2$, and let x, y be two leaves in N. The shape on $\{x,y\}$ in N is identifiable from the shapes on $\{x,y\}$ in the MLLSs.*

Proof. We now prove a series of claims which state that N contains a certain shape if and only if there are distinct MLLSs of N containing certain shape(s) on $\{x,y\}$, and not containing certain other shape(s) on $\{x,y\}$.

Claim 3. *N contains $\Lambda(x,y)$ if and only if all MLLSs of N contain $\Lambda(x,y)$.*

Proof 1: To show necessity, suppose N contains $\Lambda(x,y)$ so that $d_N(x,y) = 2$. Since the parent of x and y is a tree node, there is no reticulation edge incident to $\Lambda(x,y)$. Then $\Lambda(x,y)$ is contained in every maximum subnetwork of N, and therefore in every MLLS of N.

For sufficiency, suppose for a contradiction that all MLLSs of N on X contain $\Lambda(x,y)$, but N does not. If N contains $\lambda(x,y), \lambda(y,x), K(x,y), K(y,x)$, or $\Pi(x,y)$ then, as these are the only possible shapes and their shapes are preserved in some MLLSs by Theorem 4, we have our required contradiction. Thus the claim holds.

■

Claim 4. *N contains $\lambda(x,y)$ if and only if there exists an MLLS of N containing $\lambda(x,y)$ and no MLLSs of N contain $\lambda(y,x), K(x,y), K(y,x)$ or $\Pi(x,y)$.*

Proof 2: To show necessity note that by Theorem 4, there is an MLLS of N that contains $\lambda(x,y)$. The only possible reticulation edge incident to $\lambda(x,y)$ is at b whenever c is a reticulation. Deleting the edge (b,c) returns an MLLS containing $\Lambda(x,y)$, and deleting the reticulation edge incident to c that is not (b,c) returns an MLLS containing $\lambda(x,y)$. All other reticulation edges do not intersect $\lambda(x,y)$ and hence their deletions do not affect $\lambda(x,y)$. Thus an MLLS of N does not contain $\lambda(y,x), K(x,y), K(y,x)$ nor $\Pi(x,y)$. The condition is therefore necessary.

To show sufficiency, suppose for a contradiction that the conditions hold but N does not contain $\lambda(x,y)$. If N contains $\Lambda(x,y)$ then by Claim 1, no MLLSs of N contain $\lambda(x,y)$, a contradiction. If N contains $\lambda(y,x), K(x,y), K(y,x)$, or $\Pi(x,y)$ then, as these are the only possible shapes and their shapes are preserved in some MLLSs by Theorem 4, we have our required contradiction. The condition is necessary and the claim holds.

■

Since $A(x,y)$ is a level-1 blob in N for two leaves $x, y \in X$, Claim 3 is trivially true.

Claim 5. *N contains $A(x,y)$ if and only if all MLLSs of N contain $A(x,y)$.*

When N contains $H(x,y)$, let B_H be the blob containing the reticulation in $H(x,y)$.

Claim 6.

- *N contains $H(x,y)$ and $lvl(B_H) = k$ if and only if there exist distinct MLLSs of N containing $\Lambda(x,y)$ and $H(x,y)$, and no MLLSs of N contain $K(y,x)$.*
- *N contains $H(x,y)$ and $lvl(B_H) < k$ if and only if all MLLSs of N contain $H(x,y)$.*

Proof 4: We first prove the first statement of the claim. We first show necessity. Isolating $\{x,y\}$ returns an MLLS of N containing $\Lambda(x,y)$. By Theorem 4, there is an MLLS of N which contains $H(x,y)$. For the third condition, suppose for a contradiction that some MLLS N^{mlls} of N contains $K(y,x)$. Since we have a reticulation on y in $H(x,y)$, and because isolating $\{x,y\}$ returns $\Lambda(x,y)$, N^{mlls} must have been obtained by cutting $\{x,y\}$.

But then we have that the node b, the grandparent of y, has only reticulation children in N, contradicting the tree-child property of N. We therefore have necessity.

To show sufficiency, suppose for a contradiction that the conditions hold but N does not contain $H(x,y)$. If N contains $\Lambda(x,y)$ then by Claim 1, no MLLSs of N contain $H(x,y)$, a contradiction. If N contains $\lambda(x,y)$ or $\lambda(y,x)$ then no MLLSs of N contains $H(x,y)$ by Claim 2, a contradiction. If N contains $A(x,y)$ then no MLLSs of N contains $\Lambda(x,y)$ by Claim 3, a contradiction. If N contains $K(y,x)$ then the shape is preserved in some MLLS of N by Theorem 4, a contradiction. Finally if N contains $\Pi(x,y)$ then no MLLS of N contains $\Lambda(x,y)$ by Lemma 16, a contradiction. As these are the only possibilities, necessity follows. The claim holds for $lvl(B_H) = k$.

We now prove the second statement of the claim. We first show necessity. Now suppose that N contains $H(x,y)$ and $lvl(B_H) < k$. Then none of the reticulation edges in B_H are deleted to obtain any of the MLLSs of N by definition. It follows that all MLLSs of N contain $H(x,y)$.

We now show sufficiency. Suppose first that every MLLS of N contains $H(x,y)$. If N contained a shape that was not $H(x,y)$, then there exists an MLLS of N that contains that particular shape by Theorem 4. As this is a contradiction, we have that N contains $H(x,y)$. To show that $lvl(B_H) < k$, we note that if this was not the case, i.e., if $lvl(B_H) = k$, then we have shown above that an MLLS of N would contain $\Lambda(x,y)$, which is a contradiction. So we must have that N contains $H(x,y)$ and that $lvl(B_H) < k$.

■

Claim 7. *N contains $\Pi(x,y)$ if and only if there exists an MLLS of N containing $\Pi(x,y)$ and no MLLSs of N contain $\Lambda(x,y)$.*

Proof 5: We first show necessity. There is an MLLS of N that contains $\Pi(x,y)$ by Theorem 4. By Lemma 16, no MLLSs of N contains $\Lambda(x,y)$.

To show sufficiency, suppose for a contradiction that the conditions hold, but that N does not contain $\Pi(x,y)$. If N contains $\Lambda(x,y)$ then by Claim 1, every MLLS of N contains $\Lambda(x,y)$, a contradiction. If N contains $\lambda(x,y)$ or $\lambda(y,x)$ then no MLLSs of N contain $\Pi(x,y)$ by Claim 2, a contradiction. If N contains $A(x,y)$ or $A(y,x)$ then all MLLSs of N contains $A(x,y)$ or $A(y,x)$ by Claim 3. This is a contradiction as no MLLSs of N would contain $\Pi(x,y)$. If N contains $H(x,y)$ or $H(y,x)$ then we split into two cases. Recall that B_H is the blob of N which contains $H(x,y)$ or $H(y,x)$. If $lvl(B_H) < k$ then all MLLSs of N contains $H(x,y)$ or $H(y,x)$ by Claim 4. This is a contradiction as no MLLSs of N would contain $\Pi(x,y)$. If $lvl(B_H) = k$ then there exists an MLLS of N which contains $\Lambda(x,y)$ by Claim 4, a contradiction. The condition is sufficient. The claim therefore holds.

■

□

Theorem 5 is summarized in table 3.1. The table covers all of the different cases, showing which shapes can appear in MLLSs given the shape that the original network contains. For any two rows in the table, there is some column in which one row has a

N contains	$\Lambda(x,y)$	$A(x,y)$	$A(y,x)$	$H(x,y)$	$H(y,x)$	$\lambda(x,y)$	$\lambda(y,x)$	$\Pi(x,y)$
$\Lambda(x,y)$	✓	X	X	X	X	X	X	X
$A(x,y)$	X	✓	X	X	X	X	X	X
$H(x,y)$	✓(X)	X(X)	X(X)	✓(✓)	X(X)	?(X)	?(X)	?(X)
$\lambda(x,y)$	?	X	X	X	X	✓	X	X
$\Pi(x,y)$	X	?	?	?	?	?	?	✓
$A(y,x)$	X	X	✓	X	X	X	X	X
$H(y,x)$	✓(X)	X(X)	X(X)	X(X)	✓(✓)	?(X)	?(X)	?(X)
$\lambda(y,x)$	?	X	X	X	X	X	✓	X

Table 3.1: Given a tree-child network N contains one of the $\{x,y\}$ shapes listed in the first column, for each shape listed on the first row, this shape must appear in an MLLS of N if there is a checkmark (✓), this shape cannot appear in an MLLS if there is a cross (X), and a question mark (?) means either could be possible. In the rows $H(x,y)$ and $H(y,x)$, the non-bracketed marks are for the case $lvl(B_H) = k$ and the bracketed marks are for the case $lvl(B_H) < k$.

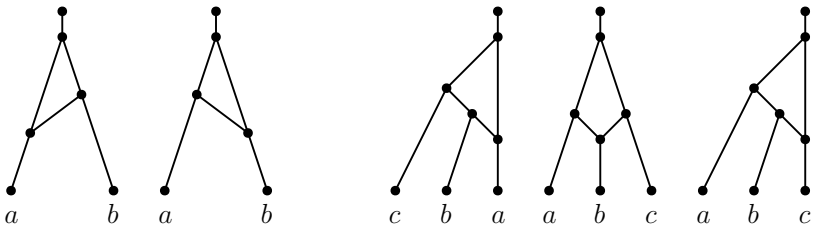


Figure 3.12: (a) Two non-isomorphic level-1 networks with girth 3 that share the same subnetworks. (b) Three non-isomorphic level-1 networks with girth 4 that share the same subnetworks.

check and the other a cross. Thus, we can distinguish between any two cases just by looking at the MLLSs, and so we can determine the structure between x and y on N. Because the given shapes are the only possibilities between two leaves x and y, the table covers all possible cases.

3.5. RECONSTRUCTIBILITY OF TREE-CHILD NETWORKS

In this section, we show that the class of *tree-child networks*, excluding trees and level-1 networks with girth at most 4, is MLLS-reconstructible and thus level-reconstructible and subnetwork-reconstructible (where the girth is the length of a smallest cycle in the underlying undirected graph). A pair of level-1 networks with girth 3 and girth 4 that is not subnetwork-reconstructible is shown in Figure 3.12.

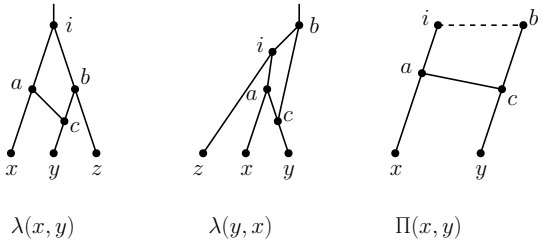


Figure 3.13: The three cases for $H(x,y)$ in Lemma 17. Deleting edge (a,c) yields $\lambda(x,y)$, $\lambda(y,x)$, and $\Pi(x,y)$ respectively.

Following the leaf pair analysis in Section 3.4, we show here that it is possible to infer the location of a missing reticulation edge for level-k blobs from the MLLSs. By Lemma 7, there exists a cherry or a reticulated cherry in every tree-child network. We know that cherries are level-0 blobs and A shapes are level-1 blobs. Then, the reconstruction of level-k blobs can be accomplished by reconstructing an H shape of every level-k blob.

We start by analyzing the possible shapes on x, y after cutting a reticulated cherry on x and y, see Figure 3.13 for examples.

Lemma 17. *Let N be a tree-child network and suppose N contains $H(x,y)$ on a leaf pair $\{x,y\}$. Then the maximum subnetwork obtained by cutting the reticulated cherry $\{x,y\}$ contains one of $\lambda(x,y), \lambda(y,x)$, or $\Pi(x,y)$. Furthermore, all other maximum subnetworks of N contain either $\Lambda(x,y)$ or $H(x,y)$.*

Proof. Suppose for a contradiction that cutting $\{x,y\}$ returns a maximum subnetwork N' of N containing either $\Lambda(x,y), K(x,y)$, or $K(y,x)$. If N' contains $\Lambda(x,y)$, then the parent of x and the parent of y must share a common parent in N. This implies that N contains $A(x,y)$, a contradiction. If N' contains $K(x,y)$, then the parent of y is a child of a reticulation in N. This contradicts the tree-child property of N. N' cannot contain $K(y,x)$ by Theorem 5.

To prove the second statement of the lemma, note that isolating $H(x,y)$ returns a maximum subnetwork of N that contains $\Lambda(x,y)$, and deleting any reticulation edge that is not incident to y returns a maximum subnetwork that contains $H(x,y)$, since the parent of x and the parent of y is not suppressed and they are adjacent. □

We now show how we can reconstruct a blob containing the reticulation of a reticulated cherry, see Figure 3.14 for an example.

Lemma 18. *Let N be a level-k tree-child network, and suppose N contains $H(x,y)$ for a leaf pair $\{x,y\}$. Suppose in addition that the blob B containing the reticulation of $H(x,y)$ is level-k. Then we can reconstruct B in the MLLSs of N.*

Proof. By Theorem 5 and Lemma 17, N contains $H(x,y)$ if and only if all MLLSs of N contain either $\Lambda(x,y)$, $H(x,y)$, $\lambda(x,y)$, $\lambda(y,x)$, or $\Pi(x,y)$, and there exist distinct MLLSs of N that contain $\Lambda(x,y), H(x,y)$, and one of $\lambda(x,y), \lambda(y,x)$, or $\Pi(x,y)$. Now find the MLLS N^{mlls} of N that contains one of $\lambda(x,y), \lambda(y,x)$, or $\Pi(x,y)$. Introduce nodes a, c

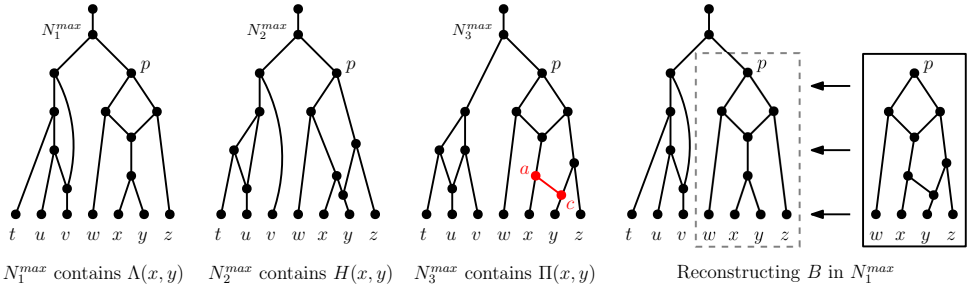


Figure 3.14: Three MLLSs N_1^{mlls}, N_2^{mlls}, and N_3^{mlls} of a level-2 tree-child network containing exactly two level-2 blobs. The three MLLSs contain $\Lambda(x,y), H(x,y)$, and $\Pi(x,y)$ respectively. We reconstruct the blob B with pure node p in N_3^{mlls} initially, then reconstruct it in the other MLLSs. In N_3^{mlls}, nodes a, c are inserted directly above x, y respectively, and an edge (a,c) is added (shown in red). To reconstruct B in N_1^{mlls} the pendant subnetwork rooted at p is replaced by the reconstructed pendant subnetwork.

directly above x, y respectively, and add an edge (a,c) to N^{mlls}. This reconstructs the blob B in N^{mlls}.

It remains to show how to reconstruct B in the other MLLSs. Hence, consider an arbitrary MLLS N_1^{mlls}. Let $A \subseteq X$ denote the set of leaf-descendants of the pure node p of B in N. Then A is a vertex of $BT(N)$. Let $\Gamma(A)$ denote the set of all outgoing neighbors of A in $BT(N)$. Let p_i for $i = 1,\ldots,|\Gamma(A)|$ denote the corresponding pure vertices in N. In N, delete the tree edge leading into p, and also delete the two outgoing edges of p_i for $i = 1,\ldots,|\Gamma(A)|$, but do not clean up. Call the component that contains the node p the *B-part of N*. By Theorem 2, the MLLS N_1^{mlls} contains pure vertices with leaf-descendant set A and with leaf-descendant set equal to each set in $\Gamma(A)$. Hence, we can define the B-part of N_1^{mlls} analogously as for N (but note that in N_1^{mlls} it is not necessarily a single blob). We can then reconstruct the blob B in N_1^{mlls} by replacing the B-part of N_1^{mlls} by the B-part of N^{mlls}. □

Definition 5. Let N be a tree-child network. A cherry $\Lambda(x,y)$ is *reduced* by deleting the node y and cleaning up (same definition as in [9]). A reticulated cherry $K(x,y)$ is *reduced* by isolating $K(x,y)$ and reducing the resultant cherry $\Lambda(x,y)$ (different definition to one in [9]).

The following observation shows how we can obtain the MLLSs of a network obtained by reducing a reticulated cherry from the MLLSs of the original network. Note that a maximum subnetwork obtained by isolating a reticulated cherry in a tree-child network remains tree-child by Lemma 4 and that a network obtained by reducing a cherry in a tree-child network also remains tree-child [9].

Observation 4. *Let N be a level-k tree-child network with a cherry or a reticulated cherry on $\{x,y\}$, and let N' be the tree-child network obtained by reducing $\{x,y\}$ from N.*

- *If N contains $H(x,y)$ and the blob B containing $H(x,y)$ is of level-k, then, in each MLLS of N, reconstruct B by Lemma 18 and subsequently reduce the reticulated cherry $H(x,y)$.*

- *Otherwise, reduce $\{x, y\}$ in all MLLSs of N.*

Let $\mathscr{S}$ denote the set of networks we obtain from either of the above two cases. Then $\mathscr{S}$ is precisely the set of all MLLSs of N'.

Theorem 6. *The class of binary level-k tree-child networks is MLLS-reconstructible, for $k \geq 2$.*

Proof. We prove by induction on $|X|$ that, for each level-k tree-child network N on X with MLLS set $\mathscr{M}$, the network N is the unique level-k tree-child network with MLLS set $\mathscr{M}$. The base case $|X| = 1$ is trivially true as when there is only one leaf, any network of level-2 or higher is no longer tree-child. So suppose $|X| > 1$ and that the claim is true for each level-k tree-child network on at most $|X| - 1$ leaves. Let N be a level-k tree-child network on X and let $\mathscr{M}$ be its MLLS set.

By Lemma 7, there exists at least one leaf pair $\{x, y\}$ that forms a cherry or a reticulated cherry in N.

If N contains $H(x, y)$ and the blob B containing the reticulation of $H(x, y)$ is of level-k, then reconstruct B in each element of $\mathscr{M}$ as outlined in Lemma 18. If B is the only level-k blob, we are done. Otherwise, we proceed as follows. Note that we can do this, as we can identify all level-k blobs by Lemma 13.

Reduce either the cherry or the reticulated cherry $\{x, y\}$ in each element of $\mathscr{M}$. Call this new set of networks $\mathscr{S}$. Each network in $\mathscr{S}$ is tree-child and contains $|X| - 1$ leaves. By Observation 4, $\mathscr{S}$ is the set of all MLLSs of N', the level-k tree-child network obtained by reducing $\{x, y\}$ in N. By the induction hypothesis, N' is the unique level-k tree-child networks with MLLS set $\mathscr{S}$. Reconstructing N' and undoing the reduction operation on $\{x, y\}$ yields the tree-child network N, which is therefore the unique level-k tree-child network with MLLS set $\mathscr{M}$. □

Gambette et al. have shown that level-1 networks with girth at least 5 are level-reconstructible [5]. The next corollary follows from their results, Observation 2, Theorem 6, and the following observation.

Observation 5. *Let N and N' be two tree-child networks that are both either level at least 2 or girth at least 4. If the level of N and N' is different, then they do not have the same set of lower-level subnetworks.*

Corollary 2. *The class of tree-child networks, excluding trees and level-1 networks with girth at most 4, is MLLS-reconstructible, level-reconstructible and subnetwork-reconstructible.*

3.6. RECONSTRUCTION ALGORITHM FOR TREE-CHILD NETWORKS

In this section we present an algorithm in the form of pseudo-code for reconstructing tree-child networks from their MLLSs. As shown in Section 3.5, we need only to reconstruct the H shapes contained in level-k blobs. Algorithm 1 systematically rebuilds every level-k blob from the bottom-up, reducing common pendant subnetworks to leaves on the way. We give an example of Algorithm 1 in Figure 3.15. To keep the description of the algorithm concise, it assumes that the input $\mathscr{T}$ consists of the set of MLLSs of some

level-k tree-child network, with $k \geq 2$. Nevertheless, the algorithm can in principle also be used to decide whether such a network exists or not. If the algorithm returns a network N, then we can check whether $\mathcal{T} = \mathcal{N}^{mlls}(N)$. If this is not the case, or the algorithm fails to output a network, then such a network does not exist (see Theorem 7). Checking whether $\mathcal{T} = \mathcal{N}^{mlls}(N)$ can be done in $O(|\mathcal{T}|^2|X|^2)$ time, because checking whether two tree-child networks are isomorphic can be done in $O(|X|^2)$ time [7].

Moreover, the algorithm can even be applied to an arbitrary set of level-$k-1$ tree-child networks as input. If a network displaying the input networks exists the algorithm may find it, but is not guaranteed to do so (see Theorem 8.)

Before presenting the algorithm, we first go over a few key ideas required to prove the correctness and find the time complexity of the algorithm. We reiterate the idea of collapsing a pendant subnetwork from a network (presented in Subsection 3.3.3, and additionally define what it means to collapse a common pendant subnetwork from a set of networks. Let N be a network and let N_A be a pendant subnetwork of N rooted by a vertex with leaf-descendant set A. Collapsing N_A from N means that we replace N_A by a leaf A. Let $N \setminus N_A$ denote the network obtained by collapsing N_A from N. Let $\mathcal{M}$ be a set of networks containing a common pendant subnetwork N_A. *Collapsing* N_A from $\mathcal{M}$ means that we collapse N_A from every network in $\mathcal{M}$. Let $\mathcal{M} \setminus N_A$ denote the set of networks obtained by collapsing N_A from $\mathcal{M}$.

Lemma 19. *Let N be a level-k tree-child network where $k \geq 2$. If there exists a common pendant subnetwork N_A for the MLLSs of N, then N_A is a pendant subnetwork of N.*

Proof. Consider the blob tree $BT(N_A)$. Since N_A is a common pendant subnetwork of all MLLSs of N, the blob tree $BT(N_A)$ is a common pendant subtree of all blob trees of the MLLSs of N. By Theorem 2, $BT(N)$ must contain $BT(N_A)$. By Lemma 13, N_A must be a level-$k' < k$ network. Therefore N_A is a pendant subnetwork of N. □

The following two observations follow directly from Lemma 13.

Observation 6. *Let N be a level-k tree-child network where $k \geq 2$. There exists no common pendant subnetwork for the MLLSs of N if and only if all lowest blobs in N are of level-k.*

Observation 7. *Let N be a level-k tree-child network where $k \geq 2$. There exists a common pendant subnetwork N_A for the MLLSs of N if and only if there exists a common pendant subtree rooted at A for the blob trees of the MLLSs of N.*

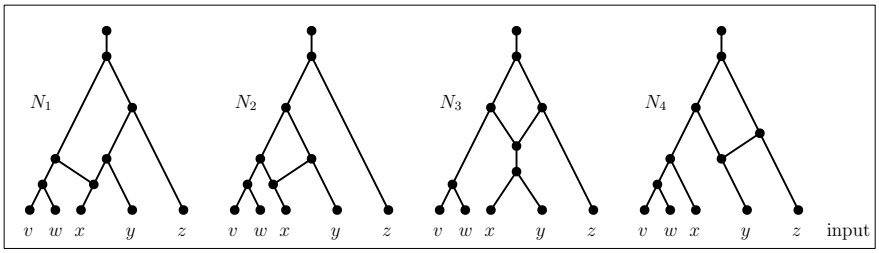
Theorem 7. *Let N be a level-k tree-child network on X where $k \geq 2$, and let $\mathcal{T} = \mathcal{N}^{mlls}(N)$. Algorithm 1 finds the network N in time $O(|\mathcal{T}||X|^3/k)$.*

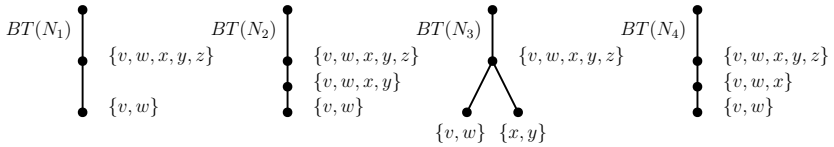
Proof. We first prove the correctness of the algorithm. By Lemma 19, every maximal common pendant subnetwork of $\mathcal{T}$ is a pendant subnetwork of N. Let N_A be a maximal common pendant subnetwork of $\mathcal{T}$. Then we can collapse N_A from N and $\mathcal{T}$, solve a smaller instance of the MLLS reconstruction problem by reconstructing $N \setminus N_A$ from $\mathcal{T} \setminus N_A$, and then appending N_A to the leaf labelled A – which is the final step of the algorithm – returns the network N. Since all maximal pendant subnetworks are disjoint from one another, we can repeat this reduction for all maximal pendant subnetworks,

Algorithm 1: Algorithm TCMLLS-RECONSTRUCTION($\mathcal{T}$)

Data: A set $\mathcal{T} = \mathcal{N}^{mlls}(N)$ for some level-k tree-child network N, where $k \geq 2$
Result: The network N

Update $\mathcal{T}$ by collapsing maximal common pendant subnetworks from every network in $\mathcal{T}$
Find the blob tree for each network in $\mathcal{T}$
Find a minimal set A that is a vertex of the blob tree of each network in $\mathcal{T}$
Find a leaf pair $\{x, y\}$ where $x, y \in A$ such that distinct networks N_1, N_2, N_3 of $\mathcal{T}$ contain $\Lambda(x, y), H(x, y)$, and one of $\lambda(x, y), \lambda(y, x)$, or $\Pi(x, y)$ respectively
Update N_3 by adding nodes a, c directly above x, y respectively and an edge (a, c)
Let N_A denote the pendant subnetwork rooted at the top vertex of this blob in N_3
for $N^{mlls} \in \mathcal{T}$ **do**
    Find the pure node p with leaf-descendant set A
    Replace the pendant subnetwork rooted at p by N_A
    Collapse N_A from N^{mlls}
if *$\mathcal{T}$ contains a single element T* **then**
    $N' := T$
else
    $N' :=$ TCMLLS-RECONSTRUCTION($\mathcal{T}$)
Construct N from N' by appending the maximal common pendant subnetworks we have collapsed
return N





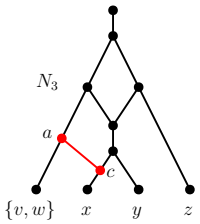
The set of foundation nodes is $\{\{v,w\},\{v,w,x,y,z\}\}$
The cherry $\Lambda(v,w)$ is the only common pendant subnetwork of the input networks. We reduce it to a leaf $\{v,w\}$.

Reducing common pendant subnetworks

In the reduced networks, we reconstruct the blob with leaf-descendant set $\{\{v,w\},x,y,z\}$, since this set corresponds to a lowest foundation node.

N_1 contains $H(\{v,w\},x)$
N_2 contains $H(\{v,w\},x)$
N_3 contains $\Pi(\{v,w\},x)$
N_4 contains $\Lambda(\{v,w\},x)$

Then we add nodes a,c, and add edges $(a,\{v,w\}),(a,c),(c,x)$ to N_3



Reconstructing a level k blob

Upon reconstructing the same blob in N_1, N_2, and in N_4, we note that all the updated networks are isomorphic.

We are in the *if* case of the algorithm, and thus it returns the above network, together with the cherry $\Lambda(x,y)$ appended to it.

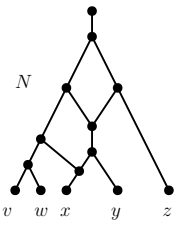


Figure 3.15: An example of the algorithm TCMLLS-RECONSTRUCTION($\{N_1,N_2,N_3,N_4\}$).

by considering the reductions separately. Let $\mathcal{T}'$ and N' denote the set of networks and network obtained by collapsing these pendant subnetworks from $\mathcal{T}$ and N respectively. By Observation 6, we have that all lowest blobs of N' are of level-k. We search for a lowest level-k blob B by finding a minimal set A' that is a vertex of the blob tree of each network in $\mathcal{T}$. Then A' is a lowest foundation vertex of N' by Theorem 2. Then we search for a leaf pair $\{x, y\}$ which form a reticulated cherry in N' with the reticulation in B. We note that such a leaf pair exists since B is a lowest blob and since we have collapsed all common pendant subnetworks. Moreover, we can find it by searching for a pair of leaves as described in the algorithm by Table 3.1 and its proof in Theorem 5. Now we reconstruct B using the steps outlined in the proof of Lemma 18. Let A' denote the leaf-descendant set of B, and let $N'_{A'}$ denote the corresponding pendant subnetwork (the subnetwork is pendant since B is a lowest blob). At this point, if we only needed to reconstruct one level-k blob (i.e. the case when N' contains one level-k blob), then we have reconstructed N'. Otherwise, collapsing $N'_{A'}$ gives the full set of MLLSs of the network $N' \setminus N'_{A'}$. Continuing this reasoning, the recursive call will return the network N'. Appending the collapsed maximum common pendant subnetworks to N' returns the network N.

Each recursive call of TCMLLS-RECONSTRUCTION reconstructs one level-k blob, and therefore the algorithm terminates once we have reconstructed all level-k blobs, in which case we have reconstructed the network.

For the running time, observe that each recursive call of TCMLLS-RECONSTRUCTION acts on an instance $\mathcal{N}^{mlls}(N')$ on leaf set X' such that $|X'| < |X|$ and there is one fewer level-k blob that needs to be reconstructed in the networks of $\mathcal{N}^{mlls}(N')$ when compared to that of $\mathcal{N}^{mlls}(N)$. Since every level-k blob has at least $k+1$ outgoing edges, the number of level-k blobs in N is at most $|X|/k$. Then TCMLLS-RECONSTRUCTION is called at most $|X|/k$ times.

Each single iteration of TCMLLS-RECONSTRUCTION can be split into four parts - collapsing largest common pendant subnetworks from the networks, finding a lowest foundation node A (i.e. finding a lowest level-k blob), reconstructing a lowest level-k blob, and checking whether all updated networks are isomorphic to one another. Finding a largest common pendant subnetwork can be done by looking at the blob trees, which can all be constructed in $O(|\mathcal{T}||X|)$ time. By Observation 7, there exists a common pendant subnetwork rooted at the pure vertex with leaf-descendant set A, if there exists a common pendant subtree rooted at A in the blob trees. The number of blob vertices is maximized for a tree on $|X|$ leaves (or whenever every reticulation is in a triangle), in which case it contains $2|X|$ vertices in total (including the root). Then there exist at most $|X|-1$ foundation nodes, and thus at most $|X|-1$ pendant subnetworks. For every possible pendant subnetwork, we iterate through the networks in $\mathcal{T}$. This step takes $O(|\mathcal{T}||X|)$ time. Finding a lowest foundation vertex follows immediately as we have found all pendant subnetworks of the blob trees. Then this step takes constant time. The level-k blob reconstruction chooses a pair of leaves which descend from a blob and subsequently searches through all networks in $\mathcal{T}$ to see if the pair is the H shape we seek. This takes $O(|\mathcal{T}||X|^2)$ time if we try each pair of leaves (or $O(|\mathcal{T}|^2|X|)$ time if we try each cherry of each network). To decide whether all networks have become isomorphic, we only need to check whether each blob tree consists of just two vertices (including the

root). It is not necessary to check whether some networks have become isomorphic after each recursion, since the algorithm still works if the input set contains isomorphic networks. The algorithm terminates when all level-k blobs have been reconstructed: this is precisely when all networks become isomorphic. Since we can construct (or update) the blob trees in $O(|\mathcal{T}||X|)$ time, we can decide whether all networks have become isomorphic in $O(|\mathcal{T}||X|)$ time.

Thus the total time over a single iteration of TCMLLS-RECONSTRUCTION is $O(|\mathcal{T}||X|^2)$ (or $O(|\mathcal{T}|^2|X|)$).

It follows that the total running time of the algorithm is $O(|\mathcal{T}||X|^3/k)$ (or $O(|\mathcal{T}|^2|X|^2/k)$). □

Here we restrict the input data $\mathcal{T}$ to be the full-set of MLLSs of some tree-child network N, and return N. We now show that it is not necessary to have this restriction: in fact, we require only three MLLSs to reconstruct N. That is, the same three MLLSs can be used to reconstruct each level-k blob. However, if we do not have all MLLSs, we are unable to identify the level-k blobs. Therefore, we require here that also the number of reticulations in the network is given.

Theorem 8. *Three MLLSs suffice to reconstruct a level-k tree-child network, with $k \geq 2$, if the number l of level-k blobs is known.*

Proof. Let N be a level-k tree-child network with l level-k blobs and $k \geq 2$. We first pick three MLLSs N_1^{mlls}, N_2^{mlls}, and N_3^{mlls} of N and then show that Algorithm 1 returns N with these inputs and that no other tree-child network exists with these three MLLSs and l level-k blobs.

Let $B_1, \ldots, B_l$ denote the level-k blobs in N, and let, for $i = 1, \ldots, l$, r_i denote a reticulation in B_i that is in a lowest reticulated cherry shape, which exists by Lemma 6. Since B_i is a level-k blob where $k \geq 2$, the parents of r_i must be non-adjacent (otherwise B_i would be a level-1 blob). Let N_1^{mlls} denote the MLLS of N obtained by cutting the reticulated cherry shapes that contain r_i, and let N_2^{mlls} denote the MLLS of N obtained by isolating the reticulated cherry shapes that contain r_i. Let N_3^{mlls} denote the MLLS of N obtained by deleting, from each level-k blob B_i, a reticulation edge that is not incident to r_i, and such that the parents of r_i remain non-adjacent in N_3^{mlls}. To see that such an MLLS exists, recall that by Theorem 4, we have that if a network contains $H(x,y)$, then there is an MLLS of N that contains $H(x,y)$. Therefore, by treating each level-k blob B_i as a level-k tree-child network, we may invoke Theorem 4 to claim that such an MLLS N_3^{mlls} exists.

Now we show that these three MLLSs suffice to reconstruct N with Algorithm 1. Recall that Algorithm 1 initially collapses all maximal common pendant subnetworks from the input. Let N' denote the network obtained by collapsing the same pendant subnetworks from N. Then, the algorithm finds a minimal set A that is a vertex of the blob tree of each of $N_1^{mlls}, N_2^{mlls}, N_3^{mlls}$. By Lemma 12, A is a leaf of the blob tree of N'. Since the MLLSs $N_1^{mlls}, N_2^{mlls}, N_3^{mlls}$ were constructed in such a way that all common pendant subnetworks are of level strictly lower than k, the set A is the set of leaf-descendants of some lowest level-k blob of N'. Note that, in N', r_1 is contained in a reticulated cherry $H(x,y)$ for some leaves x, y, since there are no blobs below B_1. By construction, the MLLS N_1^{mlls} contains one of $\lambda(x,y), \lambda(y,x)$, or $\Pi(x,y)$. The MLLS N_2^{mlls} contains $\Lambda(x,y)$, and the MLLS N_3^{mlls} contains $H(x,y)$. Hence, we can argue similarly to

in the proof of Theorem 7 that the algorithm then reconstructs the blob B_1 in all input MLLSs, and that recursing the algorithm reconstructs the next lowest level-k blob (which can be reconstructed analogously as done for B_1). It follows then that the algorithm reconstructs N after l recursions of the algorithm.

We now show that no other tree-child level-k network with l level-k blobs exists that displays N_1^{mlls}, N_2^{mlls} and N_3^{mlls}. Suppose such a network N^* exists. Then its MLLS set contains N_1^{mlls}, N_2^{mlls} and N_3^{mlls}. Hence, by the arguments above, running Algorithm 1 on the full set $\mathscr{N}^{mlls}(N^*)$ of MLLSs of N^* returns N. In particular, note that since N^* has the same number of level-k blobs as N, the same common pendant subnetworks are collapsed (in each recursive call) when running the algorithm on $\mathscr{N}^{mlls}(N^*)$ as when we run the algorithm on N_1^{mlls}, N_2^{mlls} and N_3^{mlls}. By Theorem 7, running the algorithm on $\mathscr{N}^{mlls}(N^*)$ returns N^*. Hence, N^* and N are isomorphic. □

Note that in the proof of Theorem 8, we crafted three particular MLLSs N_1^{mlls}, N_2^{mlls}, and N_3^{mlls} however, there could be many triples of MLLSs which are sufficient in reconstructing the original network. Suppose that we have deleted the reticulation edge e_i^j from the blob B_i to obtain the MLLS N_j^{mlls} for $j = 1,2,3$. The proof of Theorem 8 depends on the argument that if for every level-k blob, we can find the particular three shapes, then we can reconstruct said level-k blob. Then we can define three new MLLSs that are also sufficient for reconstructing N as follows. Let N_1 be the MLLS obtained by deleting one of the three reticulation edges (e_1^j, e_2^j, e_3^j) from each level-k blob (B_i). Let N_2 be the MLLS obtained by deleting one of the remaining two reticulation edges, and let N_3 denote the MLLS obtained by deleting the remaining reticulation edges from N. If there were l level-k blobs in N, then we would have $3l$ possible choices of N_1, $2l$ possible choices of N_2, and 1 possible choice of N_3. Therefore we have $6l^2$ possible choices for having a triple of MLLSs that are sufficient for reconstructing the network, given the reticulation edges e_i^j. Note that we simply looked at a particular lowest reticulation r_i for each level-k blob B_i, and also note that there could be more than one reticulation edge that we could have deleted in retrieving the MLLS N_3^{mlls} (in the proof of Theorem 8). This implies that there could be many more triples of MLLSs that suffice to reconstruct N.

Therefore, it is possible to reconstruct the network from a subset of the MLLSs, given that they hold enough information. In particular, if we have the three MLLSs as stated in the proof of Theorem 8, then our algorithm returns the network in time $O(|X|^2/k)$.

3.7. DISCUSSION

IN this chapter, we have shown that level-k tree-child networks, where $k \geq 2$, are determined by their MLLSs and provided a polynomial-time algorithm for reconstructing such a network from its MLLSs. We achieved this result by exploiting one of the tree-child properties - the lowest tree vertex is in a cherry or a reticulated cherry.

The results of this chapter illustrate one way for how a network can be encoded. The key element here is uniqueness, that no two tree-child networks, up to isomorphism, may display the same MLLSs. In particular, this means that in presence of perfect data, we will be able to reconstruct the one true phylogeny.

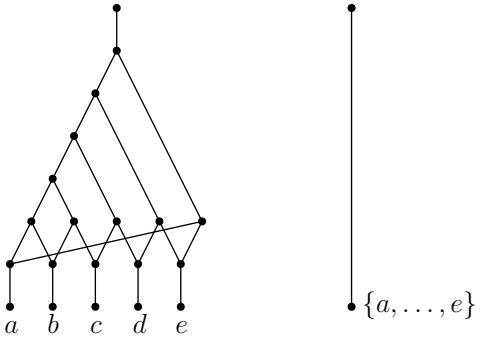


Figure 3.16: A valid network with a tessellating crown blob and its blob tree. Deleting any of the reticulation edges keeps the original blob biconnected, and hence it will not affect the blob tree.

An apparent hindrance to our method is that there is no guarantee nor reason to have the set of all MLLSs of the original network. Converting sequence data into the MLLSs could be quite challenging, especially for higher level. It would therefore be interesting to focus on ways to make our results more practical, possibly employing similar approaches used in methods such as trinet-based methods [12], which work with subnetworks with only three leaves.

On the positive side, we have shown that it is not necessary to know all MLLSs to reconstruct the original network: we need only three, see Theorem 8. Therefore, a more plausible application of our approach would be the following. Suppose, for example, that different studies each manage to produce a network with some reticulations. However, in each of these networks some actual reticulate events have been missed, possibly due to computational limitations or lack of data. Then, a method based on our theoretical results could be used to reconstruct the full network from the networks with missing reticulate events.

Extending our MLLS reconstructibility results to a more general class of networks is another natural step forward. We briefly discuss and explain how it might be possible to adapt our results to the class of *valid* networks, where every reticulation edge in the network is valid.

In the case of valid networks, it is not always true that there exists a reticulation edge in every blob such that its deletion results in a maximum subnetwork where the blob tree differs from the blob tree of the original network (i.e., an analogous statement to Observation 3 does not always hold for valid networks). An example of this, a tessellating crown, is shown in Figure 3.16.

For leaf pair analysis, there is an extra shape on two leaves $\{x, y\}$ where both parents p_x, p_y of x, y respectively are reticulations, and they share a common parent g_x. This shape, which we call a *double-reticulated cherry* (see leaves b, c of Figure 3.16), is distinct from all others (given we adapt the definition of when N contains $\Pi(x, y)$) as it contains $K(x, y)$ and $K(y, x)$ in its MLLSs respectively[1]. Unlike tree-child networks, MLLSs of valid networks can contain a cherry, reticulated cherry, or a double-reticulated

[1]Double-reticulated cherries have a different meaning in other literature [9]. There, they called a leaf triple (x, y, z) a double-reticulated cherry if the network contained $H(x, y)$ and $H(z, y)$.

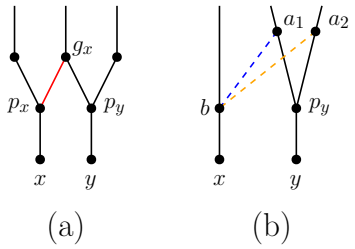


Figure 3.17: (a) Double-reticulated cherry on $\{x, y\}$. (b) MLLS where the red edge is deleted from (a). We have two options, a_1, a_2, for inserting the reticulation edge.

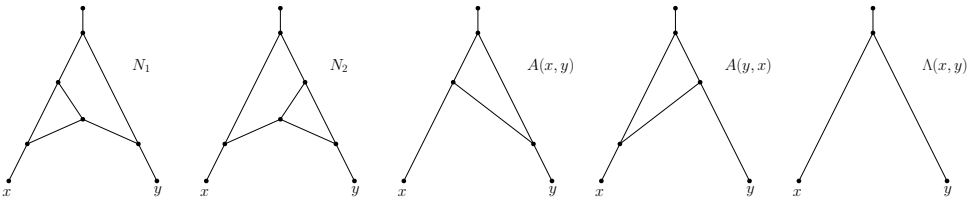


Figure 3.18: Two level-2 networks N_1 and N_2 are non-isomorphic but have the same lower-level subnetworks. In general, invalid networks are not level-reconstructible.

cherry stemming from the lowest tree node. The reconstruction of a double-reticulated cherry poses a challenge as there are two potential places to reinsert the reticulation edge. That is, given an MLLS where (g_x, p_x) has been deleted, we add two nodes a, b with edge (a, b). We know that b must be placed directly above x. However, we have two possibilities for inserting a above p_y (illustrated in Figure 3.17).

Nevertheless, we conjecture the following:

Conjecture 1. *The class of binary valid networks is MLLS-reconstructible.*

Note here that invalid networks, where not every reticulation edge is valid, are not level-reconstructible in general. A counter example is given in Figure 3.18.

Ultimately we wish to characterize precisely which networks are reconstructible from their MLLSs. Though this could perhaps be possible by an analogous leaf pair analysis as done in Section 3.4, we quickly reach a large number of cases, with level-2 networks already containing 15 possible shapes. A more efficient methodology will be required to treat such general networks. Considering the level-k generators [13] may perhaps provide an interesting approach to this problem.

REFERENCES

[1] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing tree-child networks from reticulate-edge-deleted subnetworks,* Bulletin of mathematical biology **81**, 3823 (2019).

[2] K. T. Huber, L. van Iersel, V. Moulton, and T. Wu, *How much information is needed to infer reticulate evolutionary histories?* Systematic biology **64**, 102 (2014).

[3] S. J. Willson, *Regular networks can be uniquely constructed from their trees,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 785 (2011).

[4] L. van Iersel and V. Moulton, *Trinets encode tree-child and level-2 phylogenetic networks,* Journal of Mathematical Biology **68**, 1707 (2014).

[5] P. Gambette, K. T. Huber, and S. Kelk, *On the challenge of reconstructing level-1 phylogenetic networks from triplets and clusters,* Journal of Mathematical Biology **74**, 1729 (2017).

[6] L. van Iersel, V. Moulton, E. de Swart, and T. Wu, *Binets: fundamental building blocks for phylogenetic networks,* Bulletin of mathematical biology **79**, 1135 (2017).

[7] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[8] J. Jansson and W.-K. Sung, *Inferring a level-1 phylogenetic network from a dense set of rooted triplets,* Theoretical Computer Science **363**, 60 (2006).

[9] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[10] D. Gusfield and V. Bansal, *A fundamental decomposition theory for phylogenetic networks and incompatible characters,* in *Annual International Conference on Research in Computational Molecular Biology* (Springer, 2005) pp. 217–232.

[11] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions,* SIAM Journal on Computing **10**, 405 (1981).

[12] J. Oldman, T. Wu, L. van Iersel, and V. Moulton, *Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories,* Molecular biology and evolution **33**, 2151 (2016).

[13] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout, *Constructing level-2 phylogenetic networks from triplets,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 667 (2009).

[14] M. Bordewich, K. T. Huber, V. Moulton, and C. Semple, *Recovering normal networks from shortest inter-taxa distance information,* Journal of mathematical biology **77**, 571 (2018).

[15] S. J. Willson, *Properties of normal phylogenetic networks,* Bulletin of mathematical biology **72**, 340 (2010).

4

ON CHERRY-PICKING SEQUENCES AND NETWORK CONTAINMENT

In this chapter, we introduce a new class of networks called the orchard networks. These are networks that may be reduced by repeatedly applying reduction rules on substructures formed by two leaves in the network. We give an encoding for orchard networks that is different from the encoding given in the previous chapter. Instead of characterizing networks from their subnetworks, we show that orchard networks are encoded by 'minimal' cherry-picking sequences that reduce them. We explore the characteristics of such classes, and we also show that orchard networks may be reduced in any order. This leads to a linear-time algorithm to decide if a given network is orchard.

We use cherry-picking sequences to tackle the NETWORK CONTAINMENT *problem: decide whether a network is contained in another network. We show that an orchard network is contained in another orchard network if a sequence for the latter network reduces the former network. Lastly, we show that the converse of the above statement holds for tree-child networks, thereby showing that* NETWORK CONTAINMENT *for tree-child network inputs can be solved by computing cherry picking sequences in linear time. We show that a Python implementation achieves the linear-time theoretical bound in practice.*

4.1. INTRODUCTION

ALTHOUGH the evolutionary history of a set of species may be reticulate, small stretches of DNA (e.g., pieces of DNA coding for protein domains) still evolve mostly tree-like. The network representing the species' evolution must then contain the trees for such pieces of DNA. This leads to the following mathematical problem. For a given network N and a tree T on the same set of species, decide whether N contains T. Solving such a problem gives one way of validating outputs obtained from network building algorithms,

The contents of this chapter have been published in International Conference on Algorithms for Computational Biology **AlCoB 2020**, 93-107 (2020) [1] and in Theoretical Computer Science **856**, 121-150 (2021) [2].

i.e., a way of making sure that the obtained network is biologically plausible. Indeed, a solution to the above problem presents a way of checking that a network contains gene trees from a repository agreed upon by biologists for the given set of species (for more on this, see [3]).

This problem, called TREE CONTAINMENT, is NP-complete for general directed phylogenetic networks [4]. Many have overcome this computational challenge by considering inputs of topologically restricted networks. It was shown initially that TREE CONTAINMENT can be solved in polynomial time for non-binary normal networks, binary tree-child networks, and binary level-k networks [5]. Stronger results have been proven for genetically stable networks (quadratic time: [6]), binary nearly-stable networks (linear time: [7]), and for binary tree-child networks (linear time: [7, 8]).

From a biological and a computational perspective, there is no reason to restrict to inputs of a tree and a network. Indeed, while small stretches of DNA evolve tree-like, it is possible for larger parts of the genome to evolve as a network. In such instances, it is of interest to consider a generalization of TREE CONTAINMENT introduced in [1] called NETWORK CONTAINMENT: For given networks N and N' on the same set of species, decide whether N contains N'.

Computationally, it is natural to wonder whether we can also solve NETWORK CONTAINMENT efficiently in all network classes where TREE CONTAINMENT can be solved efficiently. This question was first posed in the conference paper [1], and an extended version was published as a journal paper [2]. This chapter is based on [2]. In that conference paper, the focus was restricted to semi-binary tree-child networks and all proofs were omitted. In the journal version, we included all proofs, and we broadened the scope to include non-binary networks in the class of so-called orchard networks (which we called *cherry-picking networks)*, which contains the class of tree-child networks. orchard networks are networks that may be reduced to a network on a single leaf by a sequence of reductions on two-leaf structures.

We note that the class of binary orchard networks was also introduced independently by [9] (they were the first to call the class *orchard*). One of the main results of this chapter, that binary orchard networks may be reduced in any order (Theorem 9), was also shown by [9]. The focus of that paper is rather different from this chapter as they focus on reconstructing networks from so-called *ancestral profiles*. Therefore both works have independently come across the same network class in different contexts.

We investigate the correspondence between orchard networks and the sequences that reduce them, and ultimately show that within a particular *reconstructible* class, orchard networks are encoded by their *smallest* cherry-picking sequences (Theorem 10). These reconstructible classes are based on several constructions used to obtain a network from a cherry-picking sequence. Although [10] and [11] mention how one can construct some network from a cherry-picking sequence, no further characterizations for the type of networks that can be obtained from such sequences have been investigated. Here, we reintroduce these constructions as a way to reverse the reduction. As multiple different networks can be reduced by the same cherry-picking sequence, this reversal cannot be unique. Hence, we investigate several construction algorithms corresponding to the different reversals of the reductions. Each construction algorithm then leads to a reconstructible class of networks, for which we can prove relations between contain-

ment and reduction by cherry-picking sequences. We show that within particular classes of orchard networks, if a sequence for a network N reduces another network N', then N' is contained in N (Lemma 33). Unfortunately the converse does not hold (Theorem 12), unless the two input networks are tree-child.

It turns out that the class of tree-child networks is contained in the class of orchard networks, as each tree-child network has a special type of cherry-picking sequence—a *tree-child sequence*—that reduces it. We examine how these sequences can be used to solve NETWORK CONTAINMENT for tree-child networks. In particular, within some orchard classes, we show that a tree-child sequence for a tree-child network N reduces another tree-child network N' if and only if N' is contained in N (Theorem 13). Finally, we provide a linear-time algorithm for NETWORK CONTAINMENT for inputs of tree-child networks (Algorithm 7). We test our implementation on simulated data, and show that even for large data sets (1000 leaves and 1000 reticulations), our software outputs the solution within a tenth of a second.

Structure of the chapter In Section 4.2, we recall all relevant definitions that are not in Chapter 2 and outline how to construct networks from cherry-picking sequences. In Section 4.3, we investigate properties of orchard networks, and show that the order in which cherries are picked does not matter (Theorem 9). Furthermore, we show that networks are unique up to a particular minimal cherry-picking sequence that reduces them, given an order on the set of species. In Section 4.4, we show that if a sequence for a network N reduces another network N', then N' is contained in N (Lemma 32). We also give a counter-example for why the converse does not hold (Theorem 12). In Section 4.5, we restrict our attention to tree-child networks.We show that a sequence for a tree-child network N reduces another network N' if and only if N' is contained in N (Theorem 13). In Section 4.6, we use this characterization in an algorithm for NETWORK CONTAINMENT for tree-child networks, and show that its running time is linear. We also show that, by defining an ordering on the leaves, it is possible to check whether two orchard networks are isomorphic in polynomial-time (Theorem 16). In Section 4.7, we present an efficient implementation for solving NETWORK CONTAINMENT in Python, and show that the theoretical linear running time is achievable in practice. In Section 5.6, we conclude with open problems and future directions for the use of cherry-picking strategies. We also summarize the containment results from this chapter in Table 4.2.

4.2. PRELIMINARIES

We say that an edge is *binary* if the head of the edge is a degree 3 vertex. We call an edge uv an *rr-edge* if u and v are both reticulations, a *tr-edge* if u is a tree vertex and v a reticulation, an *rt-edge* if u is a reticulation and v a tree-vertex, and a *tt-edge* if u and v are both tree-vertices. We call a directed path $u_1u_2\dots u_n$ an *rr-path* if u_i is a reticulation for all $i \in [n] = \{1,\dots,n\}$, an *rtr-path* if u_1 and u_n are reticulations and u_i is a tree vertex for at least one $2 \leq i \leq n-1$, a *trt-path* if u_1 and u_n are tree vertices and u_i is a reticulation for at least one $2 \leq i \leq n-1$, and a *tt-path* if u_i is a tree vertex for all $i \in [n]$.

4.2.1. CHERRY-PICKING SEQUENCES

In this subsection, we introduce cherry-picking sequences and their action on networks. This starts with defining what it means to *reduce* cherries and reticulated cherries from networks, and show that reversing such reductions—called *adding* pairs to networks—can be done in many ways. We show that these additions can be applied to some sequence of ordered pairs of leaves to obtain a network. We impose conditions on the sequences to ensure that these additions are well-defined, and, in doing so, we formally define cherry-picking sequences and orchard networks. See Figure 4.2 for an illustration of the terms defined in this subsection.

REDUCIBLE PAIRS

We may *reduce* cherries and reticulated cherries from a network to obtain a network of smaller size.

Definition 6. Let N be a network and let (x, y) be an ordered pair of leaves. *Reducing* (x, y) in N is the action of

- deleting x and suppressing degree-2 nodes in N if (x, y) is a cherry in N;
- deleting the reticulation edge between the parents of x and y and subsequently suppressing degree-2 nodes, if (x, y) is a reticulated cherry in N;
- doing nothing to N otherwise.

In all cases, the resulting network is denoted $N(x, y)$. We sometimes refer to this as *picking a cherry or pair* (x, y) *from* N. We say that an ordered pair (x, y) *affects* N if $N \neq N(x, y)$.

Given a network N and a sequence of ordered pairs S, we denote by NS the network obtained by repeatedly reducing N with each element of S in order. We say that S reduces N if NS is a network with a single leaf (for any leaf in N), a root, and no other vertices. In particular, we call these networks *single-leaf networks*. We say that a sequence of ordered pairs S *affects* N if every element of S affects N when applied successively.

ADDING PAIRS TO NETWORKS

As each reduction makes a simple change to a network, it is natural to attempt to reverse this change. Such reversals can be done by adding a leaf to obtain a new cherry in the network, or by adding a reticulation edge to create a new reticulated cherry. If the reduction involved the pair (x, y), then we call the reverse action *adding* (x, y) *to the network*. Since we allow for non-binary networks, it is possible to reduce reticulated cherries with a *multi-reticulation* (a reticulation with indegree at least 3). Because of this, there may not be a unique way to add the reticulation edge back: we have the option of choosing an existing reticulation vertex or a newly created reticulation vertex as the head of this reticulation edge.

A similar observation can be made for tree nodes. Just like multi-reticulations, reductions may pick cherries or reticulated cherries that contain *multifurcations* (tree nodes of outdegree more than 2). Here, we have the option of choosing an existing tree vertex or a newly created tree vertex as the tail of the inserted edge.

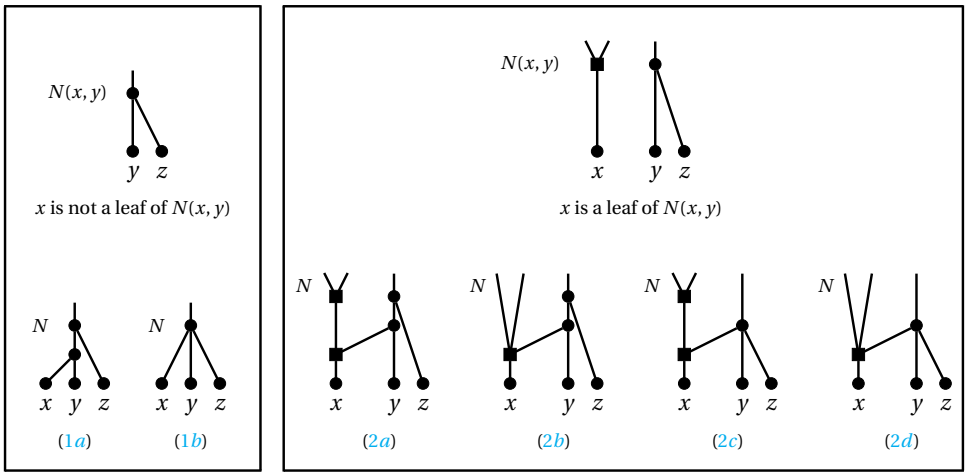


Figure 4.1: The six different constructions for adding a reducible pair (x, y) to a network $N(x, y)$, as in Definition 7. The left and the right boxes show how cherries and reticulated cherries can be added, respectively: The top subgraphs show the part of the network $N(x, y)$ that will be changed by adding (x, y); the bottom subgraphs show the corresponding parts of N for the different constructions. There are more cases, for example when adding a cherry (left box) and the parent of y is a reticulation. In such cases, however, the constructions are the same for both 1a and 1b. Similarly, there are more cases for when adding a reticulated cherry. We only depict these examples, as they showcase each of the different constructions.

With this in mind, there are 6 ways of adding (x, y) to a network: 2 ways of adding cherries and 4 ways of adding reticulated cherries.

Definition 7. Let N be a non-binary network with a reducible pair (x, y). Let p_x and p_y denote the parents of x and y in $N(x, y)$, respectively (note that x and p_x may not be nodes in $N(x, y)$ if (x, y) is a cherry in N). Then we may *add* (x, y) *to* $N(x, y)$ to obtain N by using one of the following six *constructions* (see Figure 4.1):

1. If x is not a leaf in $N(x, y)$ (i.e., if (x, y) is a cherry in N), then add a labelled node x, add a node q directly above y, and add an edge qx.

   (a) Do not contract any edges; or

   (b) If p_y is a tree node, then contract $p_y q$.

2. If x is a leaf in $N(x, y)$ (i.e., if (x, y) is a reticulated cherry in N), then add nodes p, q directly above x, y, respectively, and add an edge qp.

   (a) Do not contract any edges;

   (b) If p_x is a reticulation, then contract $p_x p$;

   (c) If p_y is a tree vertex, then contract $p_y q$; or

   (d) If p_x is a reticulation, contract $p_x p$; and, if p_y is a tree vertex, contract $p_y q$.

Since all tree vertices have indegree-1 and all reticulations have outdegree-1, there are no other ways of adding a reducible pair to a network other than the six ways mentioned above. Note that the constructions 1b, 2b, 2c, and 2d may only be used if the 'if' conditions are satisfied. Also note that the above actions are only well-defined if y is a leaf in $N(x,y)$. In the setting of Definition 7, this is not an issue: since we assume that (x,y) is a reducible pair of N, it is indeed the case that y is a leaf in $N(x,y)$.

On the other hand, if we were to start with any sequence of ordered pairs and sought to construct a network by successively adding ordered pairs backwards through the sequence, the story would be a little different. That is, we may come across a case where, upon trying to add a reducible pair (x,y) to a network, y does not already exist in the network as a leaf. Let $S = S_1S_2\dots S_{|S|} = (x_1,y_1)(x_2,y_2)\dots(x_{|S|},y_{|S|})$ be a sequence of ordered pairs. Starting with a network on a single leaf $y_{|S|}$, we may iteratively add S_i to the network for $i = |S|,|S|-1,\dots,1$ (i.e., backwards through the sequence S), choosing a suitable construction for each ordered pair, to obtain some network. We call this *a network obtained from S*. Now, if y_i was not a leaf in the network when adding S_i, then such a construction would not be well-defined. Fortunately, we can fix this by imposing a simple condition on the sequences. This motivates the following definition.

Definition 8. A *cherry-picking sequence (CPS)* on a set X is a sequence of ordered pairs on distinct elements from X, such that the second coordinate of each ordered pair occurs as a first coordinate in some ordered pair in the rest of the sequence, or as the second coordinate of the last pair.

Returning to the example that we had before, we observe if S was a CPS, then y_i must already have been a leaf in the network when adding $S_i = (x_i,y_i)$. By definition of CPSs, y_i appears as a first coordinate in some ordered pair that appears after S_i, or y_i appears as the second coordinate of the final ordered pair, which implies that the network contains the leaf y_i in both cases when adding $S_i = (x_i,y_i)$. Therefore, this construction is well-defined, and we can always obtain a network from a CPS. This brings us to the definition of an orchard network.

Definition 9. A network on X is an *orchard network* if it can be obtained from some CPS S. Equivalently, an orchard network is a network that can be reduced by some CPS.

See Figure 4.2 for an example of an orchard network with a CPS that reduces it. See Figure 4.3 for examples of networks that are not orchard networks. In particular, single-leaf networks are also orchard networks, since these can be reduced by the empty CPS. By definition, an orchard network with at least two leaves contains either a cherry or a reticulated cherry. Intuitively, reducing these structures returns a network of smaller size that is an orchard network; we may repeatedly reduce cherries and reticulated cherries until the network has been reduced.

A *subsequence* of a CPS refers to any sequence of ordered pairs that can be obtained by deleting some elements from the CPS. Note that a subsequence need not be a CPS. In what follows, we will often have to reduce a network by a subsequence of a CPS. These subsequences are most often the initial parts of the sequence, and hence we introduce notation for them. Let $S = (x_1,y_1)(x_2,y_2)\dots(x_n,y_n)$ be a CPS. For $i \in [n]$, we use the following notations to denote some subsequence of S. The ith ordered pair of S is $S_i =$

$$S = (2,1)(3,2)(3,4)(2,1)(1,4)$$

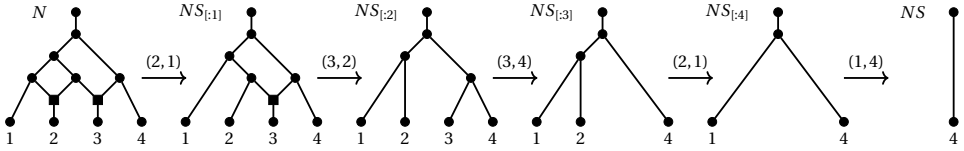


Figure 4.2: A binary orchard network N reduced to leaf 4 by the CPS S. The reduction is shown as a sequence of networks $NS_{[:i]}$ for $i = 0,1,\ldots,5$ from left to right, in which an element of S is applied to the network successively. This sequence is minimal for the network, as every element of the sequence reduces either a cherry or a reticulated cherry of the network. An example of a cherry $(3,4)$ can be seen in the network $NS_{[:2]}$, and a reticulated cherry $(2,1)$ can be seen in the network N. Observe that this sequence is not a tree-child sequence, as the element 2 appears as a first coordinate in S_1 and as a second coordinate in S_2. Constructing a network from a sequence S in the class (1*a*,2*a*) can be seen by moving through the six networks in reverse order (from right to left).

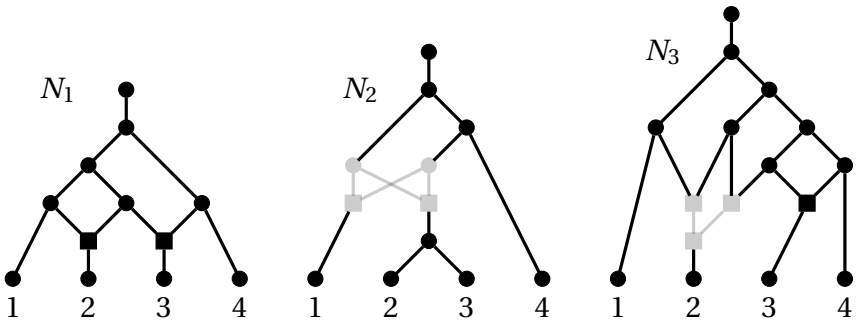


Figure 4.3: an orchard network N_1 and two non-orchard network networks N_2 and N_3. We know from Figure 4.2 that N_1 is an orchard network. N_2 contains a cherry $(2,3)$; upon reducing $(2,3)$, the resulting network does not contain a reducible pair because the crown structure (in gray) prevents cherries and also reticulated cherry. N_3 contains a reticulated cherry $(3,4)$; upon reducing $(3,4)$, the resulting network does not contain a reducible pair because the gray structure prevents 2 from being picked as a cherry or a reticulated cherry.

(x_i, y_i). The first i ordered pairs in S are denoted by $S_{[:i]} = (x_1, y_1)\dots(x_i, y_i)$. The subsequence of S without the first i ordered pairs is denoted by $S_{[i+1:]} = (x_{i+1}, y_{i+1})(x_{i+2}, y_{i+2})\dots(x_n, y_n)$. We let $S_{[:0]}$ denote the empty sequence.

A CPS S is *minimal* for an orchard network N if S reduces N and each ordered pair S_i of S affects $NS_{[:i-1]}$ for all $i \in [|S|]$. In other words, NS is a network on a single leaf, and $NS_{[:i-1]} \neq NS_{[:i]}$ for all $i \in [|S|]$. We often write *a CPS of / for a network N* to refer to a minimal CPS for N.

A *partial CPS* S' of length i is a sequence of ordered pairs such that there exists a CPS S where $S_{[:i]} = S'$. If S and S' are partial CPSs and N is a non-binary network, then applying S and then S' is the same as appending S' to S, denoted SS', and applying the whole sequence. In notation, we write

$$(NS)S' = N(SS'),$$

and hence we denote this network without brackets as NSS'.

Observation 8. *Let N be a non-binary orchard network that can be reduced by a CPS S. Then the network $NS_{[:i]}$ is an orchard network for all $i = 1, \dots, |S|$.*

By choosing a suitable construction, we may obtain an orchard network from any of its minimal CPSs.

Observation 9. *Every non-binary orchard network can be obtained from a minimal CPS that reduces it.*

4.2.2. ORCHARD CLASSES

Using different combinations of the six constructions from Definition 7 can yield different orchard networks from the same CPS. These differences could be due to the nature of the network vertex degrees (binary, semi-binary, non-binary) or due to their topological features (stack-free, tree-child). One way of categorizing these orchard networks is to choose and stay consistent with one particular construction for adding cherries and reticulated cherries to networks. That is, we construct networks from CPSs with a chosen construction A to add cherries and a chosen construction B to add reticulated cherries.

There are two motivations to do so. Firstly, this categorizes orchard networks into classes defined by their topological restrictions. We may specify orchard classes that contain only binary networks, those that contain only semi-binary networks, those without stacks, and many more. Secondly, and more importantly, we can introduce some notion of a correspondence between orchard networks and minimal CPSs that reduce them. Within some orchard classes, it turns out that if a sequence is minimal for two networks, then the networks must be isomorphic.

Definition 10. Let A and B be a cherry construction and a reticulated cherry construction, respectively. We let (A, B) denote the class of all orchard networks that can be obtained from CPSs by using the suitable constructions A or B.

Within the orchard class (A, B), we say that we use the (A, B)-*construction* to obtain orchard networks from CPSs. We write $N \in (A, B)$ or say that N is an (A, B)-orchard to

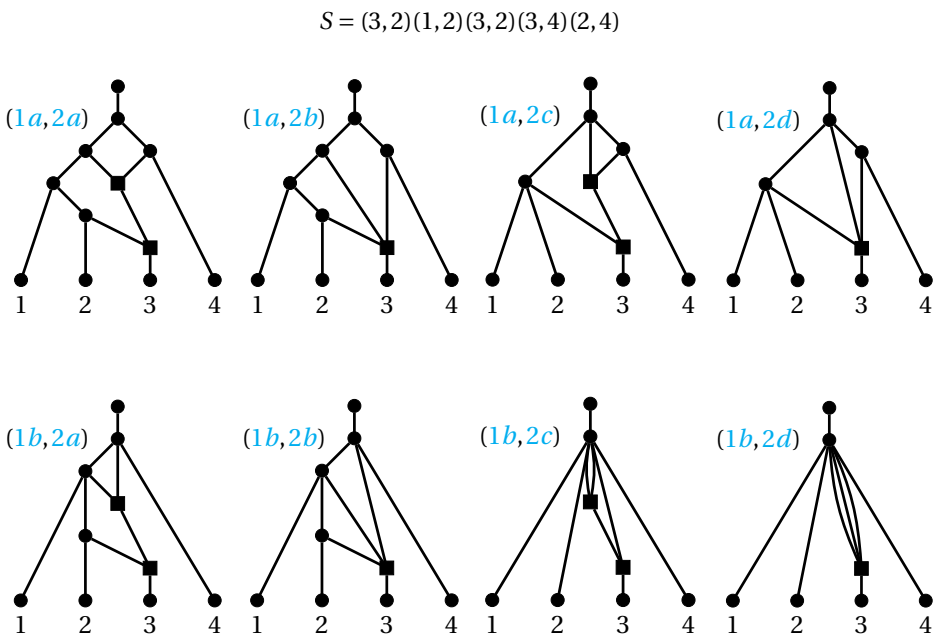


Figure 4.4: A CPS S and the unique networks obtained from it within the eight orchard classes. Observe that the eight networks are distinct.

mean that N is an orchard network in the orchard class (A,B)[1].

Since there are two cherry constructions and four reticulated cherry constructions, there are in total eight orchard classes. For example, the orchard class $(1a,2a)$ contains all binary orchard networks (see Figure 4.2 for an example of obtaining a $(1a,2a)$-class orchard network from a CPS). We note that it is possible obtain the same orchard network from the same CPS within different orchard classes. Indeed, a CPS corresponding to a tree will give the same network in all the orchard classes that use the 1a cherry construction, and the same can be said for orchard classes that use the 1b cherry construction. On the other hand, there do exist CPSs that return distinct networks amongst the different orchard classes; an example of such a CPS is given in Figure 4.4.

Suppose that we are given an orchard network N within an orchard class (A,B), and let S be a minimal CPS that reduces N. To form some notion of correspondence between orchard networks and the sequences that reduce them, we pose the following question: is it always the case that applying the (A,B) construction on S returns the network N? It turns out that this is true only for half of the orchard classes. We start by defining what it means for an orchard class to be *reconstructible*.

Definition 11. an orchard class $C = (A,B)$ is called *reconstructible* if for any two net-

[1] Note that these orchard classes differ from the class of orchard networks that we have mentioned before. The class of orchard networks refers to the collection of all orchard networks. Orchard classes refer to orchard networks that can be obtained by applying certain constructions on some CPS. It will be clear which classes we refer to, either from context or by stating so explicitly.

works $N, N' \in C$ with a common minimal CPS, we have that N and N' are isomorphic.

Since the construction is fixed, each CPS gives rise to a unique network within each of the orchard classes. Then if two distinct networks N and N' have a common minimal CPS S, at most one of these networks, say N, can be constructed from the sequence. This means that although S is a minimal CPS of N', it cannot be used to construct the network N'. Indeed, there does exist some minimal CPS of N' which can be used to construct N'. Reconstructible orchard classes have the nice property that for a given orchard network N, *any* minimal CPS for N can be used to construct N.

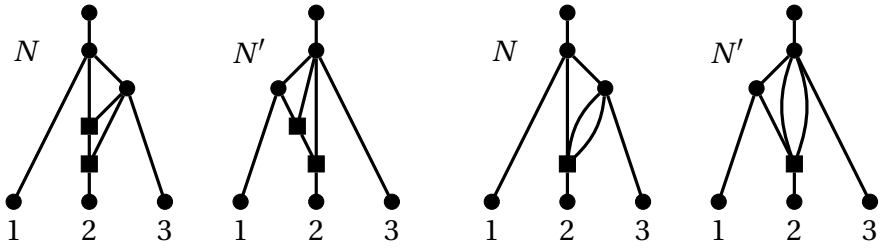
Lemma 20. *Let* $(A,B) \in \{(1a,2a);(1a,2b);(1b,2c);(1b,2d)\}$. *Let* N *be an orchard network in the* (A,B)*-class, and let* (x,y) *be a reducible pair in* N. *Then adding* (x,y) *to* $N(x,y)$ *using the* (A,B) *construction results in* N.

Proof. Observe that these four classes are characterised by the following properties. The networks in $(1a,2a)$ are binary; the networks in $(1a,2b)$ do not contain rr-edges; the networks in $(1b,2c)$ do not contain tt-edges; and the networks in $(1b,2d)$ do not contain rr-edges nor tt-edges. Since N is a network of one of these classes, N must also have these properties. Furthermore, the network $N(x,y)$ also has these properties, since deleting edges and potentially suppressing vertices does not create new vertices, which may subdivide existing rr-edges or tt-edges. This means that upon inserting an edge to the network (as a result of adding a cherry or a reticulated cherry (x,y)), one should either do nothing; contract all rr-edges; contract all tt-edges; or contract all rr-edges and tt-edges, depending on which orchard classes are being considered. Note that these contractions, should they occur, only involve vertices that have just been added as a result of adding the reducible pair, since $N(x,y)$ also has the properties. This is precisely what happens when we add (x,y) back to the network $N(x,y)$ using the respective constructions, and it follows immediately that the constructions defined in these orchard classes returns the original network N. □

Lemma 20 states that four of the eight orchard classes have the property that adding back a reduced pair to the network returns the original network. By applying this lemma in the construction of a network from a CPS, we obtain the following corollary.

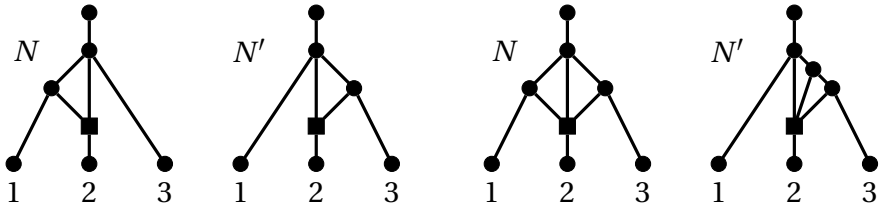
Corollary 3. *Let* $(A,B) \in \{(1a,2a);(1a,2b);(1b,2c);(1b,2d)\}$. *Then* (A,B) *is reconstructible.*

To show that the above lemma and the corollary do not hold for the other four orchard classes, we present two networks with a common minimal CPS for each of the orchard classes in Figure 4.5. Unlike their reconstructible counter-parts, the networks in these four classes can contain both tt-edges and rr-edges whilst also containing multifurcations and multi-reticulations. When constructing networks from CPSs, this allows for a mixture of choosing to contract some tt-edges and some rr-edges, but not all. This can make adding reticulated cherries problematic. Take the $(1a,2c)$ class for example. Since there can exist tt-edges that are binary, we may, in particular, assume that a network in the class contains a reticulated cherry (x,y) where the parent of y is a head of a binary tt-edge e. But this means that upon reducing (x,y) and adding back the reticulated cherry using the 2c construction, we essentially contract this tt-edge, which returns a different network (see Figure 4.6).



(a) $(1a,2c) : (2,3)(2,1)(2,3)(1,3)$

(b) $(1a,2d) : (2,3)(1,3)(2,3)(1,3)$



(c) $(1b,2a) : (2,1)(2,1)(1,3)$

(d) $(1b,2b) : (2,1)(2,1)(2,3)(1,3)$

Figure 4.5: Two distinct networks N and N' that can be reduced by the same minimal CPS for the $(1a,2c)$, $(1a,2d)$), $(1b,2a)$), $(1b,2b)$-classes. The networks obtained by using the respective constructions are N. This means that given a network and a minimum sequence that reduces it, the sequence cannot always be used to construct the original network (let N' be the original network in these four cases).

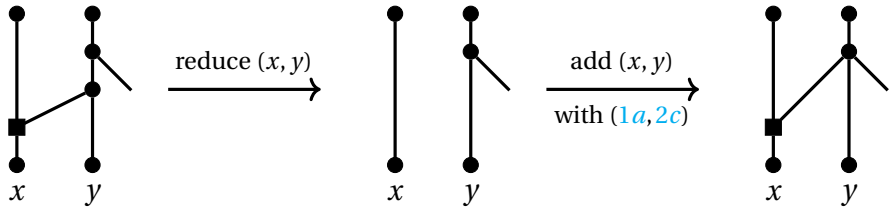


Figure 4.6: Reducing a reticulated cherry (x,y) and adding it back using the construction $(1a,2c)$ can return a different network.

REFINEMENT OF CONSTRUCTED NETWORKS

The six constructions that were introduced in Definition 7 can be rephrased as follows. When adding (x, y) to a network, check if x is a leaf in the network. If x is not a leaf in the network, then add a labelled leaf x and an edge from the (newly added) parent of y to x (add a cherry). If x is a leaf in the network, then add an edge between the newly added parents of y and x (add a reticulated cherry). Decide whether or not to contract some of the edges incident to the parent of x and edges incident to the parent of y. This means that given some CPS S, the binary network N in the $(1a, 2a)$-class constructed from S is a refinement of all networks that can be constructed from S, using any combination of the constructions. This gives the following observation.

Lemma 21. *Given a non-binary orchard network N and a minimal CPS S for N, there exists a binary refinement N_b of N such that S is a minimal CPS for N_b.*

Proof. The unique binary network N_b obtained by using construction $(1a, 2a)$ on S is a refinement of N, and S is a minimal CPS for N_b by definition of this network. □

Finally, the following lemma shows how general refinements of orchard networks (not necessarily binary) are related to the orchard networks.

Lemma 22. *Let N_r be a refinement of a non-binary network N that is an orchard network. Then N is an orchard network, and every minimal CPS of N_r is also a minimal CPS of N.*

Proof. We prove this statement by induction on $|N|$, the number of edges in N. For the base case take the single-leaf network. So suppose that for every network of size at most $|N| - 1$, the claim is true.

Let S be a minimal CPS of N_r, and let $S_1 = (x, y)$ be the first element of S. Since N_r can be obtained from N by refining vertices, it must be the case that S_1 is also a reducible pair in N. Furthermore, if S_1 is a cherry in N_r then S_1 is also a cherry in N; if S_1 is a reticulated cherry in N_r then S_1 is also a reticulated cherry in N. Now it is easy to see that $N_r S_1$ is a refinement of NS_1 Note that $|NS_1| < |N|$ since every reduction reduces the size of the network. The network $N_r S_1$ is an orchard network by Observation 8. By induction hypothesis, NS_1 is an orchard network and every minimal CPS of $N_r S_1$ is also a minimal CPS of NS_1. Then in particular, $S_{[:2]}$ is a minimal CPS of NS_1. It follows then that S is a minimal CPS of N. □

Note that the converse of Lemma 22 does not hold in general. Consider the tree T on three leaves $\{x, y, z\}$ that all share a common parent (the claw graph with a root). Let T_r be a binary refinement of T in which x and y form a cherry. Then the CPS $(y, z)(x, z)$ is minimal for T but not for T_r.

4.3. PROPERTIES OF ORCHARD NETWORKS

IN this section, we investigate properties of orchard networks. First, we continue where we left off in the previous section: we inspect the relation between CPSs and orchard networks. This includes the reticulation number defined by a CPS, changes in the sets of reducible pairs that are ready for picking after picking a pair, and the order in which

we can reduce a network. The last of these allows us to consider distinguishability of two orchard networks by their CPSs. Then, we use this to investigate the relation between embedded networks of an orchard network and its CPSs.

4.3.1. WHY ORCHARD NETWORKS ARE NICE: ORDER DOES NOT MATTER

Lemma 23. *Let S be a minimal length sequence of ordered pairs of leaves that reduces a non-binary network N. Then S is a CPS. Furthermore, $|S| = n+r-1$, where n and r denote the number of leaves and the reticulation number of N, respectively.*

Proof. Suppose for a contradiction that S is not a CPS. Then, there is an $i < |S|$ with $S_i = (x, y)$ such that y is not a first coordinate in any of the elements of $S_{[i+1:]}$ or the second coordinate of $S_{|S|}$. This means y cannot be a leaf in $NS_{[:i-1]}$ (if it were, then S would not reduce N). This implies $NS_{[:i-1]} = NS_{[:i]}$, and there is a shorter sequence $S_{[:i-1]}S_{[i+1:]}$ that reduces N, a contradiction. We conclude that S is a CPS.

We now prove the second part of the lemma. Let $S_i = (x, y)$. We first construct a binary network M from S using the (1*a*,2*a*) construction. Upon constructing $MS_{[:i-1]}$ from $MS_{[:i]}$, a new leaf x is added if x is not a leaf in $MS_{[:i]}$, and a reticulation is added otherwise. By Lemma 21, M is a binary refinement of N, and therefore N has the same leaf set and it has the same reticulation number as that of M. Since S is a minimal CPS for N, it follows that $|S| = n + r - 1$. □

Definition 12. Let N be a non-binary network. Denote with $\mathscr{C}_c(N)$ the set of cherries of N, and with $\mathscr{C}_r(N)$ the set of reticulated cherries of N. The set of all reducible pairs is denoted $\mathscr{C}(N) = \mathscr{C}_c(N) \cup \mathscr{C}_r(N)$.

The following lemma states that all new reducible pairs after picking a pair (x, y) must involve either x or y.

Lemma 24. *Let N be a non-binary network on a taxa set X, and let (x, y) be a reducible pair of N. Then we have the following inclusion:*

$$\mathscr{C}(N(x,y)) \setminus \mathscr{C}(N) \subseteq \big(\{x,y\} \times X\big) \cup \big(X \times \{x,y\}\big).$$

Proof. Note that the LHS of the containment relation represents the reducible pairs in $N(x, y)$ that were not present in N. Suppose, for contradiction, that this set contains a pair (z, w) not involving x or y. Then, this pair is not reducible in N, but it is in $N(x, y)$. Adding the pair (x, y) back into $N(x, y)$ may only subdivide the pendant edges leading to x and y. This implies that this action will not change the fact that z and w form a reducible pair. Therefore, (z, w) is a reducible pair in N as well, a contradiction. Hence, all new cherries and reticulated cherries of $N(x, y)$ involve x or y. □

We also have similar inclusions for looking at reducible pairs in the original network that are not reducible pairs in the new network. Roughly speaking, the following lemma states that reducing a network by the element (x, y) preserves the other reducible pairs.

Lemma 25. *Let N be a network on X, and (x, y) a reducible pair of N. Then, if N is non-binary, we have the inclusion $\mathscr{C}(N) \setminus \mathscr{C}(N(x, y)) \subseteq \{x\} \times X \cup X \times \{x\}$, and in particular*

$$\mathscr{C}_c(N) \setminus \mathscr{C}_c(N(x,y)) \subseteq \{x\} \times X \cup X \times \{x\},$$

and

$$\mathscr{C}_r(N) \setminus \mathscr{C}_r(N(x,y)) \subseteq \{x\} \times X.$$

If N is semi-binary, the inclusions can be sharpened to $\mathscr{C}(N) \setminus \mathscr{C}(N(x,y)) \subseteq \{(x,y),(y,x)\}$, with

$$\mathscr{C}_c(N) \setminus \mathscr{C}_c(N(x,y)) \subseteq \{(x,y),(y,x)\},$$

and

$$\mathscr{C}_r(N) \setminus \mathscr{C}_r(N(x,y)) \subseteq \{(x,y)\}.$$

Proof. Let N be a non-binary network and let (x,y) be a reducible pair of N. Let (z,w) be a reducible pair in N where $z \neq x$ and $w \neq x$. We claim that (z,w) must also be a reducible pair in $N(x,y)$. Suppose for a contradiction that it was not. Adding the pair (x,y) may only subdivide the pendant edges leading to x and y; this action will not change the fact that (z,w) does not form a reducible pair. But this means that (z,w) is not a reducible pair in N, which is a contradiction. Therefore, (z,w) must also be a reducible pair in $N(x,y)$. It follows that every reducible pair in N that is not a reducible pair in $N(x,y)$ must contain the element x. Note that since semi-binary networks are non-binary, this fact also holds for semi-binary networks.

If (x,y) is a cherry in N, then, since the network is non-binary, it is possible for x to form a cherry with another leaf, say z. Then (x,z) and (z,x) are both reducible pairs in N, while they are not reducible pairs in $N(x,y)$ since x is not a leaf in $N(x,y)$. On the other hand, if (x,y) is a reticulated cherry in N, then x may only appear as a first coordinate in a reducible pair of N. Therefore the inclusions for non-binary networks follow.

Suppose now that N is semi-binary. As stated in the first paragraph of this proof, every reducible pair in N that is not a reducible pair in $N(x,y)$ must contain the element x. If (x,y) is a cherry in N, then x may only be in a cherry (and reducible pair) with the leaf y. If (x,y) is a reticulated cherry in N, then the only reticulated cherry involving the parent p_y of y is (x,y). All other reticulated cherries that contain x do not involve p_y, as p_y is of outdegree-2: this implies all reticulated cherries in N involving x (as the first coordinate) is still a reducible pair in $N(x,y)$. The inclusions for the semi-binary networks then follow. □

We now start our investigation into the order in which pairs can be reduced. We start with a lemma that implies a cherry on two leaves x and y can be reduced either as (x,y) or as (y,x). Then we show that reducing an arbitrary pair in an orchard network gives a new orchard network.

Lemma 26. *Let S be a minimal CPS for a non-binary orchard network N and suppose $S_i = (x,y)$ reduces a cherry when applying the sequence. Let z and w be distinct leaves (not necessarily different from x and y) that have a common parent, equal to the parent of x and y. Let S' be the sequence $S_{[i+1:]}$ where each occurrence of z is replaced by x. Then $S_{[:i-1]}(z,w)S'$ is a minimal CPS for N.*

Proof. Because (x,y) forms a cherry in $N' = NS_{[:i-1]}$, and x and y share their parents with z and w, the reduced network $N'(x,y)$ is equal to the network $N'(z,w)$ when z is replaced by x. Hence, if we switch the roles of x and z in the remaining part of the sequence, the result after reduction by both sequences is the same modulo the $x \leftrightarrow z$ replacement. □

Lemma 27. *Let N be a non-binary orchard network that can be reduced by a minimal CPS $S = S_1, S_2, \ldots, S_{|S|}$ such that $S_2 \in \mathscr{C}(N)$. Then NS_2 is an orchard network.*

Proof. Note that $S_1, S_2 \in \mathscr{C}(N)$ by assumption. We distinguish several cases and prove in every case that NS_2 is an orchard network.

- **The leaves in S_1 and S_2 are the same.** Then either $S_1 = S_2$, or $S_1 = (x, y)$ and $S_2 = (y, x)$ for some pair of leaves x, y. In the first case $NS_2 = NS_1$, which is an orchard network. In the second case, as (x, y) and (y, x) are both present in N, (x, y) must be a cherry in N. This means that $NS_1S_2 = NS_1$, and thus S is not a minimal CPS for N. This case is not possible.

Let $S_1 := (x, y)$.

- **The pairs S_1 and S_2 have exactly one leaf in common.**
  - **$S_2 = (x, z)$.** The common leaf x is below the reticulation common to the two reticulated cherries. Applying S_1 and S_2 in any order removes these two reticulation edges, so clearly $NS_1S_2 = NS_2S_1$. By Observation 8, NS_1S_2 is an orchard network. This implies NS_2S_1 is an orchard network and, therefore, that NS_2 is also an orchard network.
  - **$S_2 = (z, x)$.** Observe first that (x, y) cannot form a reticulated cherry, as otherwise the first coordinate of every reducible pair that involves x is x, which contradicts our assumption that $S_2 = (z, x) \in \mathscr{C}(N)$. Therefore (x, y) must be a cherry. Then the network $NS_1 = N(x, y)$ does not have the leaf x, which implies that $S_2 = (z, x)$ is not a reducible pair of NS_1. This contradicts the fact that S was a minimal CPS for N, and therefore this case is not possible.
  - **$S_2 = (y, z)$.** The two possibilities for this case are either that x, y, z all share the same parent, or that (x, y) form a reticulated cherry in N and z shares a common parent with y. In the former case, NS_1S_2 is the orchard network obtained by deleting the leaves x and y and suppressing all degree-2 vertices. We obtain the same orchard network by picking the cherries $S_2 = (y, z)$ and (x, z) in succession, that is, $NS_2(x, z) = NS_1S_2$. This implies that NS_2 is also an orchard network. A similar argument can be done for the reticulated cherry case—it is easy to see that $NS_2(x, z) = NS_1S_2$.
  - **$S_2 = (z, y)$.** This is the case where either y and z share a common parent, or (z, y) forms a reticulated cherry. In both of these cases, the leaf x could share a common parent with y, or (x, y) could be a reticulated cherry (there are in total 4 possible cases). In all cases, reducing N by S_1 first or by S_2 first has no real difference, and so $NS_1S_2 = NS_2S_1$. For the same reason as before, NS_2 is an orchard network.

- **The pairs S_1 and S_2 have no leaf in common.** Then obviously, S_1 and S_2 independently remove edges in N, not influenced by the order of S_1 and S_2. Hence we get $NS_1S_2 = NS_2S_1$ and for the same reason as before, NS_2 is an orchard network.

In all cases, we have concluded that NS_2 is an orchard network, so the result follows. □

Lemma 28. *Let N be a non-binary network, and $(x,y) \in \mathscr{C}(N)$. Then, there exists a minimal CPS S for N such that $S_i = (x,y)$ or $S_i = (y,x)$ for some i, and (x,y) is reducible until that point, i.e., $(x,y) \in \mathscr{C}(NS_{[:j]})$ for all $j < i$.*

Proof. Let S be a minimal CPS for N. If S contains (x,y) or (y,x) as S_i, and (x,y) is a reducible pair in $NS_{[:j]}$ for all $j < i$, we are done, so assume that this is not the case. Let $i > 0$ be minimal such that $(x,y) \notin \mathscr{C}(NS_{[:i]})$. Then $S_i = (x,z)$ or $S_i = (y,z)$ for some $z \neq x,y$ by Lemma 25. Because $(x,y) \in \mathscr{C}(NS_{[:i-1]})$, $(x,y) \notin \mathscr{C}(NS_{[:i]})$, and the second element of S_i is z, (y,z) must form a cherry in $NS_{[:i-1]}$.

First, suppose that (x,y) forms a reticulated cherry in $NS_{[:i-1]}$, and that $S_i = (x,z)$. In that case, $NS_{[:i]} = NS_{[:i-1]}(x,y)$, so replacing S_i with (x,y) in S gives a new minimal CPS for N that contains (x,y). Next, suppose that (x,y) forms a reticulated cherry in $NS_{[:i-1]}$, and that $S_i = (y,z)$. Then, upon switching the roles of y and z, we have $NS_{[:i]} = NS_{[:i-1]}(z,y)$. Letting S' denote the sequence $S_{[i:]}$ where each occurrence of z is replaced by y, we obtain a minimal CPS $S^{new} = S_{[:i-1]}(z,y)S'$ for N. In this sequence, we have that the minimal value $k > 0$ for which $(x,y) \notin \mathscr{C}(NS^{new}_{[:k]})$ satisfies $k > i$. We may repeat this until we enter the first case; such a process must terminate as the length of S is finite. On the other hand if (x,y) forms a cherry in $NS_{[:i-1]}$, then x, y and z share a common parent. Therefore, by Lemma 26, there is a minimal CPS for N that starts with $S_{[:i-1]}(x,y)$. □

Proposition 1. *Let N be a non-binary orchard network with $c \in \mathscr{C}(N)$. Then Nc is an orchard network. That is, there exists a CPS S such that cS is a CPS reducing N.*

Proof. Let $c = (x,y)$. By Lemma 28, there is a CPS S' for N that contains either (x,y) or (y,x), and (x,y) is reducible until that point in the sequence. If $S'_1 = (x,y)$, then set $S := S'_{[2:]}$ and we are done. Now suppose S'_1 is not equal to (x,y). Note that there must be a smallest $i \geq 1$ with $S'_i = (x,y)$ or $S'_i = (y,x)$.

- **Suppose $S'_i = (x,y)$.** Recall that we have $(x,y) \in \mathscr{C}(NS'_{[:j]})$ for all $j < i$. Hence, by applying Lemma 27 i times, $N(x,y)$ is an orchard network.

- **Suppose $S'_i = (y,x)$.** Again, we have $(x,y) \in \mathscr{C}(NS'_{[:j]})$ for all $j < i$. Hence, $NS'_{[:i-1]}$ has both reducible pairs (x,y) and (y,x), and it must contain the cherry (x,y). By Lemma 26 there is a CPS of N starting with $S'_{[:i-1]}(x,y)$. Redefining S' as this sequence, we are in the previous case and thus $N(x,y)$ is an orchard network.

We conclude that Nc is an orchard network. □

The following theorem is a corollary of the previous proposition. It essentially states that a network can be cherry picked in any order.

Theorem 9. *Let N be a non-binary orchard network, and S a partial CPS. If in each step of the reduction of N by S, the network is changed, then there exists a minimal CPS S' starting with S that reduces N.*

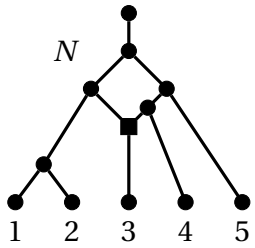


Figure 4.7: The smallest CPS for this network is $(1,2)(3,2)(3,4)(4,5)(2,5)$. Initially we have the choice of picking either $(1,2)$, $(2,1)$, or $(3,4)$. For the smallest CPS we pick the smallest reducible pair $(1,2)$.

4.3.2. DISTINGUISHABILITY

By Theorem 9, any order of picking reducible pairs gives a minimal CPS for an orchard network. This inherently implies that for a given orchard network, there could be many CPSs that reduce it. However, given a class (A,B), every CPS uniquely constructs an orchard network in that class by Lemma 21.

Remark 1. *Within an orchard class, exactly one orchard network can be constructed for each CPS. On the other hand, an orchard network can have more than one minimal CPS that reduces it.*

While this remark holds true for all eight of the orchard classes, only the classes that are reconstructible are interesting to examine. The aim of this subsection is to set up some distinguishability notion of orchard networks using their minimal CPSs. That is, we would like to encode each orchard network by delegating one of its minimal CPSs to be its representative, such that the sequence can be used to reconstruct the orchard network. Since there could be more than one orchard network that can be reduced by the same minimal CPS within orchard classes that are not reconstructible, it makes no sense to consider these classes. Therefore, we define a distinguishability notion only for the classes that are reconstructible.

Within a reconstructible orchard class, each network can have many minimal CPSs that reduce it by Remark 1. To choose a representative from these minimal CPSs, we introduce an ordering on the CPSs. Doing so allows us to prescribe a unique *smallest* CPS to each orchard network. So let us take an arbitrary ordering on the leaves, and let us define a lexicographical ordering on the reducible pairs as follows. We say that $(a,b) < (c,d)$ if and only if $a < c$ or if $a = c$ and $b < d$. We naturally extend this ordering to minimal CPSs. Let S and S' be CPSs such that $|S| \neq |S'|$. If $|S| < |S'|$, then $S < S'$—this ensures the smallest CPS is minimal. Now suppose $|S| = |S'|$ and let i be the smallest index such that $S_i \neq S'_i$. If no such i exists, then $S = S'$; otherwise, $S < S'$ if and only if $S_i < S'_i$.

By Theorem 9, we may pick an orchard network in any order. We define a *smallest* CPS as one that is obtained by picking the smallest reducible pair at each iteration (see Figure 4.7). Such a sequence is naturally a minimal CPS. By the following theorem, distinguishing two orchard networks of the same reconstructible class comes down to finding their smallest CPS and checking whether these are the same.

Theorem 10. *Suppose we are given an ordering on the taxa set X. Every orchard network on X has a unique smallest CPS. In particular within a reconstructible orchard class, these CPSs can be used to reconstruct the orchard network. Every CPS can be used to construct a unique orchard network within each of the eight orchard classes.*

Proof. Let N be an orchard network on X. Since we have a total ordering on the cherries of N, we have that if there exists a smallest CPS then it is unique. Furthermore, we know that a smallest CPS exists: simply pick a smallest reducible pair at every iteration. Therefore every orchard network on X has a unique smallest CPS. Within reconstructible orchard classes, no two networks have the same minimal CPSs. It then follows that a smallest CPS for a network can be used to construct said network.

By Remark 1, we have that every CPS gives rise to a unique orchard network. □

The following corollary is a direct consequence of Theorem 10.

Corollary 4. *Suppose we are given an ordering on the taxa set X. Within a reconstructible orchard class, two orchard networks on X are isomorphic if and only if they have the same smallest CPS.*

This leads to a polynomial-time algorithm for checking whether two orchard networks of a reconstructible orchard class on the same set of taxa are isomorphic, which we describe in Section 4.6.

4.4. REDUCTION AND CONTAINMENT

In this section, we prove that, within reconstructible orchard classes, the reduction of a network by a CPS for another network implies 'containment' of the former in the latter. We also show that the converse does not always hold: containment of a network N' in another network N does not imply that there exists a minimal CPS of N that reduces N'.

4.4.1. REDUCTION IMPLIES CONTAINMENT

Let N and N' be two non-binary networks on X and X', respectively, where $X' \subseteq X$. Recall that N' is a subnetwork of N if N' can be obtained by deleting reticulation edges from N and cleaning up the resulting graph with respect to X'. Recall also that N' is contained in N if some refinement of N is a subnetwork of N.

In what follows, in settings where we consider whether N' is a subnetwork of/contained in N, we informally refer to N as the larger network and N' as the smaller network. Note that the notions of a network being a subnetwork and a network being contained are not always synonymous. If N' is a subnetwork of N, then N' is contained in N. However, if N' is contained in N, then it does not immediately follow that N' is a subnetwork of N. They are synonymous when the smaller network (i.e., N') is binary. The following lemma shows that for any non-binary network (not necessarily an orchard network), the network obtained by reducing a pair is a subnetwork of the original network.

Lemma 29. *Let N be a non-binary network, and c a pair of leaves. Then Nc is a subnetwork of N.*

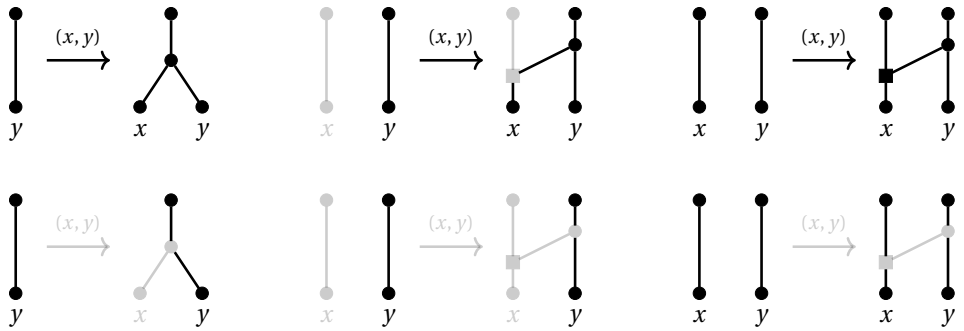


Figure 4.8: The visualization of the three cases in Lemma 30 (each column corresponds to a case). The six different cases for adding a pair (x, y) to a network N, and possibly adding (x, y) to a subnetwork N' of N. N is indicated by the black and gray nodes and edges; N' is indicated by only the black nodes and edges. Only the parts of the networks that will be affected by the addition of (x, y) is shown here. The part to the left of each arrow represents the networks before the addition of (x, y); the part to the right of each arrow represents the networks after the addition of (x, y). A black arrow indicates that (x, y) is added also to N'; a gray arrow indicates that (x, y) will not be added to N'. The embedding of the black network into the black and gray network show that in any of the six cases, regardless of whether (x, y) is added to N' or not, the resulting network is still a subnetwork of the other.

Proof. For the embedding of Nc into N, note that there is a natural map of the nodes of Nc to the nodes of N. Each edge of N, corresponds naturally to an edge of Nc, or is part of a path (of two edges) if an endpoint got suppressed. These mappings form an embedding of Nc into N, as the mapping of the nodes is respected, and no edge of Nc is part of more than one corresponding path of N. □

To show the relation between subsequences and containment, we first focus on the binary orchard class, (1*a*, 2*a*), for which the definitions of subnetwork and containment are synonymous.

SUBNETWORKS

Intuitively, when a CPS S for a binary network N also reduces another binary network N', the embedding of the network N' in N can be found as follows. Reconstruct the network N from S, and annotate the edges used by N' in the process. Let S' denote the CPS of ordered pairs in S that is used in the reduction of N'. Let $S_i = (x, y)$ be an ordered pair that appears in S'. Then in $NS_{[:i]}$, label the paths $p_y y$ and $p_y x$ as 'used'. Upon reconstructing N, the embedding of N' into N can be seen as the subnetwork of N which uses all labelled edges.

Lemma 30. *Let N and N' be binary networks, and $c = (x, y)$ a reducible pair in N and N'. If $N'c$ is a subnetwork of Nc, then N' is a subnetwork of N.*

Proof. First note that there are three cases: neither Nc nor $N'c$ contains x; only Nc contains x; or both Nc and $N'c$ contain x. To find an embedding ι of N' into N, we extend the embedding ι_c of $N'c$ into Nc in each of these cases. We denote the natural embeddings of $N'c$ into N' and of Nc into N by h' and h (see Figure 4.9). We now split into three cases (see Figure 4.8).

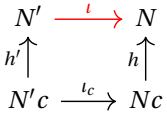


Figure 4.9: The commutative diagram for the networks N, N', Nc, and $N'c$ as in the setting of Lemma 30. Given the maps h, h', and ι_c, we wish to produce the map ι.

- **The leaf x is in neither Nc nor $N'c$.** Let p and p' be the parent of y in N and N', and g and g' the grandparent of y in N and N'. An embedding ι can be constructed from ι_c by setting $\iota(e) = h(\iota_c(h'^{-1}e))$ for all edges of N' other than $g'p'$, $p'y$, and $p'x$—h' maps each edge to a path of length one, except for the edge incident to y. Then, we map the remaining edges by setting $\iota(g'p') = h(\iota(h'^{-1}(g')y)) \setminus \{py\}$, $\iota(p'x) = px$, and $\iota(p'y) = py$—the node $h'^{-1}(g')$ is well defined because the embedding h' is injective on the nodes, and there are exactly two more nodes in N' than in $N'c$, which cannot be the image of any node as the edge incident to x is not part of the embedding. This mapping gives an embedding, because the corresponding node mapping is still injective, no edge is in more than one image-path of an edge, and the endpoint relations are respected.

- **The leaf x is only in Nc.** This case is almost the same as the previous case, except that $\iota(p'x) = p_y p_x x$. The only edge of N that may be in the ι-image of more than one edge, is $p_x x$. However, the only edge in Nc that maps to $p_x x$ is the edge incident to x, and this edge is not in the image of ι_c. Hence, as the image of each edge other than $g'p'$, $p'y$, and $p'x$ is defined by $\iota(e) = h(\iota_c(h'^{-1}e))$, no other edge of N' is mapped to a path of N that contains $p_x x$. Furthermore, the endpoint relations are still respected, so the map ι is an embedding of N' into N.

- **The leaf x is in both Nc and $N'c$.** Let p_x and p'_x be the parent of x, p_y and p'_y the parent of y, $g_x \neq p_y$ and $g'_x \neq p'_y$ the other grandparent of x, and g_y and g'_y the grandparent of y in N and N'. An embedding ι can be constructed from ι_c by setting $\iota(e) = h(\iota_c(h'^{-1}e))$ for all edges of N' other than $g'_y p'_y$, $p'_y y$, $p'_y p'_x$, $g'_x p'_x$ and $p'_x x$—h' maps each edge to a path of length one, except for the edges incident to x and y. Then, we map the remaining edges by setting $\iota(g'_y p'_y) = h(\iota(h'^{-1}(g'_y)y)) \setminus \{py\}$, $\iota(p'_y y) = p_y y$, $\iota(p'_y p'_x) = p_y p_x$, $\iota(g'_x p'_x) = h(\iota(h'^{-1}(g'_x)x)) \setminus \{p_x x\}$, and $\iota(p'_x x) = p_x x$—the nodes $h'^{-1}(g'_x)$ and $h'^{-1}(g'_y)$ are well defined because the embedding h' is injective on the nodes, and there are exactly two more nodes in N' than in $N'c$, which cannot be the image of any node as the reticulation edge $p'_y p'_x$ is not part of the embedding. This mapping gives an embedding, because the corresponding node mapping is still injective, no edge is in more than one image-path of an edge, and the endpoint relations are respected.

□

Lemma 31. *Let N and N' be binary orchard networks on taxa set X and $X' \subseteq X$ respectively, and suppose that a CPS S reduces both N and N', such that all elements of S that affect N' also affect N. Then N' is a subnetwork of N (see Figure 4.10).*

$$S = (2,1)(3,2)(3,4)(2,1)(1,4)$$

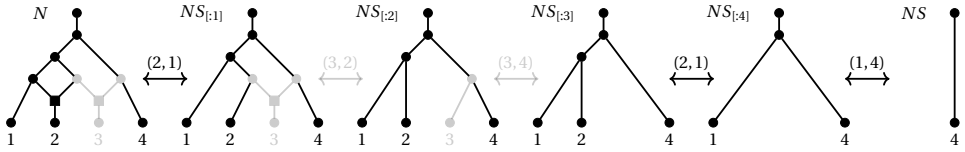


Figure 4.10: A visualization of Lemma 31. The binary orchard network N from Figure 4.2 (gray and black), together with one of its minimal CPSs S. The subnetwork of N (black) is also reduced by S, and the embedding can be constructed by building both networks simultaneously and keeping track of the edges added by the pairs that change the subnetwork (black pairs and arrows).

Proof. Let S' denote the CPS of ordered pairs in S such that every ordered pair in S' is used in the reduction of N'. Due to this, we have that for every $i \in \{1,\dots,|S'|\}$ there exists a $\pi(i) \in \{1,\dots,|S|\}$ such that $S'_i = S_{\pi(i)}$. By definition of S', $\pi : \{1,\dots,|S'|\} \to \{1,\dots,|S|\}$ is a strictly increasing function (i.e., if $i < j$ then $\pi(i) < \pi(j)$). Recall that $S'_{[:i]}$ denotes the sequence obtained by taking the first i ordered pairs from S'. We show by induction on i, for $i = |S'|,\dots,0$, that the orchard network $N'S'_{[:i]}$ is a subnetwork of the orchard network $NS_{[:\pi(i)]}$, where $N'S'_{[:0]} = N'$, and $\pi(0) = \pi(1) - 1$.

For the base case, we prove the claim for $i = |S'|$. Let $S'_{|S'|} = (x, y)$. Then $N'S'$ is the tree with one leaf y. Since (x, y) affects $NS_{[:\pi(|S'|)-1]}$, the network $NS_{[:\pi(|S'|)]}$ must still contain y. As there is a path from the root to any leaf in a phylogenetic network, $N'S' = N'S'_{[:|S'|]}$ is a subnetwork of $NS_{[:\pi(|S'|)]}$.

So now assume $0 \le i < |S'|$ and suppose we have proven that $N'S'_{[:j]}$ is a subnetwork of $NS_{[:\pi(j)]}$ for every $j > i$. As $i \ge 0$, there is an element $S'_{i+1} = S_{\pi(i+1)}$ in each sequence, and $N'S'_{[:i+1]}$ is a subnetwork of $NS_{[:\pi(i+1)]}$ by the induction hypothesis. By Lemma 30, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i+1)-1]}$ because $S'_{i+1} = S_{\pi(i+1)}$ acts on both networks. Applying Lemma 29 to $N'S'_{[:i]}$, $NS_{[:j]}$ and $NS_{[:j+1]}$ for all $j = \pi(i+1)-1,\dots,\pi(i)$, we get that $N'S'_{[:i]}$ is a subnetwork of $NS_{[:j]}$ for all $j = \pi(i),\dots,\pi(i+1)-1$. Hence, in particular, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i)]}$.

Therefore, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i)]}$ for all $i \in \{0,\dots,|S'|\}$. In particular, $N' = N'S'_{[:0]}$ is a subnetwork of $N = NS_{[:0]}$. □

Note that Lemma 31 was only shown for the class of binary orchard networks ((1*a*,2*a*) orchard class). This result generalizes to non-binary orchard networks if we consider the notion of containment instead of subnetwork.

CONTAINMENT RESULTS

Recall that a network N' is contained in another network N if there exists a refinement of N' that is a subnetwork of N. We first show a result that follows immediately from Lemma 31 given that the larger network is binary.

Theorem 11. *Let N be a binary orchard network, and N' a non-binary orchard network on X and $X' \subseteq X$, respectively. If a minimal CPS S for N also reduces N', then N' is contained in N.*

Proof. Let S' be the subsequence of S consisting of the elements that affect N', and let N'_b be the unique binary network corresponding to S'. Then, by Lemma 21, N'_b is a binary refinement of N' with minimal CPS S'. As N is the binary network corresponding to S, N'_b is a subnetwork of N by Lemma 31. Hence, N' is contained in N. □

Unfortunately for two general non-binary orchard networks, an analogue of Lemma 31 is not possible to obtain. For example, the two networks of the orchard class (1*a*, 2*c*) in Figure 4.5 are not contained in one another, yet there is a common CPS that reduces both networks. Therefore, we need to use the more relaxed notion of containment, rather than subnetwork.

Lemma 32. *Let C be a reconstructible orchard class. Let N and N' be networks in C on taxa set X and $X' \subseteq X$, respectively. Then N' is contained in N if and only if there exist binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b.*

Proof. See Figure 4.11 for a visualization of the proof. Suppose first that N' is contained in N. Then there exists a refinement N'_f of N' that is a subnetwork of N. There exists an embedding ι of N'_f into N.

Observe that every reticulation vertex in N'_f is mapped to a reticulation vertex in N of the same or higher degree. Let r' be a reticulation vertex in N'_f, such that $\iota(r') = r$ for some reticulation vertex r in N. Let $e'_1, \ldots, e'_a$ denote all incoming edges of r', and let $e_1, \ldots, e_b$ denote all incoming edges of r where $a \leq b$, such that the paths $\iota(e'_i)$ contains the edge e_i for all $i \in [a]$. Resolve r' as a single path of reticulation vertices $r'_1 \cdots r'_{a-1}$ such that the edge e'_1 is incident to r'_1, and the edges e'_{i+1} are incident to r'_i for all $i \in [a-1]$. Resolve r as a single path of reticulation vertices $r_1 \cdots r_{b-1}$, such that the edge e_1 is incident to r_1, and the edges e_{i+1} are incident to r_i for all $i \in [b-1]$. We now show that N'_f with r' refined in this manner is a subnetwork of N with r refined in this manner. We do this by altering the mapping ι as follows. The edges e'_i are still mapped to the same paths containing the edges e_i for $i \in [a]$. The reticulation edges $r'_i r'_{i+1}$ are mapped to $r_i r_{i+1}$ for $i \in [a-2]$. Let c denote the child of r' in N'_f. The edge $r'_{a-1}c$ is mapped to the path from r_{a-1} to $\iota(c)$. All other mappings remain unchanged.

A similar observation can be made for tree vertices. Now, observe that binary refinements of these orchard networks are obtained by either resolving all multifurcations, all multi-reticulations, or both (depending on C). Then we may apply the above to all multifurcations / multi-reticulations as needed to obtain binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b.

Now suppose that there exist binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b. We note that N can be obtained from N_b by contracting all tt-edges, all rr-edges, or both depending on if N is an orchard network of class (1*a*, 2*b*), (1*b*, 2*c*), or (1*b*, 2*d*). Now, if $N_b = N$, then we are done. So suppose that N is not a binary network.

The plan is to contract the edges of N_b to obtain N. In the process, we choose to either contract or not contract 'corresponding' edges in N'_b, ensuring at each step that the resulting network is a subnetwork of some refinement of the main network. We start by looking at contracting the rr-edges of N_b. We claim that contracting all rr-edges in N'_b that do not map to an rtr-path gives a refinement of N' that is a subnetwork of N.

Let e be an edge in N'_b. If e is an rr-edge, then it is mapped to some path P_e in N_b. Since reticulation vertices are mapped to reticulation vertices in the embedding, the path P_e is an rr-path or an rtr-path. If P_e is an rr-path, then contract all edges of P_e in N_b. Contract e in N'_b. It is easy to see that by mapping the reticulation vertex obtained by contracting e to the reticulation vertex obtained by contracting P_e, we may extend the embedding of N'_b into N_b in a natural manner, thereby showing that the newly obtained network is a subnetwork of the other. Now suppose that P_e was an rtr-path. Then we contract all rr-edges contained in P_e. Observe that N'_b is still embedded in this newly obtained graph, since we may extend the original embedding by simply changing the mapping of the edge e to the newly contracted rtr-path. On the other hand, suppose that e was not an rr-edge. If it is mapped to a path containing rr-edges in N_b, then we may simply contract the rr-edges of the path, and update the embedding by changing the mapping of the edge to the newly contracted path. Finally suppose that there exists an rr-edge in N_b that is not used in the embedding. Then contracting such an edge has no effect on the embedding.

Similarly, contracting all tt-edges in N'_b that do not map to a path containing a trt-path gives a refinement of N' that is a subnetwork of N.

We now obtain a sequence of networks $N'_b = N'_0, N'_1, \ldots, N'_k$ and $N_b = N_0, N_1, \ldots, N_k = N$ such that N'_i is a subnetwork of N_i for all $i \in [k]$, and N'_i is obtained from N'_{i-1} by contracting an rr-edge that does not map to an rtr-path in N_{i-1} (or by contracting a tt-edge that does not map to a trt-path in N_{i-1}, depending on the orchard class) for $i \in [k]$. The networks N_i are obtained by contracting the edges of the rr-path or the rtr-path (or the tt-path or the trt-path) as outlined in the previous paragraph. Note also that N'_i is a refinement of N'. Then we have that N'_k, which is a refinement of N', is a subnetwork of N. Therefore N' is contained in N. □

Finally, we present a lemma analogous to Lemma 31 for two orchard networks within the same reconstructible class.

Lemma 33. *Let C be a reconstructible orchard class. Let N and N' be orchard networks in C on taxa set X and $X' \subseteq X$ respectively, and suppose that a minimal CPS S for N also reduces N'. Then N' is contained in N.*

Proof. Let S' be the subsequence of S consisting of the elements that affect N', and let N_b and N'_b be the unique binary orchard networks obtained from S and S' using the (1a,2a) construction. Then, by Lemma 21, N_b is a binary refinement of N with minimal CPS S, and N'_b is a binary refinement of N' with minimal CPS S'. By Lemma 31, N'_b is a subnetwork of N_b. By Lemma 32, N' is contained in N. □

Note that Lemma 33 does not hold if we used subnetwork instead of containment (see Figure 4.12). The converse of Lemmas 31 and 33 do not hold for general orchard networks, and we show this in the following subsection.

4.4.2. CONTAINMENT DOES NOT ALWAYS IMPLY REDUCTION

Following subsection 4.4.1, we give an example of an orchard network N and a tree T on the same leaf-sets, for which T is a subnetwork of N, such that there exists no minimal CPS for N that reduces T to a single leaf. This example is shown in Figure 4.13.

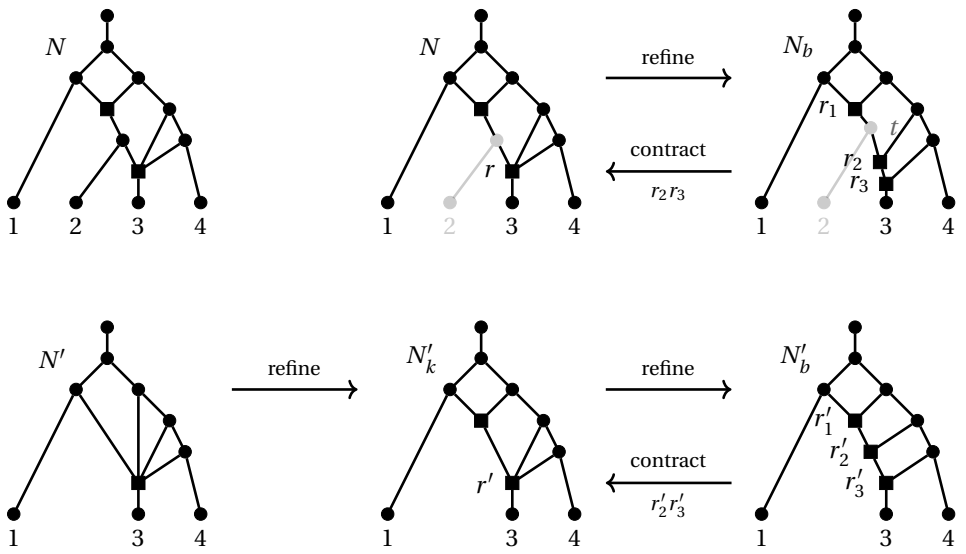


Figure 4.11: Visualization of proof of Lemma 32. N and N' are orchard networks of the (1a,2b) class on different sets of taxa where N' is contained in N. In proving the "only if" direction (supposing that N' is contained in N), we use the fact that N'_k, a refinement of N' is a subnetwork of N. The embedding of N'_k into N can be seen in the top middle figure, by looking only at vertices and edges in black. In this embedding, r' is mapped to r. We obtain a binary resolution N_b of N by refining r. We refine r' to obtain N'_b, a binary resolution of N, which is a subnetwork of N_b. For the other direction (supposing that a binary resolution N'_b of N' is a subnetwork of a binary resolution N_b of N), notice that an embedding of N'_b into N_b maps the edges $r'_1r'_2$ and $r'_2r'_3$ into r_1r_2 and r_2r_3, respectively. Observe that r_1r_2 is an rtr-path (with the tree vertex t) and r_2r_3 is an rr-path; because of this, we contract the edge $r'_2r'_3$ but not $r'_1r'_2$ to obtain the refinement N'_k. Upon contracting all rr-edges of N_b, which is only the edge r_2r_3 in this case, we obtain N, and we can see that N'_k is a subnetwork of N.

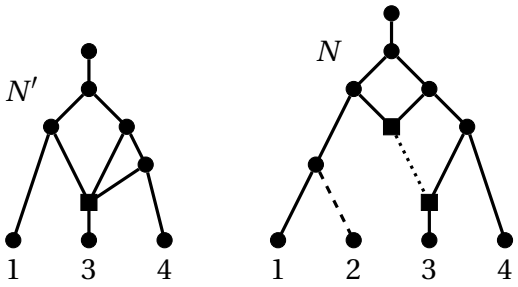


Figure 4.12: An illustration of Lemma 33. The tree-child network N' is contained in the tree-child network N, which can be seen by deleting leaf 2 (and as a result deleting the dashed edge and suppressing the degree-2 vertex) and contracting the dotted edge in N. The TCS $(2,1)(3,4)(3,1)(3,4)(1,4)$ for N reduces N' as well. However, N' is clearly not a subnetwork of N.

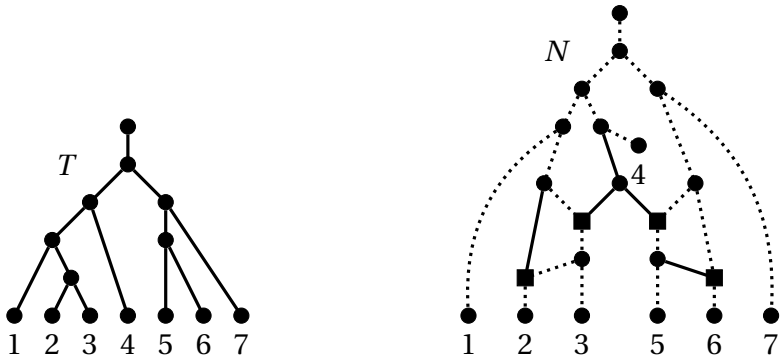


Figure 4.13: The tree T is a subnetwork of an orchard network N, and one embedding of T into N is shown (dotted edges). There exists no minimal CPS of N that reduces T to a single leaf.

Theorem 12. *There exists a binary orchard network N that contains a tree T, such that no minimal CPS for N reduces T.*

Proof. We refer by N and T to the network and the tree of Figure 4.13.

Note first that $\mathscr{C}(N) = \{(2,3),(6,5)\}$. So initially, we are required to pick one of the reticulated cherries $(2,3)$ or $(6,5)$.

Picking $(2,3)$ first reduces both T and N, and in the next step we have the option of picking one of the reticulated cherries $(3,2)$ or $(6,5)$. Picking $(3,2)$ does not affect $T(2,3)$, and reduces the reticulated cherry $(3,2)$ in $N(2,3)$. In $N(2,3)(3,2)$, 3 is no longer a child of a reticulation. Let u denote the LCA of 1 and 3. The path from u to 3 contains the parent of 4. This implies that one of 3 or 4 is picked by the time we can pick leaf 1, which ultimately means that T is not reduced to a leaf with any CPS starting with $(2,3)(3,2)$. So we pick $(6,5)$.

Picking $(6,5)$ first reduces both T and N, and in the next step we have the option of picking one of the reticulated cherries $(2,3)$ or $(5,6)$. Picking $(5,6)$ does not affect $T(6,5)$, and reduces the reticulated cherry $(5,6)$ in $N(6,5)$. Let u denote the LCA of 5 and 7 in $N(6,5)(5,6)$. Since u is the child of the root, the pair $(5,7)$ can only appear in the CPS as the final cherry. But $(5,7)$ is a cherry in the tree $T(6,5)(5,6)$, which still contains other leaves. This implies that a minimal CPS for N starting with $(6,5)(5,6)$ cannot reduce T to a single leaf.

Thus, the CPS must start with either $(2,3)(6,5)$ or $(6,5)(2,3)$. Note that $T(2,3)(6,5) = T(6,5)(2,3)$ and $N(2,3)(6,5) = N(6,5)(2,3)$ and so the order in which we pick the two reticulated cherries do not matter. In both cases, we have the choice of picking one of the reticulated cherries $(3,2)$ or $(5,6)$. However, doing so results in a CPS that does not reduce T to a single leaf, by the argument above. Since every CPS of N starts with these cherries, we have that T is not reduced to a single leaf for any CPS of N. □

Since this particular network is semi-binary stack-free, it serves as an example for when containment does not imply reduction for the $(1a,2a)$ and the $(1a,2b)$ classes. For the other two reconstructible classes, the claim may still hold true. We speculate that similar results follow, which we discuss in Section 5.6.

In what follows, we show that if we restrict our scope to the class of tree-child networks, then the converse does hold. That is, for those networks, containment implies reduction.

4.5. TREE-CHILD SEQUENCES

RECALL that a condition was imposed on a sequence of ordered pairs to define CPSs as those that can be used to construct networks. We show here that imposing an additional condition on the CPSs can ensure the constructed network to be tree-child. Recall that a network is tree-child if every non-leaf vertex has a child that is a tree vertex or a leaf.

In this section, we assume that we work in orchard classes that use the 2*b* or the 2*d* reticulated cherry construction, to ensure that the network constructed from the sequences is stack-free. Outside of stacks, the only forbidden structure of tree-child networks are tree vertices whose children are all reticulations. To ensure these structures do not appear in the constructed networks, we impose a condition on our sequences, that the first coordinate of each pair does not appear as a second coordinate of another pair in the remainder of the sequence. We will refer to this condition as the *tree-child condition.*

Definition 13. A CPS is a *tree-child sequence (TCS)* if every leaf appearing as the first coordinate does not appear as a second coordinate in the rest of the sequence.

Lemma 34. *Let S be a TCS. Then each of the networks obtained by choosing construction 2b or 2d for each reticulated cherry is tree-child.*

Proof. By construction, the networks obtained from S are orchard networks and they are stack-free. It remains to show that each tree vertex has at least one child that is a tree vertex or a leaf.

Let $S_i = (x, y)$ be a reticulated cherry, and consider the network N after having just added S_i. In N, the tree vertex parent p_y of y currently has at least one leaf child y. Suppose that all its other children were reticulations. For this vertex p_y to have all reticulation children in the fully constructed network, we require some reticulation vertex to be inserted between p_y and y, which can only happen if we add some ordered pair $S_j = (y, z)$ where $j < i$. However, this would mean that y appears as a first coordinate of some pair S_j and also as a second coordinate of some pair S_i later on in the sequence, which contradicts our tree-child condition.

Now, if $S_i = (x, y)$ was a cherry, then the parent p of x cannot be a parent of only reticulations after adding more pairs to the network. Indeed, this would imply that we have added some reducible pair (y, z) later on to the network—and hence it would appear earlier in the sequence—which again contradicts our tree-child condition.

Hence all tree vertices of a network obtained from a TCS have at least one child that is a tree vertex or a leaf. Therefore such a network is tree-child. □

Tree-child networks (TCNs) always contain a reducible pair, and after reducing one of these, we obtain a new tree-child network (Lemma 4.1 of [12]). Naturally, this implies

that TCNs are orchard networks. As we have seen in the previous section, given an orchard network that contains a tree on the same set of taxa, there may not exist a minimal CPS of the network that reduces the tree (Theorem 12). In this section, we make the switch from CPSs to TCSs, and show that this is no longer an issue for TCSs. In fact, we prove a stronger result: within a reconstructible orchard class, a TCN contains another TCN on the same taxa if and only if every minimal TCS of the first TCN reduces the second TCN. If, in addition, the first TCN is binary, then if *any* minimal TCS for the larger network reduces the smaller network, then the smaller network is a subnetwork of the larger network. We also show that—similar to Proposition 1 for orchard networks—we may pick reducible pairs in any order for TCNs. We start by showing that every TCN has a minimal TCS. This was shown implicitly for semi-binary TCNs in the proof of Lemma 3.4 of [10]; however we include it here for completeness and to generalize for non-binary TCNs.

Lemma 35. *Let N be a non-binary TCN. Then there is a TCS for N.*

Proof. Let N be a network on X. For a partial TCS S, denote by $F(S) \subseteq X$ the set of first elements of pairs of S. Because N is tree-child, for each node v, there is a path to a leaf $t(v)$ in which all internal nodes are tree nodes, and $t(v) = t(c)$ for some child c of v if v is not a leaf node.

Assume S is a partial TCS such that $t(v)$ is still below v via a tree-path for all nodes v of NS, and for each $l \in F(S)$, the nodes v for which $t(v) = l$, are l, and possibly the parent of l if it is a reticulation. Suppose N is not fully reduced by S, we show that we can find a longer sequence S' starting with S to which the conditions also apply.

Let p be a lowest tree node in NS, then each node below p is either a leaf, or a reticulation which is directly above a leaf. The leaf $t(p)$ is directly below p (as there is a tree-path from p to $t(p)$) and $t(p) \notin F(S)$ by the assumption on S. By reducing all cherries and reticulated cherries involving p using $t(p)$ as second element, we obtain a new partial tree-child sequence S'. In essence, this reduction deletes all outgoing edges of p except for $pt(p)$, suppresses all degree-2 vertices (which includes p), and deletes any isolated vertices. We claim that for all nodes v of NS', $t(v)$ is still below v by a tree-path. Indeed, deleting all but one outgoing edge of p only affects the tree-paths that were leading to p. So for all vertices in NS that were not ancestors of p, the claim holds. On the other hand, for all ancestors g of p in NS with $t(g) = t(p)$, there is still a tree-path from g to $t(g)$ in NS'. So our claim holds. Furthermore, each leaf $l \in F(S')$ is either directly below a reticulation, or directly below a tree node v with $t(v) \neq l$, as in NS, l was below a reticulation below p; or $l \in F(S)$ and l was already below v, so $t(v) \neq l$ in NS by assumption.

Starting with the empty partial TCS, we can repeat this process until NS has no more tree nodes. When this is the case, we have found a TCS S for N. □

4.5.1. SUBNETWORK / CONTAINMENT IMPLIES REDUCTION

As in the previous sections, we will try to keep the results as general as possible. We first show results on reduction and subnetworks.

Lemma 36. *Let N be a non-binary tree-child network, N' a tree-child subnetwork of N with the same leaf set, and S a TCS. Then $N'S$ is a subnetwork of NS.*

Proof. We prove this fact inductively on the length of S. If S is empty, then $N'S = N'$ and $NS = N$, so $N'S$ is a subnetwork of NS.

Now suppose that for any TCS S' of length at most j, $N'S'$ is a subnetwork of NS'. We prove that for any TCS S of length $j+1$, $N'S$ is a subnetwork of NS. Let us denote $S = S'(x,y)$. Note that S' is of length at most j. Hence, by the induction hypothesis, $N'S'$ is a subnetwork of NS'. We consider the following cases:

- **$N'S'$ has only one of x and y.** Because $S = S'(x,y)$ is a TCS, y is not the first coordinate in any element of S'. Hence, $N'S'$ must still contain y, and x must have been deleted from N' by applying S'. This means the edge of NS' deleted by applying (x,y) is not used by the embedding of $N'S'$ into NS', and $N'S = N'S'$ can still be embedded in NS.
- **$N'S'$ has both x and y.** There are a few cases we must consider, depending on whether there are reducible pairs (x,y) in $N'S'$ and NS'.
  - **NS' has a cherry (x,y).** As $N'S'$ also contains both leaves x and y, $N'S'$ also has the cherry (x,y), which is mapped to the corresponding cherry in NS' by the embedding. The reduction of (x,y) in both networks removes the pendant edge leading to x in both networks, not changing the embedding otherwise.
  - **NS' has a reticulated cherry (x,y).**
    - **The edge $p_y p_x$ is used by the embedding of $N'S'$ into NS'.** First note that $N'S'$ must have either a cherry or a reticulated cherry (x,y): if the edge $p_y p_x$ is used by the embedding, then the only way to reach x and y in NS', is by using the edges $p_x x$ and $p_y y$, making (x,y) either a cherry or a reticulated cherry in $N'S'$. Now applying (x,y) to $N'S'$ deletes the edge using $p_y p_x$ of NS' in the embedding. Hence, upon deleting both these edges by applying (x,y) to both networks, we may naturally extend the embeddings.
    - **Otherwise.** $N'S'$ can be embedded in NS' without the edge $p_y p_x$. Hence, $N'S'$ is a subnetwork of NS' after the removal of this edge $p_y p_x$ (i.e., the network NS). As $N'S$ is a subnetwork of $N'S'$ and $N'S'$ is a subnetwork of NS, $N'S$ is a subnetwork of NS.
  - **Otherwise.** The network NS' contains neither a cherry, nor a reticulated cherry on x and y. This means $NS = NS'$. As $N'S$ is a subnetwork of $N'S'$, and $N'S'$ is a subnetwork of NS', $N'S$ is a subnetwork of $NS (= NS')$.

□

The following corollary follows immediately, as subnetworks of single-leaf networks are single-leaf networks.

Corollary 5. *Let N and N' be non-binary tree-child networks on the same leaf set, with N' a subnetwork of N. If a TCS S reduces N, then S also reduces N'.*

Observe that in the setting of Corollary 5, the two networks are reduced to the same single-leaf network, with the same leaf label.

Theorem 13. *Let C be a reconstructible orchard class. Let N and N' be tree-child networks in C on the same leaf set. Then N' is contained in N if and only if* any *TCS of N reduces N'.*

Proof. Follows immediately from Corollary 5, Lemma 31, and Lemma 33. □

Note that in Theorem 13, it is necessary for the two networks to be contained in the same reconstructible orchard class. Consider the networks in the (1a, 2d) and the (1b, 2d) classes in Figure 4.4, which are both tree-child and can be reduced by the same tree-child sequence. The latter network is not contained in the former.

For semi-binary TCNs, we show that the notions of containment and subnetwork are equivalent.

Lemma 37. *Let N, N' be both semi-binary TCNs on the same leaf-set. Then N contains N' if and only if N' is a subnetwork of N.*

Proof. One direction is clear by definition of containment and subnetwork, so suppose that N' is contained in N. Then there exists some refinement N'_f of N' that is a subnetwork of N (i.e., there exists an embedding of N'_f into N). If $N'_f = N'$, then we are done, so we may assume that there exists an rr-edge e of N'_f that is mapped to an rtr-path P_e of N in the embedding. This implies that the tree-vertices on this path P_e are not used in the embedding; in particular, this means that any outgoing edge of a tree-vertex that is not in P_e is not used in the embedding. Let t denote the lowest tree-vertex on P_e. The child of t on P_e is a reticulation. By the tree-child property, t must have another child that is either a tree vertex or a leaf. Such a vertex is not on P_e, and in particular, this implies that there exists a tt-path from t to some leaf l, such that the path is not on P_e. So this tt-path is not used in the embedding. But every path from the root to the leaf l uses this tt-path. It follows that l does not appear as a leaf in the network N'_f and therefore in N'. This is a contradiction as N and N' have the same leaf-sets. Thus reticulations in N' need not be refined to obtain N'_f. However, since all tree vertices in N' are binary, all refinements of N' arise from refining its reticulation vertices. Then $N'_f = N'$, and therefore N' is a subnetwork of N. □

This lemma does not hold in general, for all other network classes for which the smaller network is not binary (see Figure 4.14). Furthermore the lemma does not hold for when the larger network N is not a TCN.

Theorem 14. *Let N and N' be semi-binary TCNs on the same leaf set. Then N' is a subnetwork of N if and only if any TCS of N reduces N'.*

Proof. Follows immediately from Theorem 13 and Lemma 37 □

Note that we could assume N' is non-binary, but to be a subnetwork of a semi-binary network, it has to be semi-binary as well, because N and N' are both TCNs on the same leaf set. Note that Theorem 14 is no longer true if we allow for the networks to have different leaf-sets. Let N' be the cherry on leaf-set $\{1,2\}$, and let N be the balanced tree on leaf-set $\{1,2,3,4\}$ with cherries $(1,3)$ and $(2,4)$. Then both networks are semi-binary tree-child, and N' is a subnetwork of N. However, the TCS $(1,3)(2,4)(3,4)$ of N does not reduce N'.

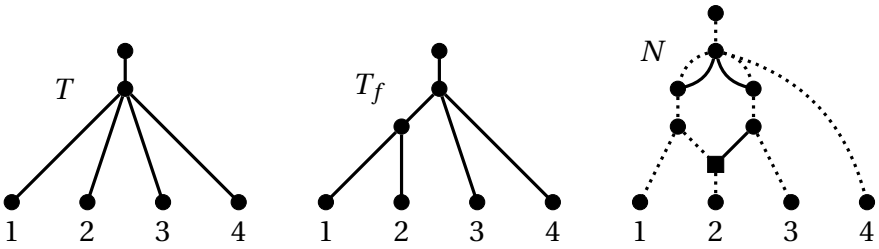


Figure 4.14: The tree T is contained in the tree-child network N using the refinement T_f, but it is not a subnetwork of N The embedding of T_f into N is shown by dotted edges. In particular, T and N belong to the (1*b*,2*c*) and the (1*b*,2*d*) classes.

4.5.2. ORDER DOES NOT MATTER IN TCSS

Theorem 9 states that we may pick cherries from an orchard network in any order and still obtain a minimal CPS. In this section, we show that this also holds for TCSs.

Lemma 38. *Let N be a non-binary TCN, S a partial TCS and c an ordered pair of leaves such that cS is also a partial TCS. Then NcS is a subnetwork of NS.*

Proof. We consider the networks $NcS_{[:j]}$ and $NS_{[:j]}$ for $j \geq 0$. We prove, using induction on j, that $NcS_{[:j]}$ is a subnetwork of $NS_{[:j]}$. Obviously, this is true for $j = 0$, as Nc is a subnetwork of N.

Write $S_{j+1} = (x, y)$ for the $j+1$-th element in the sequence. First note that if x is not a leaf of $NcS_{[:j]}$, then $NcS_{[:j]}$ is still a subnetwork of $NS_{[:j]}(x, y)$, as the embedding of $NcS_{[:j]}$ in $NS_{[:j]}$ does not use the removed edge that leads only to x. Therefore $NcS_{[:j+1]} = NcS_{[:j]}$ is a subnetwork of $NS_{[:j+1]} = NS_{[:j]}(x, y)$.

Now assume x is a leaf of $NcS_{[:j]}$. If reducing $NS_{[:j]}$ with $S_{j+1} = (x, y)$ does not remove an edge e used by the embedding of $NcS_{[:j]}$, then $NcS_{[:j+1]}$ is still a subnetwork of $NS_{[:j+1]}$.

So, we now assume reducing $S_{j+1} = (x, y)$ in $NS_{[:j]}$ removes an edge e used by the embedding of $NcS_{[:j]}$. Suppose for a contradiction that $NcS_{[:j+1]}$ is not a subnetwork of $NS_{[:j+1]}$. For this to be true, the edge of $NcS_{[:j]}$ mapped to the path containing e must not be removed by reducing S_{j+1}. This can only happen if $NcS_{[:j]}$ does not contain the reducible pair (x, y), whereas $NS_{[:j]}$ does have it. The reducible pair (x, y) in $NS_{[:j]}$ must be a reticulated cherry, as the only other option is that it is a cherry, but then, $NcS_{[:j]}$ also contains this cherry, as x and y are both part of $NcS_{[:j]}$ and $NcS_{[:j]}$ is displayed by $NS_{[:j]}$. Hence, $NcS_{[:j]}$ does not contain a cherry nor a reticulated cherry (x, y), and $NS_{[:j]}$ contains a reticulated cherry (x, y). As $NcS_{[:j]}$ is a subnetwork of $NS_{[:j]}$ whose embedding uses the reticulation edge of the reticulated cherry (x, y), the leaf y must have been deleted from $NcS_{[:j]}$ already. This means y must be the first element of a pair in the partial TCS $cS_{[:j]}$. However, cS is a partial TCS with y as the second coordinate of S_j, so we have a contradiction. We conclude that if the reduction removes an edge from $NS_{[:j]}$, it also removes the corresponding edge (i.e., the edge of $NcS_{[:j]}$ that was mapped to the path of $NS_{[:j]}$ containing the edge) from $NcS_{[:j]}$.

We conclude that $NcS_{[:j+1]}$ is a subnetwork of $NS_{[:j+1]}$. □

Lemma 39. *Let S and S' be TCSs such that S' is a subsequence of S. Then there exists a sequence of TCSs $S = \Sigma^0, \ldots, \Sigma^{|S|-|S'|} = S'$ such that $|\Sigma^i| = |\Sigma^{i-1}| - 1$ for $i \in [|S| - |S'|]$.*

Proof. Let S'' denote the sequence (not necessarily a CPS) obtained by taking the elements of S which do not occur in S' (in order). For $i \in [|S''|]$, let $f(i)$ denote the position where $\Sigma^{i-1}_{f(i)} = S''_i$, which is the element in the sequence that will be deleted to obtain Σ^{i+1}. We claim that if Σ^i is a TCS, then we can obtain a TCS Σ^{i+1} by removing the element $S''_i = \Sigma^{i-1}_{f(i)}$. Upon repeating this for all i, we obtain the sequence of TCSs that we are after.

To show this, suppose for a contradiction that Σ^i is a TCS, but removing S''_i from Σ^i results in a sequence Σ^{i+1} that is not a TCS. Note first that the 'tree-child property' of the sequence is retained when deleting elements from TCSs: indeed, every leaf appearing as a first coordinate still does not appear as a second coordinate in the rest of the sequence. So we must have that Σ^{i+1} is not a CPS, that is, there exists a leaf that appears as a second coordinate, but not as a first coordinate in the remaining sequence. Then the CPS property is violated for an element which occurs in $\Sigma^i_{[:f(i)-1]}$. Note that $\Sigma^i_{[:f(i)-1]}$ forms the first $f(i)-1$ elements of S', since we defined S'' as the elements of S which do not occur in S' in order. Furthermore, note that at each step we do not add elements to the sequence. Then $\Sigma^{|S|-|S'|} = S'$ cannot be a CPS, let alone a TCS, which contradicts our assumption that S' was a TCS. Therefore Σ^{i+1} must be a TCS and the lemma follows. □

Corollary 6. *Let N be a non-binary TCN with S and S' TCSs such that S' is a subsequence of S. Then NS is a subnetwork of NS'.*

Proof. By Lemma 39, we only have to prove this for S' of length one less than S, so suppose $S = S'_{[:i]} s S'_{[i+1:]}$, where the first or the last part may be empty.

We consider $NS = NS'_{[:i]} s S'_{[i+1:]}$, by applying the three parts of the sequence separately. First we note that by writing $N' := NS'_{[:i]}$, we have $NS = N' s S'_{[i+1:]}$ and $NS' = N' S'_{[i+1:]}$. Because S is a TCS, the sequence $sS'_{[i+1:]}$ is in particular also a TCS. Hence, by Lemma 38, NS is a subnetwork of NS'. □

The following proposition says that we can find a TCS for a TCN by picking an arbitrary reducible pair, reducing it, and repeating this process.

Proposition 2. *Let N be a semi-binary TCN and S a partial TCS with every element affecting N. Then there exists a minimal TCS for N starting with S.*

Proof. We prove this using induction on the length l of S. If l is 0, then, as each TCN has a minimal TCS, there is a TCS $S' = SS'$ for N, which starts with S. Now suppose for any partial TCS S of length $l < L$ affecting N in every step, there is a minimal TCS SS' of N. We prove that the same holds for any such sequence of length L.

Let $S(x, y)$ be such a sequence of length L (where (x, y) is the last element of this sequence). Because each element of $S(x, y)$ affects N, we know in particular that $(x, y) \in \mathscr{C}(NS)$. By the induction hypothesis, there is a TCS SS' for N starting with S. The part S' of this sequence must contain an element (x, y) or (y, x) as N is semi-binary (Lemma 25). Let S'_i be the first occurrence of such an element.

Each of the intermediate networks $NSS'_{[:j]}$ for $j < i$ has the reducible pair (x, y). This means the only pairs involving x that affect these networks have x as first coordinate, or are equal to (y, x). As S'_i is the first occurrence of (x, y) or (y, x) in S', all S'_j with $j < i$ cannot have x as second coordinate. This means that $S(x, y)S'$ is a TCS, and it reduces N by Corollary 6.

Note that each element of $S(x, y)$ affects N. Since we know SS' is a minimal TCS for N, and because $S(x, y)S'$ contains one extra element, there is an element of S' that reduces nothing in N in the sequence $S(x, y)S'$. Removing this element gives a minimal TCS for N starting with $S(x, y)$. □

Proposition 3. *Let N be a non-binary TCN and S a partial TCS that affects N. Then there exists a minimal TCS for N starting with S.*

Proof. Let N' denote the refinement of N obtained by applying the (1*a*, 2*b*) construction on NS for the sequence S. Observe that N' is a TCN since NS is tree-child. Because of this, every multifurcation of N' has a child that is a leaf or a tree vertex, which we call t. By refining each multifurcation as a path in which the lowest vertex is the parent of t, we obtain a semi-binary refinement N'_{sb} of N' that is tree-child. In particular, the tree-vertices of the reducible pairs in N'_{sb} that are reduced by S have outdegree-2, because of the construction we have used to obtain N'. The way in which we have obtained N'_{sb} ensures that S is a partial TCS that affects N'_{sb}. By Proposition 2, there exists a minimal TCS S' for N'_{sb} starting with S. Since N'_{sb} is a refinement of N, it follows then that S' must also reduce N. □

4.6. COMPUTATIONAL ASPECTS OF CONTAINMENT PROBLEMS

THE TREE CONTAINMENT problem is a well studied problem, where one asks whether a tree is contained in a given network (with a common set of taxa). In this section, we look at the more general problem NETWORK CONTAINMENT, where the aim is to determine whether a network is contained in another network. We restrict our attention to the problem where the input networks are both semi-binary, tree-child networks with the same leaf set. We will give an algorithm for this problem that runs in linear time. We also show that it is possible to check in linear time whether two orchard networks are isomorphic.

NETWORK CONTAINMENT
Instance: Two networks N and N' on the same leaf-set.
Question: Does N contain N'?

4.6.1. TREE-CHILD NETWORK CONTAINMENT

In this section, we give the linear time algorithm (Algorithm 7) for NETWORK CONTAINMENT. We use a few small subroutines (Algorithms 2, 3, 4, 5, 6). The idea of the algorithm follows from Theorem 13 and Proposition 2. For two TCNs N and N', find a minimal TCS of N by picking reducible pairs in any order. See if this TCS reduces N' to a network on a single leaf: if it does, then N contains N'; otherwise, N does not contain N'. Assume in this subsection that every network is semi-binary stack-free, unless stated otherwise.

Within semi-binary networks, tree vertices are of outdegree-2. This means that each leaf appears as a second coordinate in at most one reducible pair in a network. Algorithm 2 finds such a reducible pair, if it exists, for a given leaf, in constant time.

Algorithm 2: FINDRP2ND(N, x)

Data: A semi-binary network N and a leaf x
Result: The singleton set containing the reducible pair of N having x as a second coordinate if it exists, $\emptyset$ otherwise

```
Let p be the parent of x
if p is a tree node then
    let c(p) be the child of p that is not x
    if c(p) is a leaf then
        return {(c(p), x)}
    if c(p) is a reticulation and the child c(c(p)) of c(p) is a leaf then
        return {(c(c(p)), x)}
return ∅
```

Lemma 40. *Let N be semi-binary, and x a leaf of N. If a reducible pair with x as the second element of the pair exists, then Algorithm 2 finds this pair. Otherwise, it returns the empty set. The algorithm runs in constant time.*

Proof. Each leaf is a second coordinate of at most one reducible pair, and this pair is found by taking the unique path down from the parent of this leaf. This algorithm runs in constant time: using in- and out-adjacency lists, we can check in constant time whether a node is a tree node, a reticulation node, or a leaf, and the out-list of each node has size at most 2 (as tree nodes have outdegree 2, reticulations outdegree 1, and leaves outdegree 0). □

Algorithm 3 on the other hand finds all reticulated cherries that contain a given leaf as the first coordinate of the reducible pair. The running time for this algorithm depends on the indegree of the parent of the given leaf, as this gives the maximum possible number of such reticulated cherries.

Algorithm 3: FINDRC1ST(N, x)

Data: A semi-binary network N and a leaf x
Result: The set $\mathscr{C}_r = \{(l,k) \in \mathscr{C}(N) : l = x\}$ of reticulated cherries in N having x as first coordinate

```
Let p be the parent of x
Set C_r = ∅
if p is a reticulation then
    for every parent g of p do
        let c(g) be the other child of g
        if c(g) is a leaf then
            C_r = C_r ∪ {(x, c(g))}
return C_r
```

Lemma 41. *Let x be a leaf in a semi-binary network N and let p_x denote the parent of x. Let I denote the indegree of p_x. Algorithm 3 finds the set of all reticulated cherries with the reticulation on the leaf x in $O(I)$ time.*

Proof. We simply check that the parent of x is a reticulation, and that (x, y) is a reticulated cherry if a grandparent of x is the parent of y. The for loop iterates at most I times. The steps within the for loop runs in constant time, since we may use the in- and out-adjacency lists as stated in the proof of Lemma 40. Therefore, Algorithm 3 runs in $O(I)$ time. □

Algorithm 4: REDUCEPAIR$(N, (x, y))$

Data: A non-binary network N and a pair (x, y)
Result: The network $N(x, y)$

```
1 if (x, y) is a cherry in N then
2     Let p be the parent of x and y
3     Remove edge px from N
4     Suppress p (if necessary) and remove x in N
5 if (x, y) is a reticulated cherry in N then
6     Let p_x be the parent of x and p_y the parent of y
7     Remove edge p_y p_x from N
8     Suppress p_y and p_x (if necessary) in N
9 return N
```

Lemma 42. *Algorithm 4 reduces a given reducible pair in a non-binary network N. The algorithm runs in constant time.*

Proof. The algorithm takes exactly the steps which define a reduction of a pair in a network, so the algorithm is correct. Then for the running time: comparing the unique parents of the leaves, we can check whether a pair constitutes a cherry or a reticulated cherry in constant time. Indeed, if a pair (x, y) forms a cherry, then the parents of x and y are the same; and if (x, y) forms a reticulated cherry, then the unique other child of the parent of y is the parent of x. The removal of the edge with subsequent suppression takes at most constant time. Hence, the subroutine runs in constant time. □

Algorithm 5: FINDTCS(N)

Data: A semi-binary TCN N
Result: A minimal TCS S for N

1 Set $\mathscr{C} = \emptyset$
2 **for** $x \in L(N)$ **do**
3 $\quad \mathscr{C} \cup$FINDRP2ND(N, x)
4 Let S be an empty sequence
5 **while** $\mathscr{C} \neq \emptyset$ **do**
6 $\quad$ Choose $(x, y) \in \mathscr{C}$
7 $\quad$ Set $S = S(x, y)$
8 $\quad N' =$ REDUCEPAIR$(N, (x, y))$
9 $\quad$ **if** (x, y) *is a cherry in* N **then**
10 $\quad\quad \mathscr{C} = \mathscr{C} \setminus \{(x, y), (y, x)\} \cup$FINDRP2ND$(N', y) \cup$FINDRC1ST$(N', y)$
11 $\quad$ **if** (x, y) *is a reticulated cherry in* N **then**
12 $\quad\quad \mathscr{C} = \mathscr{C} \setminus \{(x, y)\} \cup$FINDRP2ND$(N', y) \cup$FINDRC1ST$(N', y)$
13 $\quad N = N'$
14 **return** S

Recall that TCSs have the additional rule, on top of that of the CPSs, that any leaf appearing as a first coordinate of a pair in the sequence must not appear as a second coordinate of a pair later on in the sequence. Therefore we may have a cherry (x, y) in the network that can only be picked as (x, y) and not (y, x). We define a notion of when a reducible pair is *forbidden* before proving the correctness of Algorithm 5.

Definition 14. Given a partial TCS S, a reducible pair (x, y) is *forbidden* after S if y has appeared as the first coordinate of a pair in S.

Lemma 43. *Let N be a semi-binary TCN on taxa set X and reticulation number r. Algorithm 5 finds a minimal TCS for N in $O(|X| + r)$ time.*[2]

Proof. The first for loop finds all reducible pairs of N, as they all have a unique second coordinate, and each leaf is a second coordinate in at most one pair in a semi-binary network. Now by Proposition 2, each choice of non-forbidden pair gives a partial TCS for N. Hence, the algorithm may choose any non-forbidden pair and reduce it, so suppose that we pick (x, y). Regardless of whether (x, y) is a cherry or a reticulated cherry, the reduction takes constant time as REDUCEPAIR$(N, (x, y))$ runs in constant time. If (x, y) is a cherry, then we remove the pairs $(x, y), (y, x)$ from our list of pairs $\mathscr{C}$. All other pairs of $\mathscr{C}$ are still reducible in $N(x, y)$ by Lemma 25, and they are not forbidden because x is not a leaf of $N(x, y)$, so clearly there is no reducible pair with x as second coordinate. Noting that x is no longer a leaf in $N(x, y)$, the only new cherries in $N(x, y)$ must involve y. Since y is not a forbidden leaf, any cherry pair involving y is not forbidden: therefore we update our list $\mathscr{C}$ by appending both FINDRP2ND(N, y) and FINDRC1ST(N, y).

[2] [13] provided an improvement to our old version of Algorithm 5 in lines 11 and 13, when the newly obtained network is scanned for new reducible pairs. The algorithm presented here is the improved version. Before, the running time of the algorithm depended on the maximum indegree of the vertices in the network; now, the running time only depends on the number of leaves and the number of reticulation vertices.

On the other hand if (x, y) is a reticulated cherry, then the new cherry pairs in $N(x, y)$ must involve x or y, so by removing the pair (x, y) from the current list $\mathscr{C}$ and then adding the new pairs involving x and y, we get the updated list of pairs $\mathscr{C}$ for the TCN $N(x, y)$; we ensure that this updated list contains only non-forbidden pairs.

Observe that all reducible pairs of $N(x, y)$ that involve x are already contained in the set $\mathscr{C}$. Indeed, if x is involved in a cherry or a reticulated cherry (x, z), then (x, z) must have formed a reticulated cherry in N, which would have been found in the previous iteration of the while loop or in the initial search for all reducible pairs with second coordinate z. Note that x cannot be involved in a reticulated cherry where its parent p_x is a tree node, as this would imply that p_x was the parent of two reticulations in N, contradicting the tree-child property of N. Furthermore, since x has just been picked as a first element of some reducible pair, all pairs involving x as a second coordinate are now forbidden. Therefore, all possible new cherry pairs not in $\mathscr{C}$ that appear as a result of reducing (x, y) from N must involve the leaf y. These pairs are made up of those pairs contained in FINDRP2ND(N, y) or in FINDRC1ST(N, y). The only potential problem that we may face is when $(y, x) \in$FINDRC1ST(N, y). However this is impossible; indeed, if $N(x, y)$ contained a reticulated cherry (y, x), then it is immediately clear that N was not a tree-child network. The grandparent of x that is not the parent of y is a tree node parent of two reticulations in N. Therefore appending the cherries from the two sets FINDRP2ND(N, y) and FINDRC1ST(N, y) to $\mathscr{C}$ ensures that there are no forbidden cherry pairs in the updated list $\mathscr{C}$. Furthermore, updating $\mathscr{C}$ in this way ensures that we only look for each reducible pair once.

In the above cases, we must also make sure that the list $\mathscr{C}$ is non-empty as long as the network has not been reduced to a single leaf. But this is immediate by Proposition 2, and since we add all non-forbidden newly possible reducible pairs to the list. Therefore, Algorithm 5 is correct; repeating the while loop will eventually completely reduce the network, and give a TCS. In particular, since all ordered pairs in the sequence reduce a cherry or a reticulated cherry, the output TCS is minimal.

We now compute the running time of the algorithm. Let $|X|$ and r denote the number of leaves and the reticulation number of the network, respectively. Each call of FINDRP2ND takes constant time, so the for loop takes $O(|X|)$ time. Within the while loop, the algorithms REDUCEPAIR, FINDRP2ND, and FINDRC1ST are called $|X| + r - 1$ times since the length of a minimal TCS is of length $|X| + r - 1$ by Lemma 23. Each call of REDUCEPAIR and FINDRP2ND runs in constant time. Since every reducible pair is found at most once, Lines 5-7 of FINDRC1ST are called at most r times in total. This means that although FINDRC1ST will be called $|X| + r - 1$ times, the part of the algorithm that is dependent on the indegree of the reticulation vertex will only run at most r times. It follows then that the while loop runs in time at most $O(|X| + r)$, and therefore that Algorithm 5 runs in $O(|X| + r)$ time. □

Algorithm 6: CPSREDUCESNETWORK(N, S)

Data: an orchard network N and a CPS S
Result: Yes if S reduces N, No otherwise

```
for i = 1,...,|S| do
    N = REDUCEPAIR(N, S_i)
if N is a network on a single leaf then
    return Yes
return No
```

Lemma 44. *Let N be a CPS on taxa set X and reticulation number r, and let S be a CPS of length $|S|$. Algorithm 6 correctly checks whether S reduces N and runs in $O(|S|)$ time if N is semi-binary.*

Proof. The correctness of the algorithm follows straight from the definition of reducing a pair from orchard networks. To compute the running time, note that we iterate over the for loop $|S|$ times. The step within the for loop, Algorithm 4, runs in constant time for semi-binary networks. Hence, it is clear that the algorithm runs in $O(|S|)$ time. □

Algorithm 7: TCNCONTAINS(N, N')

Data: Two semi-binary TCNs N and N' on the same set of taxa
Result: Yes if N contains N', No otherwise

```
Set S = FINDTCS(N)
return CPSREDUCESNETWORK(N', S)
```

Theorem 15. *Given two semi-binary TCNs N and N' on taxa set X where the reticulation number of N is r, it can be decided in time $O(|X| + r)$ whether N' is a subnetwork of N.*

Proof. We prove the correctness of Algorithm 7 and show that it runs in linear time. We use Algorithms 5 and 6. First we find a TCS for N in $O(|X| + r)$ time with Algorithm 5. Then, again in $O(|X| + r)$ time, we see if this TCS reduces N' using Algorithm 6. If this TCS does indeed reduce N', then N' is a subnetwork of N by Theorem 13. Hence, combining the two algorithms, the second algorithm outputs yes precisely if N' is a subnetwork of N. Furthermore, the algorithm runs in linear time as the combination of the two algorithms runs in $O(|X| + r)$ time. □

A python implementation of Algorithm 7 was presented and tested in [13]. The algorithm was tested again on different data sets in [1]. In both analyses, it was concluded that the algorithm runs in linear time in practice as well, thereby showing that NETWORK CONTAINMENT can be solved for semi-binary tree-child networks in linear time in practice. The results show that even networks with over a thousand leaves can be dealt with within a second.

Two networks are isomorphic if both networks are subnetworks of the other. Therefore, Theorem 15 has the following corollary regarding NETWORK ISOMORPHISM, which asks whether two given networks are isomorphic. The problem for tree-child networks was previously shown to be solvable in $O(|X|^2)$ time [14]. We present the first linear-time algorithm for checking whether two tree-child networks are isomorphic.

Corollary 7. *Given two semi-binary TCNs N and N' on taxa set X where the reticulation number of N is r, it can be decided in $O(|X|+r)$ time whether N is isomorphic to N'.*

4.6.2. ISOMORPHISM RESULTS FOR ORCHARD NETWORKS

We mentioned in Section 4.3.2 that we can distinguish orchard networks within reconstructible classes by comparing their smallest CPSs. Indeed, by altering Algorithm 5, we can find the smallest CPS for a given orchard network in polynomial-time.The change is simple: at the start of the while loop, simply choose the smallest cherry instead of choosing any arbitrary cherry. Furthermore, every time we update the cherry list $\mathscr{C}$, if we have just reduced a reticulated cherry (x,y) from N, then we add another set FINDRP2ND(N,x).

There is another thing to take into consideration. Tree-child networks are defined to be stack-free; binary orchard networks are not necessarily stack-free. Because of this, upon reducing a reticulated cherry (x,y) from a binary orchard network, it is possible that a new reticulated cherry with x as the first coordinate is in the reduced network. This means that when updating the list of reducible pairs ($\mathscr{C}$ in the algorithms), we must also append elements in which x occurs as the first element, by calling the algorithm FINDRC1ST(N,x). Observe that we do not need to do this if the orchard network is stack-free, due to the same argument as stated in the proof of Lemma 43.

We start by introducing an algorithm that finds a smallest CPS for a semi-binary stack-free orchard network, and then show that simply changing one of the lines in the algorithm allows for an input of binary orchard networks. A slightly more involved change allows for an input of non-binary reconstructible orchard networkS.

Algorithm 8: FINDCPS(N)

Data: A semi-binary stack-free orchard network N, an ordering on the leaf set of N
Result: A smallest CPS S for N

```
Set 𝒞 = ∅
for x ∈ L(N) do
    𝒞 ∪ FINDRP2ND(N, x)
Let S be an empty sequence
while 𝒞 ≠ ∅ do
    Choose (x, y) ∈ 𝒞 such that (x, y) is smallest
    Set S = S(x, y)
    N' = REDUCEPAIR(N, (x, y))
    if (x, y) is a cherry in N then
        𝒞 = 𝒞 \ {(x, y), (y, x)} ∪ FINDRP2ND(N', y) ∪ FINDRC1ST(N', y)
    if (x, y) is a reticulated cherry in N then
        𝒞 = 𝒞 \ {(x, y)} ∪ FINDRP2ND(N', x) ∪ FINDRP2ND(N', y) ∪ FINDRC1ST(N', y)
return S
```

Lemma 45. *Let N be a semi-binary stack-free orchard network on taxa set X with reticulation number r. Algorithm 8 finds a smallest CPS for N in $O((|X|+r)|X|\log(|X|))$ time, if an ordering is given on X.*

Proof. Initially, we find the full set of reducible pairs by noting that a leaf appears as a second coordinate in a reducible pair at most once (the first for loop). Within the while loop, we always pick a smallest reducible pair. After picking a reducible pair (x, y), we update the list of all possible reducible pairs by adhering to Lemma 24: all new reducible pairs must contain one of the leaves x or y. As was the case for Algorithm 5, there is no need to include the set of reducible pairs $\textsc{FindRC1st}(N, x)$ as these pairs would have been found in one of the previous iterations. This ensures that all possible reducible pairs are put into the set $\mathscr{C}$. Therefore, the algorithm iteratively picks the smallest reducible pair from a list of all possible reducible pairs—hence, it returns a smallest CPS of the input orchard network (minimal in particular, since all the elements of the sequence reduce a cherry or a reticulated cherry in the orchard network).

To compute the running time, observe first that the for loop takes $O(|X|)$ time. Within the while loop, we first choose a reducible pair that is smallest. This takes $O(|\mathscr{C}|\log(|\mathscr{C}|))$ time by running some sorting algorithm, and since any leaf can appear as a second coordinate of at most one reducible pair, we have that $|\mathscr{C}| = O(|X|)$. Therefore, choosing a smallest reducible pair takes at most $O(|X|\log(|X|))$ time. By the same argument used in the proof of Lemma 43, we have that all other steps within the while loop take $O(|X| + r)$ time. Therefore the algorithm runs in $O((|X| + r)|X|\log(|X|))$ time. □

For the algorithm to work on binary orchard networks, simply exchange Line 12 of Algorithm 8 by the following:

$$\mathscr{C} = \mathscr{C} \setminus \{(x, y)\} \cup \textsc{FindRP2nd}(N', x) \cup \textsc{FindRC1st}(N', x) \\ \cup \textsc{FindRP2nd}(N', y) \cup \textsc{FindRC1st}(N', y);$$

Lemma 46. *Let N be a binary orchard network on taxa set X with reticulation number r. Algorithm 8 with the modification to Line 12 finds a smallest CPS for N in $O((|X| + r)|X|\log(|X|))$ time, if an ordering is given on X.*

Proof. The correctness of the algorithm follows from the proof of Lemma 45, with the addition that upon reducing the network by a pair (x, y), the newly introduced reducible pairs may have x as a first coordinate (which appear due to stacks). We resolve this by appending $\textsc{FindRC1st}(N', x)$ to the list of possible reducible pairs (Line 12).

For the running time analysis, observe first that the for loop runs in $O(|X|)$ time. The while loop is iterated $O(|X| + r)$ times, as a minimal CPS for a network on $|X|$ leaves and reticulation number r has length $|X| + r - 1$ by Lemma 23. As shown in the proof of Lemma 45, the running time of finding the smallest reducible pair from $\mathscr{C}$ takes $O(|X|\log(|X|))$ time. All other steps within the while loop take constant time: in particular, $\textsc{FindRC1st}$ takes constant time as the indegree of the reticulation vertices is bounded in a binary network. Hence, the while loop runs in $O((|X| + r)|X|\log(|X|))$ time, so the algorithm runs in $O((|X| + r)|X|\log(|X|))$ time. □

In case of non-binary reconstructible orchard networks, the operations require a slight care. This precaution stems from two reasons. Firstly, tree vertices are now allowed to have outdegree greater than 2. This means that every leaf no longer appears at most once as a second coordinate of a reducible pair: they can be a second coordinate of a pair multiple times. Therefore Algorithm $\textsc{FindRP2nd}$ may now return a set

containing more than one element. This poses a potential problem in the original Algorithm 8, since upon reducing the network, we update the list of reducible pairs by adding either FINDRP2ND(N', x) or FINDRP2ND(N', y) (or both). Indeed, this could mean that we could obtain the same reducible pairs multiple times if there exist cherries or reticulated cherries with a multifurcation. To avoid this, we only invoke these updates whenever the outdegree of the parent of x and y (in either the original or the reduced network) is 2. This helps avoid double or triple counting of the same reducible pairs. Secondly, as seen in Lemma 25 reducible pairs of the original network may not be reducible pairs in a reduced network, although the pair itself was not reduced. For example if a network contains cherries (x, y) and (x, z), then upon reducing by the pair (x, y), the new network no longer contains the cherry (x, z), since x is no longer a leaf in the network. Due to this, we must replace the lines 9 – 12 of Algorithm 8 by the following.


```
10 if (x, y) is a cherry in N then
11     for l ∈ L(N) do
12         if (x, l) is a cherry in N then
13             C = C \ {(x, l), (l, x)}
14     C = C ∪ FINDRC1ST(N', y)
15     Let p_y denote the parent of y in N
16     if outdegree(p_y) = 2 then
17         C = C FINDRP2ND(N', y)
18 if (x, y) is a reticulated cherry in N then
19     for l ∈ L(N) do
20         if (y, l) is a cherry in N then
21             C = C \ {(x, l)}
22     C = C \ {(x, y)} ∪ FINDRC1ST(N', y)
23     Let p_x denote the parent of x in N
24     Let p_y denote the parent of y in N
25     if indegree(p_x) = 2 then
26         C = C ∪ FINDRC1ST(N', x) ∪ FINDRP2ND(N', x)
27     if outdegree(p_y) = 2 then
28         C = C ∪ FINDRP2ND(N', y)
```

Lemma 47. *Let N be a non-binary reconstructible orchard network on taxa set X with reticulation number r. Algorithm 8 with the modification to Lines 9 – 12 finds a smallest CPS for N in $O((|X| + r)|X|^2 \log(|X|^2))$ time, if an ordering is given on X.*

Proof. The correctness of the algorithm follows from the proof of Lemma 45, with the changes outlined as above. As before, the first for loop finds all reducible pairs within the network. Upon reducing the smallest reducible pair, we update the list of reducible pairs by using the above replacement of the algorithm. Suppose that we have just reduced by a reticulated cherry (x, y). Let N denote the original network and $N' = N(x, y)$, as in the algorithm. As mentioned before, there may be reducible pairs in N that are no longer reducible pairs in N'. These are all reticulated cherries of the form (x, z), for which the reticulation edge was between the parent of y and the parent of x. Then, y and z must

have formed a cherry in N—therefore we check for all such cherries (line 20), and remove the corresponding reducible pairs from the list. When updating the list, all new reducible pairs involve either the leaf x or y. To avoid seeking for pairs that are already contained in the list, we only invoke the FINDRP2ND algorithm whenever new pairs arise in N', that is, whenever the reticulation parent of x is of indegree-2 or when the tree vertex parent of y is of outdegree-2 in N. This holds similarly for updating the list upon reducing cherries. Therefore, we find each reducible pair exactly once in the algorithm.

For the running time, observe that the for loop takes at most $O(|X|^2)$ time, as there can be at most $\binom{|X|}{2}$ possible reducible pairs in a network. Therefore $|\mathscr{C}| = O(|X|^2)$. The while loop is iterated $O(|X|+r)$ times, as a minimal CPS for a network on $|X|$ leaves and reticulation number r has length $|X|+r-1$ by Lemma 23. Each call of REDUCEPAIR runs in constant time. The running time for finding the smallest reducible pair from $|\mathscr{C}|$ takes $O(|X|^2\log(|X|^2))$ time. Upon reducing (x,y) from a network N, we enter a for loop that checks whether there are reducible pairs in the list that are no longer reducible pairs in the network. This step takes $O(|X|)$ time. Since each reducible pair is found at most once, the part of the algorithms FINDRP2ND and FINDRC1ST that has a dependency on the outdegree and the indegree, respectively will only run at most $O(\binom{|X|}{2}) = O(|X|^2)$ times. Therefore the steps within the while loop runs in time at most $O(|X|^2\log(|X|^2)$ time, and therefore Algorithm 8 with the proposed changes runs in $O((|X|+r)|X|^2\log(|X|^2))$ time. □

Theorem 16. *Suppose we are given an ordering on the taxa set X. Let N and N' be two networks with r reticulations within the same reconstructible class. Then it can be decided in $O((|X|+r)|X|^2\log(|X|^2)$ time whether they are isomorphic. In particular if the classes are (1a,2a) (binary) or 1a,2b) (semi-binary stack-free), then this can be decided in $O((|X|+r)|X|\log(|X|)$ time.*

Proof. Use Algorithm 8 twice to find the smallest CPSs for the two orchard networks. This can be done in $O((|X|+r)|X|^2\log(|X|^2))$ time (in $O((|X|+r)|X|^2\log(|X|^2)$ for the classes (1a,2a) (binary) or 1a,2b) (semi-binary stack-free)). By Corollary 4, these CPSs are the same if and only if the orchard networks are isomorphic. □

4.7. IMPLEMENTATION

ALGORITHM 7, which checks whether a given tree-child network is a subnetwork of another, was implemented in Python by Robbert Huijsman to test the theoretical linear bound in practice [13]. In this section, we present running time results of the implementation on a large randomly generated data set. We show that the theoretically proven linear running time is indeed achievable in practice. The tests were run on a Linux system with a quad-core Intel Xeon W3570 running at 1.7GHz and 24GB of DDR3 RAM clocked at 1333MHz. The operating system was Debian GNU/Linux 9 with a 4.19.46-64 Linux kernel. The software was written in Python version 3.7.3.

4.7.1. GENERATING THE DATASETS

For the test data, we generated 131200 instances of the tree-child network containment problem: two yes-instances and two no-instances for all $n,r,r' \in \{25,50,\ldots,975,1000\}$

with $r' \leq r$, where n is the number of leaves of both networks, r is the reticulation number in the first network, and r' the reticulation number in the second network. Each instance consists of two semi-binary tree-child networks on the same leaf-set, for which we asked whether the first network contained the second network.

For each instance, we generated the first network with n leaves and reticulation number r using Algorithm 10. The second network was generated depending on whether it was a yes- or a no-instance. If it was a yes-instance, a subnetwork with reticulation number r' was obtained using Algorithm 11; for a no-instance, a network on the same number of leaves and reticulation number r' was randomly generated with the same process as the first network (using Algorithm 10).

This way, each generated yes-instance is always a yes-instance for the NETWORK CONTAINMENT problem. For the no-instances, however, the random generation of the second network could also give a subnetwork of the first network, but the probability of that event is very small, as the number of tree-child networks grows very quickly with the number of leaves and reticulations [15].

The dataset used for the experiment along with the code for generating random datasets, and the actual implementation of Algorithm 7 can be found on https://github.com/RemieJanssen/Cherry-picking_TC_Network_Containment.

GENERATING RANDOM NETWORKS

The tree-child networks were randomly generated as TCSs using Algorithm 10. This algorithm takes two positive integers n and r, and outputs a tree-child network with n leaves and reticulation number r. It starts with the cherry $(1,2)$, and successively adds leaves as cherries, and reticulated cherries between two leaves that already exist in the network (respecting the tree-child condition).

In the algorithm, this is achieved by building a tree-child sequence backwards. It chooses to add a reducible pair corresponding to a cherry or reticulated cherry uniformly at random until we have added the required number of leaves and reticulation number. To make sure the sequence is a tree-child sequence, we keep a list NF of taxa that are 'non-forbidden', which, in this case, means that the taxon is not currently the child of a tree node that has a reticulation as the other child (i.e., the leaf has not appeared as a second coordinate element of a reducible pair). If a taxon is in NF, it is possible to take this taxon as the first element of a pair appended at the start of the sequence. As a tree-child network always has a cherry or a reticulated cherry, NF is never empty. This implies that the algorithm should never output False, but lines 15 and 16 are kept so that the algorithm can easily be adapted to return only binary tree-child networks. To achieve this, one only has to add the line "$NF = NF \setminus \{\text{first\_element}\}$" between lines 21 and 22 in the algorithm.

Finally, the algorithm outputs a TCS, from which we can uniquely construct a TCN.

Algorithm: RANDOMTCS(X, r)

Data: A set of taxa $X = \{1, \ldots, n\}$, and a reticulation number r
Result: A TCS S on X of length $n + r - 1$

Initialize $Y = \{1, 2\}$ the current set of taxa
Initialize $S = (2, 1)$ the current sequence
Initialize $L = n - 2$
Initialize $R = r$
Initialize $NF = \{2\}$
while $L > 0$ *or* $R > 0$ **do**
    type_added = None
    if $|NF| > 0$ *and* $L > 0$ *and* $R > 0$ **then**
        With probability $\frac{L}{L+R}$, type_added = L
        Otherwise, type_added = R
    else if $|NF| > 0$ *and* $R > 0$ **then**
        type_added = R
    else if $L > 0$ **then**
        type_added = L
    else
        return False
    first_element = None
    second_element = None
    if *type_added* $= R$ **then**
        Set first_element to an element of NF chosen uniformly at random
        Set $R = R - 1$
    else
        Set first_element to the first element of $X \setminus Y$
        Set $L = L - 1$
        Set $Y = Y \cup \{\text{first\_element}\}$
        Set $NF = NF \cup \{\text{first\_element}\}$
    Set second_element to an element of $Y \setminus \{\text{first\_element}\}$ chosen uniformly at random
    $NF = NF \setminus \{\text{second\_element}\}$
    $S = (\text{first\_element}, \text{second\_element})S$
return S

Note that each tree-child network has positive probability of appearing for this process. In fact, each tree-child sequence ending with $(1, 2)$ has positive probability.

Let us now turn to the procedure to generate a tree-child subnetwork (i.e., generating the second network in a yes-instance). For this purpose, we again work with the representation of the networks as tree-child sequences.

We first select ordered pairs from the sequence of the first network, such that the resulting subsequence corresponds to a tree. This is simply done by selecting a pair with first element x for all $x \in X$ uniformly at random. Because the sequence we started with

is a tree-child sequence, the subsequence consisting of the chosen pairs is a tree-child sequence as well: suppose (x, y) and (y, z) are selected. Then (y, z) must appear after (x, y), because otherwise y appears as a first element after it has appeared as a second element in the original sequence.

After selecting the pairs that form a *base tree* of the network (the subdivision of which forms a spanning tree of the network with leaf-set X), we select r' additional pairs that will form the r' reticulations of the subnetwork. By a similar argument as for the base tree, this subsequence is a tree-child sequence. And as it is reduced by the subsequence, it is also reduced by the sequence of the original network. Hence, the network corresponding to the chosen pairs is a tree-child subnetwork of the original network.

Algorithm: RANDOMSUBTCS(S, r')

Data: A TCS S on X of length $|X| + r - 1$, and a number $r' \leq r$
Result: A sub-TCS S' of S on X of length $|X| + r' - 1$

```
Let S be indexed by {1, ..., |S|}
Set I_{S'} = ∅
for x ∈ X do
    Let I_x be the set of indices of pairs of S with x as first element
    Pick i_x uniformly at random from I_x
    Set I_{S'} = I_{S'} ∪ {i_x}
Randomly add r' elements from {1, ..., |S|} \ I_{S'} to I_{S'}
Let S' be the subsequence of S consisting of the elements indexed by I_{S'}
return S'
```

4.7.2. RESULTS

For all yes-instance tests in which the second network was a subnetwork of the first (i.e., the ones generated by Algorithm 11), the algorithm correctly returned a true value. Similarly, for all no-instance tests in which the second network was generated randomly and independently from the first network, the algorithm correctly found that the second network was not a subnetwork of the first. This means that, even though there was a non-zero probability that the second network was a subnetwork, this did not happen in any of the instances. We expected this, as the probability of this happening is extremely small.

Note that the largest test instances (1000 leaves, 1000 reticulations) had a running time of approximately 0.1s. This is expected to scale well for even larger instances, as the linear fit of the data is very good. The R^2 values for the fits and the linear dependence of the running time on the number of leaves and reticulations can be found in Table 4.1. For this fit, we performed a standard linear regression with an intercept of 0 (i.e., forced through the origin), which makes sense because the running time should be zero for an empty instance.

Note that the fits become much better when we split the data in instances where the second network is or is not a subnetwork of the first (i.e., between the yes- and the no-instances), even though the dependence of the running time on the parameters does not change much after this split. The most striking difference we can see in this analysis, is the dependence on the reticulation number of the second network.

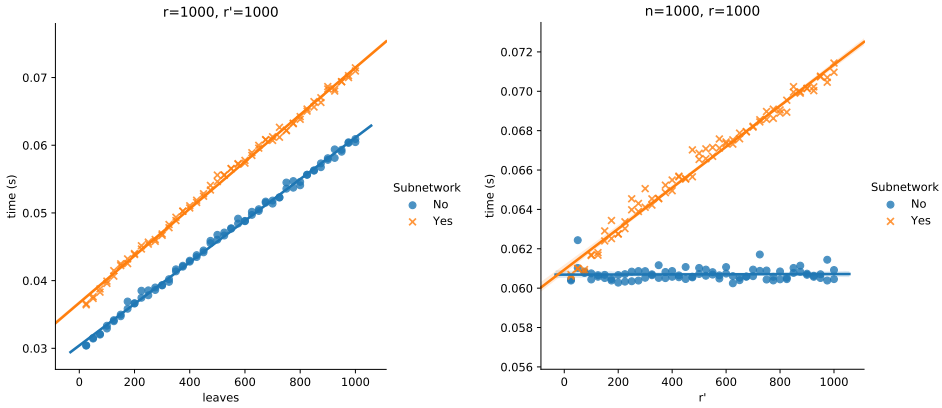


Figure 4.15: The dependence of the running time on the number of leaves n (left) and the number of reticulations in the second network r' (right). This was visualized by fixing the other parameters to a set value of 1000 in both plots.

Table 4.1: Linear regression analysis for tree-child network containment on 131200 instances, for which half were yes-instances and the other half no-instances. The high R^2 value indicates that the fit of the curve is essentially linear (where an R^2 value of 1 indicates a perfect linear fit) and the slopes indicate the change in running time for every increase in the number of leaves, reticulation number r, and reticulation number r'.

	R^2	slope		
		leaves (s/leaf)	r (s/reticulation)	r' (s/reticulation)
all data	0.9725659	$3.03328079 \cdot 10^{-5}$	$2.99713310 \cdot 10^{-5}$	$4.75681146 \cdot 10^{-6}$
subnetwork: YES	0.9966596	$3.14405850 \cdot 10^{-5}$	$2.89850496 \cdot 10^{-5}$	$9.54505907 \cdot 10^{-6}$
subnetwork: NO	0.9976078	$2.92250310 \cdot 10^{-5}$	$3.09576119 \cdot 10^{-5}$	$-3.14361016 \cdot 10^{-8}$

As shown in Figure 4.15, the no-instances were consistently, and marginally, faster than the yes-instances. For varying leaf numbers, the instances where the second network was not a subnetwork (no-instances), were consistently, but marginally, faster than when the second network was a subnetwork (yes-instances) (Figure 4.15, Left). This was similarly true for when we varied the reticulation number r' of the second network. The effect of varying r' on instances for when the second network was not a subnetwork (no-instances) was negligible. This can be seen in the right figure of Figure 4.15, but also in Table 4.1, where the order of the slope of r' for the no-instances is far smaller than all other slopes in all the instances. For the yes-instances, the running time of the algorithm showed a linear dependence on r', which was in the same order as the other parameters. This can be explained as follows. When the second network is not a subnetwork, the second part of the algorithm (Algorithm 6) will seldom need to reduce a pair: it will check whether the pair in the sequence is reducible in the second network. As the second network is randomly generated independently of the first network, it will not have many pairs in common with the first network, which means it will not have to reduce pairs often.

4.8. DISCUSSION

IN this chapter, we have looked at the correspondence between cherry-picking sequences and containment and subnetwork relations. For this purpose, we have extended the definition of cherry reductions to networks. This lead to the introduction of the class of orchard networks (consisting of all networks that can be reduced by a CPS), and of constructions that produce an orchard network from a CPS. All networks obtainable from a given construction were then said to be in the same reconstructible class.

We have shown that, within reconstructible classes, a network is encoded by the smallest sequence that reduces it (given an ordering on the leaves). This correspondence can be used to check whether two orchard networks in the same reconstructible class are isomorphic in polynomial-time. Moreover, we showed that if a sequence for a network reduces another network, then the second network is contained in the first network. For tree-child networks, we further showed that the converse was also true, i.e., that a tree-child network contained in another tree-child network will be reduced by any tree-child sequence of the second network, given that the networks are in the same reconstructible class. A summary of all results that relate reduction of a network and subnetwork / containment can be found in Table 4.2.

Table 4.2: Main results of the chapter relating network containment and CPSs. The two networks N and N' are assumed to have the same leaf-sets. The N and N' columns show the assumptions on both networks, where TC stands for tree-child. The direction column shows whether containment of N' in N implies ($\Longrightarrow$), is implied by ($\Longleftarrow$), or is equivalent to ($\Longleftrightarrow$) reduction of N' by any CPS for N. The subnetwork column contains a ✓when N' can be assumed to be a subnetwork instead of a contained network of N. For results in which only one direction is presented, we have counter-examples showing that the converse does not hold.

Result	N	N'	**Direction**	**Subnetwork**
Lemma 31	Binary		$\Longleftarrow$	✓
Theorem 11	Binary	Non-binary	$\Longleftarrow$	
Lemma 33	Same reconstructible class		$\Longleftarrow$	
Corollary 5	Non-binary TC		$\Longrightarrow$	✓
Theorem 13	Same reconstructible TC class		$\Longleftrightarrow$	
Theorem 14	Semi-binary TC		$\Longleftrightarrow$	✓

An apparent limitation of the CPS approach is that they cannot be used to characterize every phylogenetic network, but only orchard networks. The reason for this stems from the inability to pick a *double-reticulated cherry*. Two leaves x and y form a double-reticulated cherry if both parents p_x, p_y of x, y respectively are reticulations, and p_x and p_y share a common parent. As seen in other literature, the double-reticulated cherry poses a challenge when trying to correctly add back a once-removed reticulation edge [12, 16]. A CPS may contain a double-reticulated cherry, as long as there is an 'escape' at one of the sides of the double-reticulated cherry: a reticulated cherry which can be reduced, and whose reduction turns the double-reticulated cherry into a regular reticu-

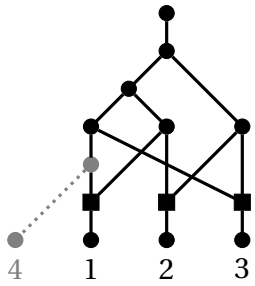


Figure 4.16: A tree-based network on leaves {1,2,3} that is not an orchard network, since it contains a crown (the subgraph induced by the parents and the grandparents of the leaves 1,2,3) without the gray vertex). The network on {1,2,3,4} is an orchard network, since it provides a reticulated cherry (1,4), an 'escape' for the crown.

lated cherry. With no escapes however, there is currently no way of dealing with double-reticulated cherries.

One way of resolving this issue would be to consider leaf attachments as done by [10]. Given any phylogenetic network, add a minimal number of leaves to make it an orchard network. Recall that every reconstructible orchard network has a unique smallest CPS (Theorem 10). Is it possible to extend this result to general networks by attaching leaves? If so, will the placement of the leaves affect the uniqueness of its smallest CPS? Furthermore, what is the minimal number of additional leaves to turn a general network into an orchard network? To answer these questions, one could study the forbidden structures of an orchard network; currently, very little is known about these.

Obviously, crowns cannot be contained in orchard networks. Aside from this, we are not aware of other forbidden structures of orchard networks. In any case, we may gauge where the class of orchard networks resides with respect to other prominent network classes. The class of orchard networks does not contain the class of tree-based networks [17], since tree-based networks may contain crowns (see Figure 4.16). Conversely, each binary orchard network is tree-based [18], and the next chapter (of which is a part of [19]) contains a proof that this holds in the non-binary case as well. Furthermore, as there exists a CPS (in fact, a TCS) for every TCN, we conclude that class of orchard networks strictly contains the class of TCNs. Therefore, in the binary case, but possibly also in the non-binary case, the class of orchard networks is intermediate between tree-child networks and tree-based networks.

We briefly comment on the correspondence between containment and reduction, and pose conjectures for remaining open questions. Let N and N' be orchard networks in the same reconstructible class on the same leaf-sets. If a CPS S of N reduces N', then N' is contained in N (Lemma 33). We have shown that the converse of this does not hold for the reconstructible classes (1a,2a) and (1a,2b) (Theorem 12). For the reconstructible classes (1b,2c) and (1b,2d), the problem remains open. Therefore we conjecture the following.

Conjecture 2. *There exist orchard networks N and N' on the same leaf-set in the* (1b,2c) *class where N' is contained in N, but no cherry-picking sequence of N reduces N'.*

Conjecture 3. *Let N and N' be orchard networks on the same leaf-set in the (1b,2d) class, where N' is contained in N. Then there exists a cherry-picking sequence of N that reduces N'.*

Upon restricting N and N' to be tree-child, we have shown that the converse holds for all reconstructible classes: if N' and N are both in the same reconstructible class and N' is contained in N, then there exists a TCS of N that reduces N'. We believe that this still holds when N and N' are general non-binary tree-child networks, by requiring a stronger condition. Note that it is easy to find an example of two non-binary tree-child networks that are not contained in one another, such that a TCS for one network reduces the other (see Figure 4.5 d).

Conjecture 4. *Let N and N' be non-binary tree-child networks on the same leaf set. Then N' is contained in N if and only if* all *TCSs of N reduce N'.*

In Section 4.1 we mentioned how cherry-picking sequences originated as a tool to tackle the problem of finding a 'simple' network that contains a given set of trees. This problem, called HYBRIDIZATION, and our results on cherry-picking sequences—in particular Lemma 31—provide insight into how we may solve generalizations of this problem, where we have inputs consisting of networks [20]. Indeed, if we find a CPS that reduces the input set of networks, then the network corresponding to this CPS contains the input set. However, it is not quite clear when this gives an optimal orchard network with minimal reticulation number. For example, take the network N and the tree T in Figure 4.13. If a CPS reduces the tree and the network, the corresponding orchard network must have at least one more reticulation than the optimal network, which is clearly the input network N. This means that given an input of networks, it may not be possible to find a CPS that corresponds to the orchard network with minimal reticulation number that contains all inputs. It would be interesting to see if this problem also occurs when the input consist solely of trees. That is, given an input of trees, can we always find the CPS which corresponds to an orchard network with minimal reticulation number containing all these trees? This problem aside, the problem of finding a minimal CPS gives an upper bound for HYBRIDIZATION, and a lower bound for TC CHERRY-PICKING, where TC CHERRY-PICKING asks to find a minimal TCS which reduces all trees in the input set. Hence, it might also be of merit to see what exactly happens when only trees are considered as the input: does the minimal CPS give useful information about any of these problems?

REFERENCES

[1] R. Janssen and Y. Murakami, *Linear time algorithm for tree-child network containment,* in *International Conference on Algorithms for Computational Biology* (Springer, 2020) pp. 93–107.

[2] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[3] S. Kelk, *Validating methods for constructing evolutionary phylogenetic networks,* (April 28, 2012), http://phylonetworks.blogspot.com/2012/04/validing-methods-for-constructing.html.

[4] I. A. Kanj, L. Nakhleh, C. Than, and G. Xia, *Seeing the trees and their branches in the network is hard,* Theoretical Computer Science **401**, 153 (2008).

[5] L. van Iersel, C. Semple, and M. Steel, *Locating a tree in a phylogenetic network,* Information Processing Letters **110**, 1037 (2010).

[6] P. Gambette, A. D. Gunawan, A. Labarre, S. Vialette, and L. Zhang, *Solving the tree containment problem for genetically stable networks in quadratic time,* in *International Workshop on Combinatorial Algorithms* (Springer, 2015) pp. 197–208.

[7] P. Gambette, A. D. Gunawan, A. Labarre, S. Vialette, and L. Zhang, *Solving the tree containment problem in linear time for nearly stable phylogenetic networks,* Discrete Applied Mathematics **246**, 62 (2018).

[8] A. D. M. Gunawan, *Solving the tree containment problem for reticulation-visible networks in linear time,* in *Algorithms for Computational Biology*, edited by J. Jansson, C. Martín-Vide, and M. A. Vega-Rodríguez (Springer International Publishing, Cham, 2018) pp. 24–36.

[9] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[10] S. Linz and C. Semple, *Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies,* Advances in Applied Mathematics **105**, 102 (2019).

[11] J. Döcker, L. van Iersel, S. Kelk, and S. Linz, *Deciding the existence of a cherry-picking sequence is hard on two trees,* Discrete Applied Mathematics **260**, 131 (2019).

[12] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[13] R. Huijsman, *Tree-child Network Containment*, Bachelor's thesis, Delft University of Technology (2019).

[14] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[15] G. Cardona, J. C. Pons, and C. Scornavacca, *Generation of binary tree-child phylogenetic networks,* PLoS computational biology **15**, e1007347 (2019).

[16] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing tree-child networks from reticulate-edge-deleted subnetworks,* Bulletin of mathematical biology **81**, 3823 (2019).

[17] A. R. Francis and M. Steel, *Which phylogenetic networks are merely trees with additional arcs?* Systematic biology **64**, 768 (2015).

[18] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, and C. Semple, *Rooting for phylogenetic networks,* arXiv preprint arXiv:1906.07430 (2019).

[19] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A unifying characterization of tree-based networks and orchard networks using cherry covers,* Advances in Applied Mathematics **129**, 102222 (2021).

[20] R. Janssen, M. Jones, and Y. Murakami, *Combining networks using cherry picking sequences,* in *International Conference on Algorithms for Computational Biology* (Springer, 2020) pp. 77–92.

5

A UNIFIED STRUCTURAL CHARACTERIZATION OF ORCHARD AND TREE-BASED NETWORKS

Phylogenetic networks are often categorized into classes by their topological features, which stem from both biological and computational motivations. We study the structural characterization of two network classes in this chapter: tree-based networks and orchard networks. Tree-based networks are those that can be obtained by inserting edges between the edges of an underlying tree. Orchard networks are those that may be reduced to a network on a single leaf by a sequence of reductions on substructures induced by two leaves. Orchard networks were introduced in Chapter 4. Structural characterizations have already been discovered for tree-based networks; this is not the case for orchard networks. In this chapter, we introduce cherry covers—a unifying characterization of both network classes—in which we decompose the edges of the networks into so-called cherry shapes and reticulated cherry shapes. We show that cherry covers can be used to characterize the class of tree-based networks as well as the class of orchard networks. Moreover, we also generalize these results to non-binary networks. Finally, we give a biologically meaningful interpretation of orchard networks, by showing that orchard networks admit a special type of time labelling on the vertices, called HGT-consistent labellings. This characterization shows that orchards are merely trees with horizontal edges added.

5.1. INTRODUCTION

PHYLOGENETIC networks have been categorized into many topological classes for both biological and computational incentives (for an overview of a few binary network classes, see, for example, [2]). One of the largest of these network classes is the class of

The contents of this chapter have been published in Advances in Applied Mathematics **AAM**, 102222 (2021) [1]

tree-based networks. Hatched from an ongoing debate on whether evolutionary histories should or should not be viewed as tree-like with reticulate events sprinkled in (e.g., in the context of horizontal gene transfer within prokaryotes [3]), tree-based networks were introduced as those that can be obtained from trees by inserting new reticulate edges between the edges of the tree [4]. In their seminal paper, Francis and Steel explored the mathematical properties of these tree-based networks and provided a linear time algorithm to check whether a binary network was tree-based. Following this, structural characterizations for binary tree-based networks were introduced in the form of forbidden substructures [5], matchings [6], and using antichains and path-partitions [7]. Jetten and van Iersel further extended the matching characterization result to non-binary networks, and showed that it is possible to decide whether a non-binary network is tree-based in polynomial-time [6].

Within the class of binary tree-based networks lies the recently introduced class of binary orchard networks (shown in [8]). These networks generalize the prominent class of tree-child networks. It was shown that orchard networks are uniquely reconstructible from their ancestral profiles [9] and that it can be determined whether two binary (or semi-binary stack-free) orchard networks are isomorphic in linear time [10]. Orchard networks contain either a cherry (two leaves with a common parent) or a reticulated cherry (two leaves with distinct parents, for which one parent is the parent of another, and the lower parent is a reticulation), such that reducing a cherry or a reticulated cherry yields an orchard network of smaller size. With this reduction, we saw in the previous chapter that one can obtain a sequence of ordered pairs—which corresponds to reducing either a cherry or a reticulated cherry that involves the two leaves in the pair—that iteratively reduces the orchard network to a single leaf. Janssen and Murakami, and Erdős et al. have independently shown that such a reduction can be done in any order, and therefore that it can be decided in linear time whether a network is orchard [9, 10]. While these sequences do characterize orchard networks, the recursive nature of this characterization may make it impractical to use.

In this chapter, we present a unified structural (non-recursive) characterization for both non-binary tree-based networks and non-binary orchard networks. We first decompose networks into so-called *cherry shapes* and *reticulated cherry shapes*. If each edge of the network belongs to at least one of these two structures, then we say that the network has a *cherry cover*. This turns out to be a necessary and a sufficient condition for the network to be tree-based (Theorem 17). In addition we consider an ordering on the cherry and reticulated cherry shapes of a network. We prove that a network is orchard precisely if it has an acyclic cherry cover (Theorem 18). In particular, this shows that the class of non-binary orchard networks are contained in the class of non-binary tree-based networks (Corollary 9). Finally, we introduce the HGT-consistent labelling of networks and show that binary orchard networks are exactly those that admit an HGT-consistent labelling (Theorem 19). This gives another proof for showing that binary orchard networks are contained in the class of binary tree-based networks. We extend the labelling characterization for non-binary orchard networks, by means of resolving the network and finding a labelling for the resolution.

We also give a biologically meaningful interpretation of orchard networks by giving a characterization based on a certain time labelling of the network class, called *HGT-*

consistent labellings. Networks that admit an HGT-consistent labelling can be seen as those that can be obtained from some base tree with only horizontal edges added, which can explain reticulate phenomena such as horizontal gene transfer. The time labelling forces one parent of each reticulation to be contemporaneous with the reticulation vertex, thereby signalling how the genes may have been inherited. This is similar, though different to the *temporal labellings* that have been studied prior [11, 12].

5.2. PRELIMINARIES

RECALL that two networks N and M on X are *isomorphic* if there exists a bijection f that maps the vertices and edges of N to the vertices and edges of M, such that uv is an edge of N if and only if $f(u)f(v)$ is an edge of M, and leaves are mapped to leaves of the same label. A *semi-binary resolution* of a network N is a semi-binary network N', from which a network isomorphic to N can be obtained by contracting edges. A *binary resolution* of a network N is a binary network N', from which a network isomorphic to N can be obtained by contracting edges. Observe that a non-binary network generally has multiple non-isomorphic (binary and semi-binary) resolutions.

5.2.1. CHERRY COVER

A *cherry shape* is a subgraph on three distinct vertices x, y, p with edges px and py. The *internal vertex* of a cherry shape is p, and the *endpoints* are x and y. A *reticulated cherry shape* is a subgraph on four distinct vertices x, y, p_x, p_y with edges $p_x x, p_y p_x, p_y y$, such that p_x is a reticulation in the network. The *internal vertices* of a reticulated cherry shape are p_x and p_y, and the *endpoints* are x and y. The *internal reticulation* and the *middle edge* of a reticulated cherry shape are p_x and $p_y p_x$, respectively. The edge $p_y y$ is called the *free edge* of the reticulated cherry shape. We will often refer to cherry shapes and the reticulated cherry shapes by their edges (e.g., we would denote the above cherry shape $\{px, py\}$ and the reticulated cherry shape $\{p_x x, p_y p_x, p_y y\}$). We say that an edge uv is *covered* by a cherry or reticulated cherry shape C if $uv \in C$. Given a set P of cherry and reticulated cherry shapes, we say that an edge is *covered* by P if the edge is covered by at least one shape in P. We now investigate how sets of cherry shapes and reticulated cherry shapes may form a *decomposition* or *cover* for a given binary, semi-binary, or non-binary network (see Figure 5.1).

BINARY NETWORKS

Definition 15. A *cherry decomposition* of a binary network is a set P of cherry shapes and reticulated cherry shapes, such that each edge except for the root edge is covered exactly once by P.

We recall the following key lemma on the number of edges and vertices for each vertex type in a binary network.

Lemma 48 (Lemma 2.1 of [13])**.** *Let N be a binary network on n leaves and reticulation number r. Then N contains $n+r-1$ tree vertices and $2n+3r-1$ edges.*[1]

[1] Note that networks in [13] have roots of indegree-0 and outdegree-2 and thus are differently defined to the

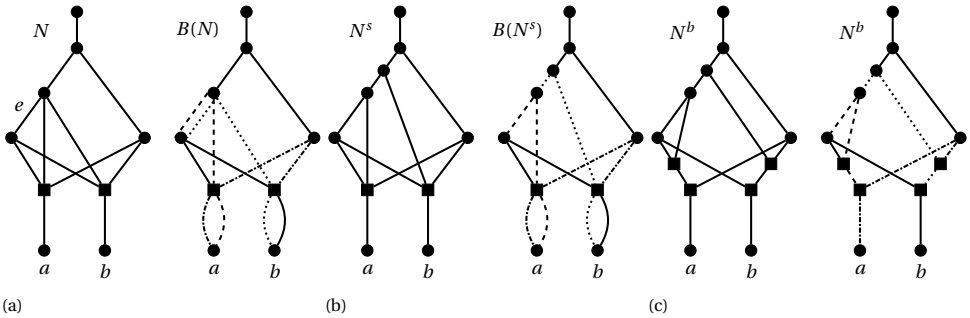


Figure 5.1: Examples of networks and their bulged versions with cherry covers and decompositions. All edges in networks are directed downwards from the root to the leaves, and reticulations are indicated by square vertices. (a) A non-binary network N and its bulged version $B(N)$. Observe that both leaves a, b are incident to parallel edges in $B(N)$, because both leaves are children of reticulation vertices with indegree-3. A cherry cover of $B(N)$ is visualized using different edge types. The edge e in N is duplicated in $B(N)$ to depict what happens when an edge is covered twice by a cherry cover. However, it does not represent parallel edges. (b) A semi-binary resolution N^s of N, obtained by resolving the multifurcation in N. The bulged version of N^s is shown on the right, together with a cherry decomposition of $B(N^s)$. (c) A binary resolution N^b of N. A cherry decomposition of $B(N^b) = N^b$ is displayed on the right network.

Lemma 49. *Let N be a binary network on n leaves and reticulation number r, and let P be a cherry decomposition of N. Then P contains exactly $n-1$ cherry shapes and r reticulated cherry shapes.*

Proof. By Lemma 48, the total number of edges in N is $2n+3r-1$. Then the total number of edges of N excluding the root edge is $2(n-1)+3r$. Recall that every outgoing edge of a reticulation vertex must be covered by a reticulated cherry shape. Indeed, since reticulations have one unique child, no outgoing edge of a reticulation vertex can be covered by a cherry shape. Since there are r such edges and because a reticulated cherry shape is composed of 3 edges, we have that $3r$ of the edges of N are covered by reticulated cherry shapes, and that the rest of the edges of N must be covered by cherry shapes. As each cherry shape is composed of 2 edges, and since every tree vertex in semi-binary networks are bifurcations, there must be $n-1$ cherry shapes in P. We conclude that P contains exactly $n-1$ cherry shapes and r reticulated cherry shapes. □

SEMI-BINARY NETWORKS

We extend the notion of a cherry decomposition to semi-binary networks by introducing the following "bulged version" of a network.

Definition 16. Let N be a network. Then the *bulged version* of N, $B(N)$, is the multigraph obtained from N by replacing the outgoing edge of each reticulation vertex with indegree-k by $k-1$ parallel edges. In $B(N)$, we call a vertex a *root* if it is a vertex of indegree-0 and outdegree-1, a *tree-vertex* if it has exactly one parent and at least two children, a *reticulation* if it has at least two parents and exactly one child, and a *leaf* if it

networks used in this chapter. However this is a technicality; their counting argument can be used in our network by tweaking values.

is labelled. In particular, tree vertices with two children are called *bifurcations* and tree vertices with more than two children are called *multifurcations*.

This action merely adds new edges between existing parent child pairs in the network; it does not add any new vertices. The edges added when obtaining the bulged version $B(N)$ of N are all parallel edges. Because of this, we observe that a vertex is a tree-vertex, a reticulation, or a leaf in N if and only if it is a tree-vertex, a reticulation, or a leaf in $B(N)$. We now define the reverse action to finding a bulged version of a network.

Definition 17. Let G be a directed acyclic multigraph. Then the *un-bulged version* $U(G)$ of G is the multigraph obtained from N by deleting all but one edge from each collection of parallel edges.

Lemma 50. *Let N be a non-binary network, and let $B(N)$ denote the bulged version of N. Then $U(B(N))$ is isomorphic to N.*

Proof. The multigraph $B(N)$ is obtained from N by adding parallel edges. Because of this, we may define a mapping f from the vertices and the edges of N to the vertices and edges of $B(N)$ such that if uv is an edge in N, then $f(u)f(v)$ is also an edge in $B(N)$, and further that f preserves leaf labels. Clearly, the mapping f uses every edge of $B(N)$ that is not a parallel edge; for each collection of parallel edges, the mapping uses exactly one edge.

Consider the graph $U(B(N))$ obtained by deleting all but one edge from each collection of parallel edges in $B(N)$. The choice for which parallel edges are deleted does not matter in this process, so choose to delete the edges that are not used in the mapping. Then f can be naturally extended to become a mapping of N into $U(B(N))$, where every edge of $U(B(N))$ is used. But this means that N and $U(B(N))$ must be isomorphic. □

When we restrict the domain to the set of non-binary phylogenetic networks and the codomain to the image of the domain under B, it is easy to see that U is the inverse of B. Therefore, we shall denote U as B^{-1} from here onwards. If N is binary, we have $N = B(N)$, but, in general, bulged versions of networks are not always networks, since they may contain parallel edges and vertices not listed in the definition of networks.

Lemma 51. *Let N be a semi-binary network on n leaves with reticulation number r. Then $B(N)$ has $2n+3r-1$ edges, r of which are out-edges of reticulation vertices.*

Proof. Let V_r be the set of reticulation vertices in N, and let $k = |V_r|$. Any binary resolution of N has the same number of tree vertices as N. By Lemma 48, N has n leaves, 1 root, k reticulation vertices, and $n+r-1$ tree vertices. Note that there are k outgoing edges of reticulation vertices in N and the sum of the indegrees of the reticulation vertices is $r+k$. Because in constructing $B(N)$, we add $\sum_{v\in V_r}(|\Gamma^-(v)|-2) = r+k-2k$ edges to N, the sum of the outdegrees of the reticulation vertices in $B(N)$ is $k+(r+k-2k) = r$. Hence, we can count the number of edges in $B(N)$ by counting the total number of outgoing edges for each vertex type: the leaves have 0 outgoing edges, the root has 1 outgoing edge, the tree vertices have $2(n+r-1)$ outgoing edges, and the reticulation vertices have r outgoing edges. Therefore, we conclude that $B(N)$ has $1+2(n+r-1)+r = 2n+3r-1$ edges. □

Definition 18. A *cherry decomposition* of the bulged version of a semi-binary network N is a set P of cherry shapes and reticulated cherry shapes, such that each edge of $B(N)$, except for the root edge, is covered exactly once by P.

Observe that a reticulation vertex in the bulged version of the network is always mapped to an internal reticulation of a reticulated cherry shape in the cherry decomposition. This brings us to the following lemma, whose proof follows an analogous argument as used in the proof of Lemma 49.

Lemma 52. *Let N be a semi-binary network on n leaves and reticulation number r, and let P be a cherry decomposition of N. Then P contains exactly $n-1$ cherry shapes and r reticulated cherry shapes.*

Proof. The bulged network $B(N)$ has $2n+3r-1$ edges (Lemma 51). Then the total number of edges of $B(N)$ excluding the root edge is $2(n-1)+3r$. Observe that every outgoing edge of a reticulation vertex must be covered by a reticulated cherry shape, and each reticulated cherry shape must cover such an edge. Since there are r such edges (Lemma 51) and because a reticulated cherry shape is composed of 3 edges, we have that $3r$ of the edges of $B(N)$ are covered by reticulated cherry shapes, and that the rest of the edges of $B(N)$ must be covered by cherry shapes. As each cherry shape is composed of 2 edges, this implies that there must be $n-1$ cherry shapes in P. Therefore P contains exactly $n-1$ cherry shapes and r reticulated cherry shapes. □

5

NON-BINARY NETWORKS

For non-binary networks, we generalize the concept of cherry decompositions by allowing certain edges to be covered multiple times.

Definition 19. A *cherry cover* of (the bulged version) of a non-binary network N is a set P of cherry shapes and reticulated cherry shapes with the following properties on $B(N)$:

- each edge except for the root edge is covered by at least one shape in P,
- each outgoing edge of a reticulation vertex is covered exactly once,
- each edge covered by the middle edge of a reticulated cherry shape is covered exactly once.

Note that cherry covers may contain cherry shapes that cover the same edge of the bulged version of the network, as long as the above properties are respected (see Figure 5.2). Note also that there may exist many distinct cherry covers for one network.

Lemma 53. *Let P be a cherry cover of a non-binary network N, and let uv be an edge of $B(N)$ that is covered by at least two shapes in P. Then u must be a multifurcation in N.*

Proof. First observe that u cannot be the root since the root edge is not covered by P, and it also cannot be a vertex of outdegree-0. Furthermore, u cannot be a reticulation vertex by the second condition of Definition 19. Therefore u must be a tree vertex. Suppose that u is a bifurcation, and let uw be an edge of $B(N)$ that is not uv. Then the edges uv and uw must be contained in a same shape A in P. If A was a reticulated cherry shape,

Figure 5.2: Cherry covers of sizes 3 (left) and 2 (right) for the same tree. We duplicate the edges incident to b and c to show how an edge can be covered more than once in a cherry cover. The cherry cover of the left tree reflects the cherry cover used in the proof of Lemma 54.

then one of uv or uw must form the middle edge of A; by the third condition of the cherry cover definition, no other shape of P can contain the edge uv. On the other hand, if A was a cherry shape, then for uv and uw to be covered by a shape B that is not A, B must be a reticulated cherry shape. But this would again violate the third condition of the cherry cover definition. Thus, no other shape of P can contain the edge uv. Therefore, the edge uv is covered only by one shape in P, and u cannot be a bifurcation. By process of elimination, it follows that u must be a multifurcation. □

It follows that cherry covers are indeed a generalization of cherry decompositions, since a cherry cover of a binary or a semi-binary network covers each edge of the bulged version of the network exactly once. Observe that the converse of Lemma 53 is not necessarily true. That is, given a cherry cover of a network, it is not always the case that a multifurcation has an outgoing edge that is covered more than once (see Figure 5.2).

Lemma 54. *Let N be a network on n leaves. Then $B(N)$ has a cherry cover using only cherry shapes if and only if N is a tree. Furthermore, if N is a tree, then there exists a cherry cover of N that contains exactly $n-1$ cherry shapes.*

Proof. The first statement follows from the definition of a cherry cover. To prove the second statement, we construct a cherry cover for N as follows. Let t be a tree vertex in N of outdegree-d. Arbitrarily enumerate the d outgoing edges of t by $e_1, e_2, \ldots, e_d$, and define cherry shapes $C_{t_i} = \{e_i, e_{i+1}\}$ for $i \in [d-1] = \{1, \ldots, d-1\}$. These $d-1$ cherry shapes cover all outgoing edges of t. We repeat this for all tree vertices, and since the tail of every edge, except for the root edge, is a tree vertex, we obtain a cherry cover.

Let $T(N)$ denote the tree vertices of N. Since the sum of the indegrees is equal to the sum of the outdegrees, we get that

$$n + |T(N)| = \sum_{v \in N} |\Gamma^-(v)| = \sum_{v \in N} |\Gamma^+(v)| = 1 + \sum_{t \in T(N)} |\Gamma^+(t)|.$$

Rearranging this equation, we find

$$\sum_{t \in T(N)} |\Gamma^+(t)| - |T(N)| = n - 1.$$

In the construction of a cherry cover of T above, each tree vertex t gives $|\Gamma^+(t)| - 1$ cherry shapes. Hence, the size of the cherry cover is exactly $\sum_{t \in T(N)} |\Gamma^+(t)| - |T(N)| = n - 1$. □

Definition 20. Let P be a cherry cover of some network. A shape $A \in P$ is *directly above* another shape $B \in P$ if an internal vertex of B is an endpoint of A. A shape $A \in P$ is *above* a shape $B \in P$ if there is a sequence $A = A_0, \ldots, A_n = B$ such that A_{i-1} is directly above A_i for all $i \in [n]$. The cherry cover P is called *acyclic* if no shape is above itself.

Given a cherry cover of some network, Definition 20 naturally gives rise to an auxiliary graph where the cherry shapes and reticulated cherry shapes are the vertices and an edge is drawn from one shape to another if it is directly above the shape. It can be used to determine the acyclicity of a cherry cover. An example of such a graph can be seen in Figure 5.3c.

5.2.2. NETWORK CLASSES

Tree-based networks For the definition of tree-based networks, see Section 2.1.4. See Figure 5.3 for an example of a tree-based network, its bulged version, and a cherry cover for the network.

Recall that tree-based networks are defined as those that can be obtained by adding vertices and edges to a tree, which we call the base tree. Given a tree-based network N, we may reverse this action by removing a subset E_r of the edges and suppressing all indegree-1 outdegree-1 vertices until we obtain a base tree T (note that E_r may not necessarily be unique). Letting $V(N)$ and $E(N)$ denote the vertices and the edges of N respectively, we define the *embedding* of T in N by the subgraph of N with vertex set $V(N)$ and edge set $E(N) \setminus E_r$. Observe that suppressing all indegree-1 outdegree-1 vertices from the embedding of T in N returns the tree T. Since a tree-based network may have more than one unique base tree, such a set E_r is not necessarily unique.

Let N be a network on X. We say that the bulged version of N, $B(N)$, is *tree-based* if the leaves of some spanning tree of $B(N)$ are labelled bijectively by X. Because a spanning tree of $B(N)$ contains exactly one edge from each set of parallel edges, we come to the following observation.

Observation 10. *A network N is tree-based if and only if $B(N)$ is tree-based.*

Orchard networks For the definition of orchard networks, see Section 2.1.4. See Figure 5.4 for an example of an orchard network, its bulged version, and its acyclic cherry cover.

5.2.3. REDUCING SHAPES

To characterize orchard networks using cherry covers of bulged networks, we show that it is possible to reduce a pair in a network N by modifying its bulged version. To do so, we first define the process of removing a reducible pair from a bulged network.

Definition 21. Let (x, y) be a reducible pair in a network N with corresponding (reticulated) cherry shape A in $B(N)$. If the parent p_y of y is a bifurcation (resp. multifurcation), then *reducing* A in $B(N)$ consists of deleting each edge of A (resp. $A \setminus \{p_y y\}$) from $B(N)$, then deleting all isolated vertices, and finally, labelling all unlabelled outdegree-0 vertices by the label of one of their children in $B(N)$ before removal. The resulting bulged network is denoted $B(N) \setminus A$ (resp. $(B(N) \setminus (A \setminus \{p_y y\}))$).

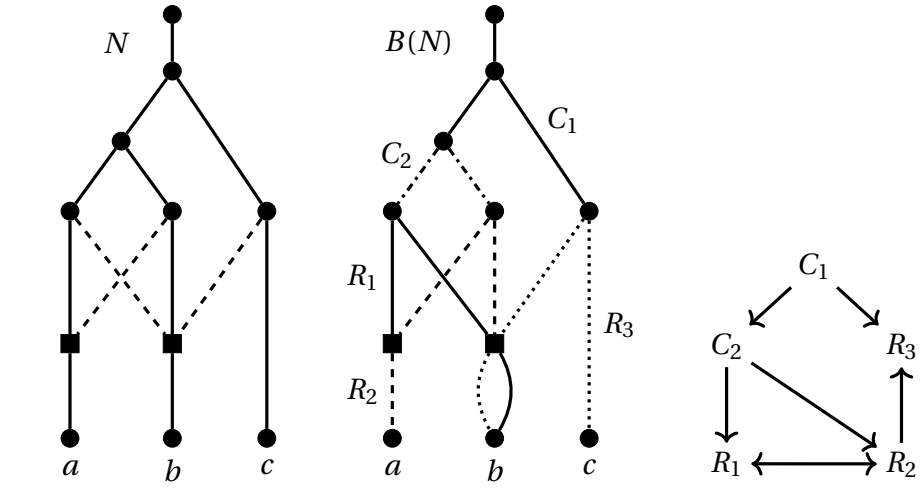


Figure 5.3: (a) A semi-binary tree-based network N that is not orchard. A base tree of N is indicated by the solid edges. (b) The bulged version of N with a cherry cover $\{C_1, C_2, R_1, R_2, R_3\}$. Each cherry shape is indicated using a distinct line type for the edges. (c) An auxiliary graph that shows the order on the cherry shapes. An edge is drawn from one cherry shape to another if it is directly above it. In this case, the cherry cover is not acyclic since $\{R_1, R_2\}$ form a cycle.

Only if A is a cherry shape and the common parent p_x of x and y is a bifurcation, we have multiple options for labelling the outdegree-0 vertex. To solve this, we reduce a cherry as an ordered pair (x, y), and we label the outdegree-0 vertex p_x with the label of y.

In this definition, we claim that the resulting graphs are bulged versions of networks. This follows from the fact that removing a reticulated cherry shape, the indegree and outdegree of a reticulation vertex both go down by one, so the number of parallel edges below the reticulation is still correct.

Finally, we prove that reducing a (reticulated) cherry in a network N is the same as reducing the corresponding (reticulated) cherry shape in $B(N)$.

Lemma 55. *Let (x, y) be a reducible pair in N, and let p_y denote the parent of y in N. Let A denote the cherry shape or the reticulated cherry shape of $B(N)$ corresponding to the reducible pair (x, y).*

- *If p_y is a bifurcation, then $N(x, y) = B^{-1}(B(N) \setminus A)$.*
- *If p_y is a multifurcation, then $N(x, y) = B^{-1}((B(N) \setminus (A \setminus \{p_y y\}))$.*

Proof. First suppose that (x, y) is a cherry. Recall that reducing (x, y) in N consists of first removing the edge of $p_y x$ and, if p_y is a bifurcation, additionally removing the edge $p_y y$ and relabelling p_y with the label of y in N. Hence, $N(x, y)$ can be obtained from N by removing every edge in $\{p_y x, p_y y\} = A$ from N and relabelling p_y with the label of y. If p_y was a multifurcation, then no suppression will happen, and $N(x, y)$ can be

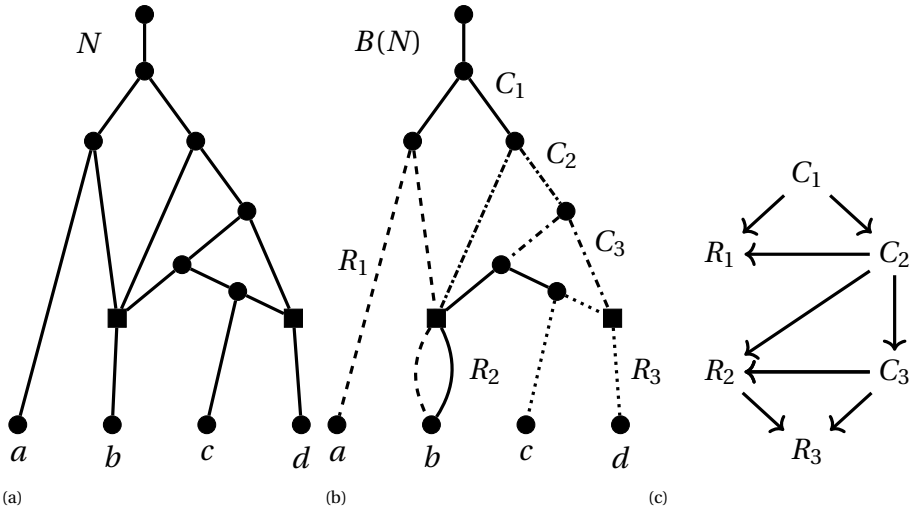


Figure 5.4: (a) A semi-binary orchard network N on taxa set $\{a,b,c,d\}$. One sequence for reducing N to a network on a single leaf is $(d,c)(b,a)(b,c)(d,c)(b,c)(a,c)$. (b) The bulged version of N with one possible cherry cover $\{C_1,C_2,C_3,R_1,R_2,R_3\}$. (c) An auxiliary graph that shows the order on the cherry shapes. In this case, the cherry cover is acyclic.

obtained from N by simply removing every edge in $\{p_yx\} = A \setminus \{p_y,y\}$ from N. As no reticulation vertices are involved here, we can equivalently remove these edges in $B(N)$, so we conclude that $N(x,y) = B^{-1}(B(N) \setminus A)$ or $N(x,y) = B^{-1}(B(N) \setminus (A \setminus \{p_yy\}))$ if p_y is a bifurcation or multifurcation respectively.

Now suppose that (x,y) is a reticulated cherry. Reducing (x,y) in N consists of removing p_yp_x and, if p_x (resp. p_y) is not binary, additionally removing p_xx (resp. p_yy) and relabelling p_x (resp. p_y) with the label of x (resp. y). In contrast to the case that (x,y) is a cherry, we must now consider the outgoing edges of p_x in $B(N)$ to see how we can equivalently remove the edges A (or $A \setminus \{p_yy\}$) from $B(N)$ instead of from N. If p_x is binary, we here remove the edge p_xx just like when we reduce (x,y) in $B(N)$, hence, there is a clear correspondence between these processes. If p_x is not binary, then the reduction in $B(N)$ removes an outgoing edge of p_x, whereas the reduction in N does not. This is compensated for by the fact that p_x has multiple outgoing edges in $B(N)$. Indeed, after removing one incoming edge of p_x in N, p_x should have should have one fewer outgoing edge in the bulged version of the resulting network. Hence, the edge p_xx needs to be removed from $B(N)$ as well to obtain the bulged version of $N(x,y)$. Hence, in the case that (x,y) is a reticulation, we also have $N(x,y) = B^{-1}(B(N) \setminus A)$ and $N(x,y) = B^{-1}(B(N) \setminus (A \setminus \{p_yy\}))$ when p_y is a bifurcation or multifurcation, respectively. □

5.3. TREE-BASED NETWORKS

In this section, we show that a binary network is tree-based if and only if it has a cherry decomposition. We do this by showing that for non-binary networks, the same char-

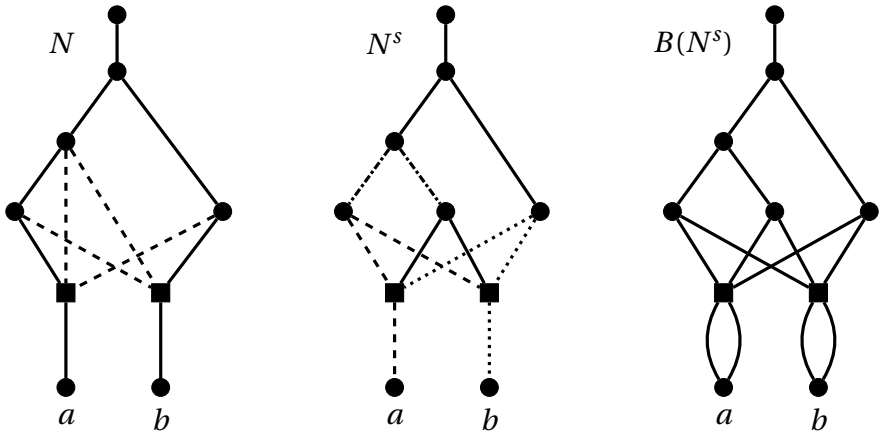


Figure 5.5: (a) A non-binary tree-based network N on $\{a,b\}$. A base tree is indicated by the solid edges. (b) A semi-binary resolution N^s of N that is not tree-based with a cherry cover. (c) The bulged version of N^s that does not have a cherry cover. This can be seen as follows. There are four edges incident to the leaves and each of these edges can only be covered by reticulated cherry shapes. However, it is not possible to add four such reticulated cherry shapes without covering any middle edge of a reticulated cherry shape more than once.

acterization holds if we look at cherry covers in the bulged version of the network. Taking the bulged version is crucial in this characterization. Figure 5.5b (from [6]) is an example of a (non-bulged) semi-binary network that is not tree-based with a cherry cover. In the same figure, we show that its bulged version does not have a cherry cover (Figure 5.5c), and also that contracting one of the edges in the network yields a non-binary network that is tree-based (Figure 5.5a). This latter point proves the following observation.

Observation 11. *Let N be a tree-based network. Then there may exist a semi-binary resolution of N that is not tree-based.*

Lemma 56. *Let N be a network. Then N is tree-based if and only if some binary resolution of N is tree-based. N is tree-based if and only if some semi-binary resolution of N is tree-based.*

Proof. The first statement follows from [6] Observation 3.2. To show the second statement, let N be a tree-based network, and let T be a base tree of N. By definition of base trees, T must visit every tree vertex in the network. In particular, it must visit every multifurcation, and exit such vertices via one of its outgoing edges. Let t denote such a tree vertex and let s denote the child of t in N such that ts is an edge that is used by T. Then we resolve t by replacing it by a caterpillar such that the parent of s is the bottom-most vertex. It remains to check that the base tree covers all the newly introduced vertices. However this is immediate; by the placement of s, we note that the path from t to s covers all the newly introduced vertices. Therefore the tree T with the edge ts changed to the path from t to s is a base tree of the new network. Repeating this for all multifurcations yields a semi-binary resolution of N that is tree-based.

On the other hand, if a semi-binary resolution N' of N is tree-based, then it is easy to see that N must also be tree-based. Indeed, upon contracting some of the edges of N', we adjust the base tree of N' by contracting an edge in the base tree if it used a path that was contracted, in the embedding, and not changing the base tree otherwise. In case a few edges of the path were contracted, we still map the edge of the base tree to the partially contracted edge. Doing so gives a base tree of N. □

Theorem 17. *A network N is tree-based if and only if $B(N)$ has a cherry cover.*

Proof. First suppose that N is a tree-based network. Let T be a base tree of N, and let $E_r = \{e_1, \ldots, e_k\}$ denote the reticulation edges that were deleted to obtain T from N. By Lemma 54, T has a cherry cover P consisting of only cherry shapes. We use this cherry cover to produce a cherry cover of N.

Each cherry shape C in P maps to a pair of paths c_1 and c_2 in $B(N)$ that are vertex-disjoint except at their highest vertices. All these paths together cover the edges of the embedding E_T of T in $B(N)$. Taking the first edge of both c_1 and c_2, we obtain a cherry shape $C|_N$ of $B(N)$. Let $P' = \{C|_N : C \in P\}$ be the set of cherry shapes in $B(N)$ obtained from cherry shapes in P, and let $F = E_T \setminus P'$ be the edges of E_T that are not covered by P'.

The edges of $B(N)$ apart from the root edge that are not yet covered by P' are as follows:

- the reticulation edges $e_i = u_i v_i \in E_r$,
- all outgoing edges of v_i for all $i \in [k]$,
- and for each u_i for all $i \in [k]$, at most one outgoing edge $f_{u_i} \in F$.

For the last point, if the endpoint u_i were to have more than one outgoing edges in F, then they would be part of a cherry shape in P'; hence, they cannot be in F, but they must be in P'. Therefore this case is not possible. If there is no outgoing edge of u_i contained in F, then u_i must have two outgoing edges that form a cherry shape in $B(N)$ that is contained in P'. Otherwise u_i would not have been covered by E_T, which would contradict the fact that T was a base tree of $B(N)$. If there was exactly one outgoing edge f_{u_i} of u_i contained in F, then u_i was not a tree vertex in T (in particular it must have been added as an attachment point). Thus, f_{u_i} is not a highest edge in the embedding of a cherry shape of P, so f_{u_i} is not covered by P'. Observe that f_{u_i} cannot be the reticulation edge e_i itself, since E_r contains all the reticulation edges that are not used in the embedding of T in N. Therefore, each endpoint u_i of a reticulation edge $e_i = u_i v_i \in E_r$ has an outgoing edge in F, or an outgoing edge that is covered by P'.

We augment P' to a cherry cover P'' of $B(N)$ by adding a reticulated cherry shape $\{v_i x_i, u_i v_i, u_i y_i\}$ for each $e_i = u_i v_i \in E_r$ satisfying the following conditions: for each i, either $u_i y_i \in F$ or $u_i y_i$ is covered by P', and for any $i \neq j$, $v_i x_i \neq v_j x_j$. This last condition is possible because the number of outgoing edges of a reticulation vertex v is equal to the number of incoming edges of v that are in E_r. By construction, P'' is a cherry cover of $B(N)$.

Now suppose that the bulged version of the network N has a cherry cover P. For every reticulation vertex v of indegree k, exactly $k-1$ incoming edges are contained in

a reticulated cherry shape as the middle edge in P. By definition of reticulated cherry shapes, the tail of each of these reticulation edges has at least one child other than v. This inherently implies that deleting these $k-1$ reticulation edges will not create any unlabelled outdegree-0 vertices. Repeating this deletion for all such reticulation edges and removing all parallel edges returns a spanning tree of the graph whose leaves are labelled bijectively by the leaf-set of N; therefore $B(N)$ is tree-based. By Observation 10, N is tree-based. □

By Lemma 54, there exists a cherry cover of a tree on n leaves that contains exactly $n-1$ cherry shapes. The next corollary follows immediately from this observation and the arguments used in the proof of Theorem 17.

Corollary 8. *Let N be a tree-based network on n leaves and reticulation number r. Then there exists a cherry cover of N that contains exactly $n-1$ cherry shapes and exactly r reticulated cherry shapes.*

5.4. ORCHARD NETWORKS

We now show that a network is orchard if and only if its bulged version has an acyclic cherry cover. Note that it is necessary to consider the bulged version of the network, as there exist networks that are not orchard that have an acyclic decomposition into cherry and reticulated cherry shapes, such as the network depicted in Figure 5.6. Note that the bulged version of this network has no acyclic cherry cover. To see this, observe that the edge incident to a must be covered by a reticulated cherry shape—say it is covered by a reticulated cherry shape containing R. In the bulged version of the network, there are parallel edges incident to the leaf b; one of these edges must be covered by a reticulated cherry shape containing the edges of R'. However, the shapes containing R and R' are then above one another, so no cherry cover can be acyclic.

In Figure 5.7, the network N is an orchard network, as $(a,b)(d,c)(b,c)(a,c)(d,c)$ is a sequence of reducible pairs that reduce N to a network on a single leaf c. The same figure presents a semi-binary resolution N^s of N that is not orchard. Since there are no reducible pairs (no cherries nor reticulated cherries) in N^s, it is immediately clear that N^s is not orchard. Therefore we obtain the following observation.

Observation 12. *Let N be an orchard network. Then there may exist a semi-binary resolution of N that is not orchard.*

Lemma 57. *Let N be a network where $B(N)$ has a cherry cover P. Suppose $A \in P$ is a lowest shape with endpoints x and y where the parent p_y of y is a tree vertex. Then,*

- *(x,y) is a reducible pair in N, and*
- *$B(N(x,y))$ has a cherry cover $P \setminus \{A\}$ if p_y is a bifurcation; otherwise, $B(N(x,y))$ has a cherry cover $(P \setminus \{A\}) \cup \{Z\}$, where Z is a shape with endpoint y.*

Proof. We first show that x and y are leaves in $B(N)$. Suppose for a contradiction that x is not a leaf. Then it is either a tree vertex or a reticulation vertex. In either case, x has an

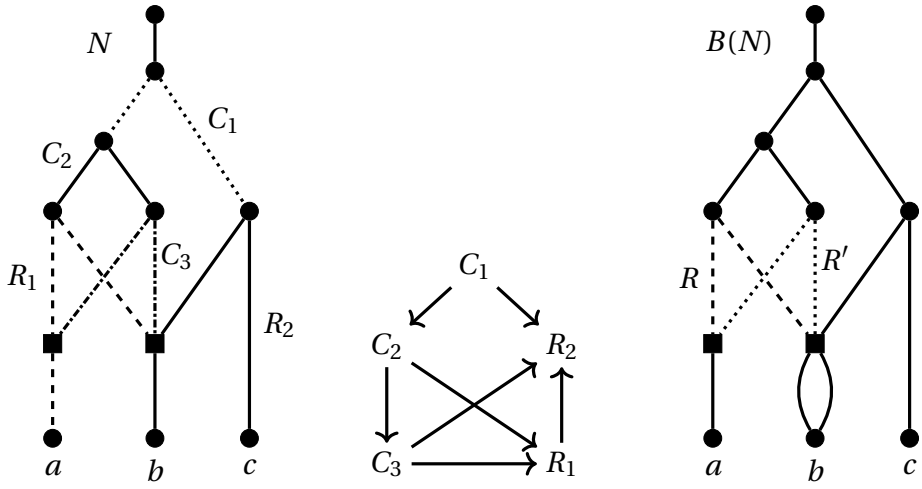


Figure 5.6: An example showing why it is necessary to consider cherry covers in bulged versions of networks. The tree-based network N (also shown in Figure 5.3(a)) is not orchard. Nevertheless, there is an acyclic decomposition of N into the cherry and reticulated cherry shapes $\{C_1, C_2, C_3, R_1, R_2\}$. Every cherry cover of $B(N)$ must be cyclic, because each of the edges labelled R and R' must be contained in a reticulated cherry shape whose endpoint is the leaf a or b. These shapes will be directly above one another, creating a cycle in the auxiliary graph.

outgoing edge which must be part of some shape $Y \in P$. As x is not a lowest vertex in Y, x must be an internal vertex of Y. This implies that A is above Y, which contradicts the fact that A is a lowest shape. Hence, x must be a leaf. By the same argument, y is a leaf. Hence, x and y are both leaves of N. We now split into two cases: either A is a cherry shape, or A is a reticulated cherry shape.

First suppose $A = \{p_y x, p_y y\}$ is a cherry shape. As $B(N)$ has edges $p_y x$ and $p_y y$, N must also have such edges. As N has edges $p_y x$ and $p_y y$, and x and y are leaves, N has the cherry (x, y). This means (x, y) is a reducible pair in N. Now suppose $A = \{p_x x, p_y p_x, p_y y\}$ is a reticulated cherry shape. Then N also contains edges $p_x x$, $p_y p_x$, and $p_y y$. As x and y are leaves in N and p_x is a reticulation vertex—by the properties of a cherry cover—(x, y) is a reticulated cherry in N, which is a reducible pair. This proves the first part of the lemma.

For the second part of the lemma, we split the proof into two subcases. First suppose that p_y is a bifurcation. The first part of the current lemma implies that A corresponds to the reducible pair (x, y) of N, so by Lemma 55, we have $N(x, y) = B^{-1}(B(N) \setminus A)$. Moreover, by assumption, P is a cherry cover of $B(N)$, A is an element of P. Hence, it follows that the set $P \setminus \{A\}$ is a cherry cover of $B(N(x, y)) = B(B^{-1}(B(N) \setminus A)) = B(N) \setminus A$.

Now suppose that p_y is a multifurcation. Then, the first part of this lemma again implies that A corresponds to the reducible pair (x, y), so $B(N(x, y)) = (B(N) \setminus A) \cup \{p_y y\}$ by Lemma 55. Moreover, $P \setminus \{A\}$ covers all edges of $B(N) \setminus A$, so only the edge $p_y y$ may not be covered by $P \setminus \{A\}$. If the edge $p_y y$ is covered by $P \setminus \{A\}$, then this is a cherry cover of $(B(N) \setminus A) \cup \{p_y y\}$ and therefore of $B(N(x, y))$ and we are done. So suppose $p_y y$ is not covered by $P \setminus \{A\}$. Excluding the edge $p_y y$, if all other outgoing edges of p_y formed

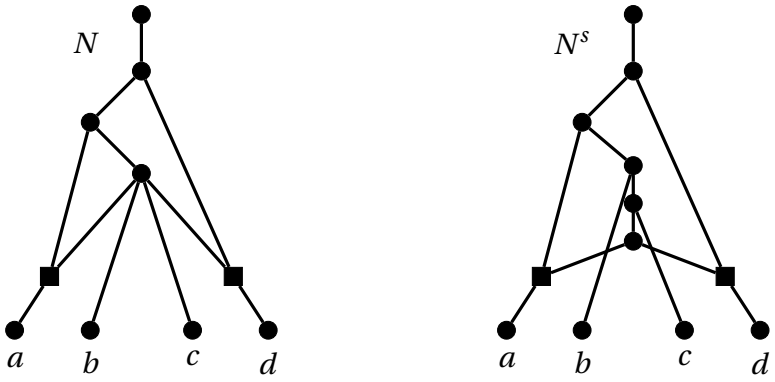


Figure 5.7: An orchard network N and a non-orchard semi-binary resolution N^s of N.

the middle edge of reticulated cherry shapes, then $p_y y$ must have formed the free edge of each of these reticulated cherry shapes. This implies that the edge $p_y y$ must already have been covered by $P \setminus \{A\}$, which is not true by our assumption. Therefore, there exists some outgoing edge $p_y z$ of p_y that is covered by $P \setminus \{A\}$, such that $p_y z$ does not form the middle edge of a reticulated cherry shape. Then, we obtain a cherry cover $(P \setminus \{A\}) \cup \{p_y y, p_y z\}$ of $(B(N) \setminus A) \cup \{p_y y\}$ and therefore of $B(N(x,y))$. □

Theorem 18. *A network N is orchard if and only if $B(N)$ has an acyclic cherry cover.*

Proof. First suppose that a network N is orchard. We prove by induction on the sum $S = n + r$ of the number of leaves n and the reticulation number r of N that $B(N)$ has an acyclic cherry cover. The induction basis is the case with one leaf and no reticulations: the empty set is an acyclic cherry cover for such a network.

Now suppose that for each orchard network N' with $n' + r' = S'$, $B(N')$ has an acyclic cherry cover. We prove that for any network N with $n + r = S = S' + 1$, $B(N)$ has an acyclic cherry cover. For this purpose, let N be an orchard network with n leaves and r reticulations, such that $n + r = S = S' + 1$, and let (x, y) be a reducible pair in N. Note that as N is an orchard network, such a reducible pair must exist.

First suppose that the parent p_y of y is a bifurcation. By Lemma 55, we have that $B(N(x,y)) = B(N) \setminus A$, where A is a cherry shape if (x, y) is a cherry, and A is a reticulated cherry shape if (x, y) is a reticulated cherry. By definition of orchard networks and reductions of reducible pairs, $N(x, y)$ is an orchard network and the sum of its leaves and reticulations is S'. By the induction hypothesis, $B(N(x, y))$ has an acyclic cherry cover P. We may obtain a cherry cover for $B(N)$ by appending the shape A to P. Therefore $B(N)$ has a cherry cover $P \cup \{A\}$. As the endpoints of A are leaves, the element A is above no other shape in P. Therefore the cherry cover $P \cup \{A\}$ is acyclic.

Now suppose that the parent of p_y is a multifurcation. By Lemma 55, $B(N(x,y)) = (B(N) \setminus A) \cup \{p_y y\}$, where A is either a cherry shape or a reticulated cherry shape on (x, y). We have again that $N(x, y)$ is an orchard network, and the sum of its leaves and reticulations is S'. By the induction hypothesis, this implies that there is an acyclic cherry cover P of $B(N(x,y)) \cup \{p_y y\}$, which gives a cherry cover $P \cup \{A\}$ of $B(N)$. This cherry cover is

acyclic because the new element A is above no other shape as its endpoints are leaves.

Hence, for each orchard network N with a total $S'+1$ of leaves and reticulations, there is an acyclic cherry cover of $B(N)$.

To prove the other direction of the theorem, suppose that $B(N)$ has an acyclic cherry cover P and let $A \in P$ be a lowest shape with endpoints x and y. Observe that such a lowest shape must exist as otherwise the cherry cover would not be acyclic. By Lemma 57, (x, y) is a reducible pair in $B(N)$, and $B(N(x, y))$ has a cherry cover $(P \setminus \{A\}) \cup \{Z\}$ or $P \setminus A$, in which the order on the remaining shapes is not changed. This means $B(N(x, y))$ is smaller than $B(N)$, and it has an acyclic cherry cover. This process continues until $P = \emptyset$, and both N and $B(N)$ are reduced to a single leaf network. Since we have successively reduced cherries or reticulated cherries from N to obtain a single leaf network, N is an orchard network. □

We now prove a lemma that is analogous to Lemma 56 for orchard networks using acyclic cherry covers.

5

Lemma 58. *Let N be a network. Then N is orchard if and only if some binary resolution of N is orchard. Similarly, N is orchard if and only if some semi-binary resolution of N is orchard.*

Proof. We first assume that there exists some binary resolution N^b of N that is orchard, and independently, that there exists some semi-binary resolution N^s of N that is orchard. We claim that contracting an edge of an orchard network whose head and tail are both tree vertices or both reticulation vertices results in an orchard network. By definition, we may obtain N by contracting exactly these edges from N^b and from N^s (different edges for the two resolutions), from which it follows that N is orchard. We now prove the claim.

Let M be an orchard network, and let st be an edge in M such that s and t are both tree vertices. We show that the network obtained by contracting st in M is orchard. By Theorem 18, M has an acyclic cherry cover P. The edge st is covered as an edge in a cherry shape or as a free edge in a reticulated cherry shape in P (or possibly both and multiple times, if s is a multifurcation). Moreover, at least one of the outgoing edges of t is also covered as an edge in a cherry shape or as a free edge in a reticulated cherry shape in P. Let us denote this edge by tu. We now contract the edge st, and replace the edge st that appeared in every shape in P by tu. All other shapes of P are preserved and we call this new set of shapes P'. All edges of the contracted network are covered and it is easy to check that P' is a cherry cover. It remains to show that P' is an acyclic cherry cover. This follows immediately, because the shapes in P that contained the edge st are no longer directly above the shapes in P that contained the vertex t as an internal vertex; furthermore, the shapes in P that contained the edge st are now directly above the shapes in P that contained the vertex u as an internal vertex. These new edges do not create a cycle in the auxiliary graph, as otherwise P would have been cyclic.

Now let pq be an edge in M such that p and q are both reticulations. By definition of cherry covers, there must exist one incoming edge kp of p such that kp is covered as an edge in a cherry shape or as a free edge in a reticulated cherry shape $A \in P$. Let r be the child of q. We now contract the edge pq, calling the new vertex q', and replace

the edge pq that appeared in every shape in P by an edge $q'r$. All other shapes of P are simply kept, and we call this new set of shapes P'. Note that the number of $q'r$ edges in shapes of P' is equal to $(|\Gamma^-(p)|-1)+(|\Gamma^-(q)|-1) = |\Gamma^-(q')|-1$, which is the number of outgoing edges of q' in $B(N)$ after contraction. Hence, P' forms a cherry cover of the contracted network.

Moreover, P' is acyclic for the following reason. The only difference between the auxiliary graph of P and the auxiliary graph of P' is that the arrow between shapes of P containing pq and the shapes of P containing qr has been deleted, and the arrow from A to shapes of P containing qr has been added. But A was already above these shapes in the auxiliary graph of P. The same can be said for all reticulated cherry shapes that covered an incoming edge of p as the middle edge. Hence, contracting an edge of an orchard network whose head and tail are both tree vertices or both reticulation vertices returns an orchard network. Therefore the network N is orchard.

To prove the other direction, suppose that N is an orchard network. By Lemma 2 in [10], there exists a binary resolution of N that is orchard. Since any binary network is also semi-binary, the binary resolution of N is also semi-binary. □

It was shown in [8] that binary orchard networks are tree-based. It follows from Theorems 17 and 18 that this is also true for the non-binary case.

Corollary 9. *All orchard networks are tree-based.*

5.5. HGT TIME-CONSISTENCY FOR ORCHARD NETWORKS

In this section, we provide another characterization of orchard networks, by means of a time labelling on the vertices of the network. We give this characterization first for binary orchard networks, and later extend the characterization to non-binary orchard networks.

5.5.1. BINARY ORCHARD NETWORKS

Definition 22. Let N be a binary phylogenetic network with vertex set V. An *HGT-consistent labelling* of N is a labelling $t : V \to \mathbb{R}$ such that

1. For all edges uv, $t(u) \leq t(v)$ and equality is only allowed if v is a reticulation.
2. For each non-leaf vertex u, there is at least one child v of u such that $t(u) < t(v)$.
3. For each reticulation r with parents u and v, exactly one of $t(u) = t(r)$ and $t(v) = t(r)$ holds.

The conditions listed above in Definition 22 will be referred to as conditions 1,2, and 3, respectively. The biological interpretation of a network with an HGT-consistent labelling is that reticulations are caused within the network by horizontal gene transfers. The third condition above ensures that reticulation vertices are contemporaneous with one of their parents – in particular, with the parent from which genetic material is passed via horizontal gene transfer. This labelling is different to the *temporal labelling* of networks as defined in Section 2.1.4, as in a temporal labelling, all parents of a reticulation

vertex, as well as the reticulation vertex itself, are contemporaneous. The next lemma shows that networks that admit an HGT-consistent labelling are tree-based, where the network can be obtained by adding horizontal edges to some base tree.

Lemma 59. *Each binary network that admits an HGT-consistent labelling is tree-based. In particular, a base tree can be obtained by deleting all reticulation edges where the tail and head have the same time labels.*

Proof. We prove the claim by induction on the number of vertices. For the base case, let N be a binary tree; the claim is trivially true. So suppose that N is a binary network on n vertices, and that the claim holds for every binary network on fewer than n vertices. Let $t : V \to \mathbb{R}$ denote the HGT-consistent labelling of N. Consider a reticulation edge uv where $t(u) = t(v)$. By condition 2 of HGT-consistent labelling, we know that the vertex u must be a tree vertex. Delete the edge uv and suppress the vertices u, v to obtain a network N'. We claim that restricting the labelling t to the vertices of N', which we shall call t' is an HGT-consistent labelling. The parts of N' that we must check carefully are the vertices in the neighbourhood of u and v in N; all other vertices and edges inherit their HGT-consistency directly from the fact that t is an HGT-consistent labelling of N.

Recalling that u is a tree vertex and v a reticulation vertex, let a be the parent of u, let b be the child of u that is not v, let c be the parent of v that is not u, and let d be the child of v. In N', we have two new edges ab and cd; we must check that these edges adhere to the HGT-consistent labelling. By condition 2, we have $t(u) < t(b)$ since $t(u) = t(v)$. This means $t'(a) \leq t(u) < t(b) = t'(b)$. If b was a reticulation, then it must have had to inherit its time label in N from the parent k that is not u – otherwise we get a contradiction for condition 3. In particular, b continues to inherit its time label from k in N'. So condition 3 still holds for b in N' under t'. But this means that the edge ab adheres to the HGT-consistent labelling in N' under t'. On the other hand, note first that $t(v) < t(d)$ by condition 2 since v is a reticulation. This implies that $t'(c) = t(c) < t(v) < t(d) = t'(d)$. If d was a reticulation, then it must have had to inherit its time label in N from the parent that is not v – otherwise t would not be a HGT-consistent labelling under condition 2 for the vertex v. So condition 3 still holds for d in N' under t'. Thus, t' is HGT-consistent for N'.

Observe that N' contains two fewer vertices than N. By the induction hypothesis, there exists a base tree T' of N' that can be obtained from N' by deleting all reticulation edges where the tail and head have the same time labels. We claim that the embedding of this base tree into N' uses the edges ab and cd. Indeed, if b was a tree vertex, the use of the edge ab would be immediate. If b was a reticulation, then its parent k that is not a must have the same time label, i.e., $t'(b) = t'(k)$. But this would imply that the edge kb is deleted to obtain the base tree, and that the edge ab then must be used in the embedding. An analogous argument can be used for d, to see that the edge cd is used in the embedding of this base tree into N'. Upon subdividing the edges ab and cd by vertices u and v respectively, and adding the edge uv to N', we undo the reticulate edge deletion from before to obtain N. A base tree T of N can be obtained from the base tree T' of N' by replacing the edges ab and cd by paths aub and cvd in the embedding. It follows immediately that this is a base tree of N that can be obtained by deleting all reticulation edges where the tail and head have the same time labels (since t' and t coincide at all vertices except for uv; we do not use the edge uv to obtain T). □

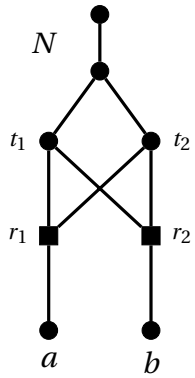


Figure 5.8: A network N on the taxa set $\{a,b\}$ containing a crown on the vertices t_1,t_2,r_1,r_2. If N admitted an HGT-consistent labelling t, then assuming without loss of generality that $t(t_1) \geq t(t_2)$, we must have that $t(t_1) = t(r_1) = t(r_2)$. But this contradicts condition 2 of the HGT-consistent labelling definition for the vertex t_1. Therefore, N does not admit an HGT-consistent labelling.

The converse of Lemma 59 does not hold. Consider a tree-based network that contains a crown (see Figure 5.8). As can be seen from the figure, it is not possible to give crowns an HGT-consistent labelling.

Lemma 60. *Each binary orchard network admits an HGT-consistent labelling t where $|t^{-1}(i)| \leq 2$ for all time labels i.*

Proof. Let N be a binary orchard network, and let $S = (x_1,y_1),\ldots,(x_m,y_m)$ be a cherry-picking sequence for N. Because N is binary, it can be reconstructed from S by starting with the one-leaf tree with leaf y_m and reattaching the pairs from S in reverse order (see Chapter 4, Definition 7). Define a labelling $t : V \to \mathbb{R}$ on the vertices V of N. Label the root vertex ρ with $t(\rho) = 0$, the leaves with times $m+1,\ldots,m+n$ arbitrarily, and each interior vertex v added when reattaching the pair (x_i,y_i) with $t(v) = m+1-i$.

We show that t is indeed an HGT-consistent labelling of N. When adding a pair to a network, the two newly introduced vertices lie below existing internal vertices. This means that vertices that are added later have a greater time labelling than internal vertices that are added before. Adding to the fact that we relabel the leaves so that they have labels of at least $m+1$ and internal vertices have labels of at most m, we have that for all edges uv, $t(u) \leq t(v)$. The relabelling of the leaves also means that two vertices in the network have the same label under t if they are a parent-child pair where the child is a reticulation. Therefore, condition 1 of the HGT-consistent labelling is satisfied. To see that t satisfies condition 2, we look at tree vertices and reticulations separately. A tree vertex u has two children, one of which is possibly added to the network at the same time as u. The other child v of u is either a leaf or an internal vertex that is added to the network after u has been added. But this would mean that $t(u) < t(v)$. On the other hand, a reticulation r has one child c. Every reticulation vertex is added to the network with one of its parents; this means that c is either a leaf or an internal vertex that is added to the network after r has been added. This implies $t(r) < t(c)$. So condition 2 is satisfied. Finally, to see that condition 3 is also satisfied, consider a reticulation r with parents u

and v. The reticulation r must have been added to the network with one of its parents, say u, so that $t(u) = t(r)$. This means that the vertex v was already in the network when r was added; due to how we have defined t, we must have that $t(v) < t(r)$. Thus t satisfies condition 3, and therefore it is an HGT-consistent labelling.

In addition, this gives a labelling t with $|t^{-1}(i)| \leq 2$ for all times i. Indeed for each $i \in \{1, \ldots, m\}$, the vertices with label i are added to the network when (x_i, y_i) is reattached, and for each such reattachment, at most two vertices are added to the network. □

Lemma 61. *Each binary network that admits an HGT-consistent labelling is orchard.*

Proof. Suppose a binary network N admits and HGT-consistent labelling t. We will prove that N is orchard by proving that each network that admits an HGT-consistent labelling and which has at least one internal vertex must contain a cherry or reticulated cherry. Moreover, after reducing such a pair, the resulting network still admits an HGT-consistent labelling. Therefore, any HGT-consistent network can be reduced to a tree with one leaf, and is thus orchard.

Let x be a non-leaf vertex with $t(x)$ maximal. If x is a reticulation, then its child l must be a leaf, and one of its parents p has label $t(p) = t(x)$, by condition 3. The vertex p cannot be a reticulation as this would contradict condition 2 of HGT-consistency. Hence, the other child l' of p must be a leaf as well, and (l, l') is a reticulated cherry in N. Reducing this reticulated cherry, we obtain a new network N', which still admits an HGT-consistent labelling $t' = t|_{V(N')}$. If x is a tree node, then either both of its children are reticulations, one of its children v is a reticulation vertex, or both its children are leaves. In the first case, the reticulation children v_1 and v_2 must have labels $t(v_1) = t(v_2) = t(x)$, since reticulations inherit their time label from the parent with the greater time label, by condition 3. But this contradicts condition 2, so this case is not possible. In the second case, the reticulation child has label $t(v) = t(x)$, and we can reduce the reticulated cherry involving x and v as in the previous case. In the third case, x has two leaf children, which must thus form a cherry. After reducing this cherry, the network still admits an HGT-consistent labelling, which can be obtained by restricting t to the remaining vertices. □

A direct consequence of these lemmas is the following new characterization of orchard networks as networks that admit an HGT-consistent labelling.

Theorem 19. *A binary network N is orchard if and only if it admits an HGT-consistent labelling.*

Since networks with HGT-consistent labellings are tree-based by Lemma 59, this gives another proof for showing that binary orchard networks are contained in the class of binary tree-based networks.

5.5.2. NON-BINARY ORCHARD NETWORKS

We extend the HGT-consistent labelling characteristics to non-binary orchard networks.

Theorem 20. *A non-binary network N is orchard if and only if some binary resolution of N admits an HGT-consistent labelling.*

Proof. The combination of Theorem 19 and Lemma 58 immediately gives the claim. □

Ideally, we can further extend this characterization by finding a direct time labelling of non-binary networks that captures orchard networks, with a meaningful biological interpretation. With Theorem 20 in mind, a natural candidate seems to be to find a binary resolution, find an HGT-consistent labelling of the resolution, and then to contract the edges of the network to obtain the original non-binary network, in the meantime altering the labelling to adhere to the monitonically increasing order (condition 1). To do so, whenever we contract an edge uv, we shall label the new vertex w with the greater time label, which would be that of v.

This results in a network N with vertex set V with the labelling $t : V \to \mathbb{R}$ such that

1. For all edges uv, $t(u) \leq t(v)$.
2. For each non-leaf vertex u, there is at least one child v of u such that $t(u) < t(v)$.
3. For each reticulation r with parents p_i for $i \in [k]$, $t(r) = t(p_j)$ for exactly one parent p_j.

Unfortunately, while it is the case that every non-binary orchard network admits such a labelling, networks that are not orchard can also have such labellings. Figure 5.9 gives a non-orchard network with such a labelling. The network here contains a crown on the tree vertices with time stamps 1,2 and the two reticulations of the network. Because the two other parents of the reticulations are there, i.e., the two tree vertices labelled 3 and 4, we are able to circumvent the issue of having to label the reticulation vertices with time stamp 2.

The last condition can be changed to something that would imply the existence of a base tree, for which all edges that are not in the base tree are horizontal edges. This requires changing condition 3 to the following.

3. For each reticulation r with parents p_i for $i \in [k]$, $t(r) = t(p_i)$ for all but exactly one parent p_j.

Unfortunately, this change still does not fully capture the class of orchard networks. Figure 5.9 gives an example of a non-binary orchard network that does not admit a labelling under this rule change.

5.6. DISCUSSION

IN this chapter we have provided a unifying structural characterization for tree-based networks and orchard networks using cherry covers. We have shown that a binary network is tree-based if and only if it can be decomposed into cherry shapes and reticulated cherry shapes. A binary network is orchard if such a decomposition exists that also satisfies a certain acyclicity condition. Moreover, we have generalized these characterizations to non-binary networks by considering bulged versions of the networks and using covers rather than decompositions. Prior to having this characterization, orchard

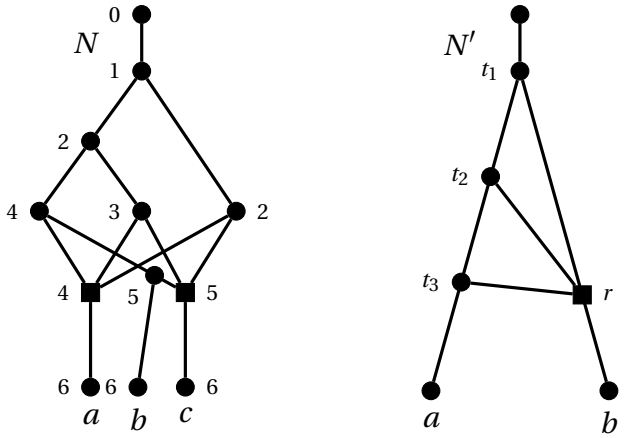


Figure 5.9: (Left) A non-binary non-orchard network N on $\{a, b, c\}$ that admits a time-labelling on the vertices that adheres to the 'reticulations have the same labels as exactly one of its parents' rule. (Right) A non-binary orchard network N' on $\{a, b\}$ that does not admit a time-labelling on the vertices that adheres to the 'reticulations have the same labels as all but one of its parents' rule. Indeed, under this rule, exactly two of t_1, t_2, t_3 must have the same time-label. But in all possible cases therein, such a labelling does not satisfy either condition 1 or 2. In particular, these examples show that neither of these two labelling rules are enough to characterize the class of non-binary orchard networks.

networks were characterized only by the sequences that reduced them. Therefore we have provided the first structural (non-recursive) characterization for orchard networks. As a result, we have shown that the class of non-binary orchard networks is contained in the class of non-binary tree-based networks. We have also given a biological interpretation of orchard networks as those that can be obtained by adding horizontal edges to its base tree.

Structural characterizations for many network classes have generally focused more on 'forbidden structures' rather than on decompositions. Tree-based networks cannot contain a maximum zig-zag path that starts and ends at a reticulation (*W-fences*) [5, 14]; tree-child networks cannot contain adjacent reticulation vertices nor tree vertices with only reticulation children. While structures such as crowns and W-shapes cannot be contained in orchard networks, it remains open whether orchard networks can be characterized by a list of forbidden substructures.

In the other direction, it may be of interest to extend our cherry cover results to characterize other network classes that are contained in the class of tree-based networks. Since (the bulged versions of) these networks have a cherry cover, this may be possible by imposing additional conditions on the cherry cover. Finding such characterizations for all known network classes, such as tree-child, reticulation-visible, and stack-free, will truly bring to light a *unifying* structural characterization of known phylogenetic network classes.

Outside of characterizing network classes, cherry covers can be used to prove other results within phylogenetics. One particular case in which this could have been useful is in proving certain results for orchard networks and tree-based networks in the paper [8]. In that paper, it was shown that leaves may be added to, and some leaves may be re-

moved from orchard and tree-based networks to obtain a network that was still orchard or tree-based (in particular, Lemmas 10, 11 and 13). Employing cherry covers to prove these results is more concise, since adjusting the cherry cover of networks after such actions is easier than trying to alter, say, the sequence for the network (for orchards).

Another area in which cherry covers may be useful is in solving enumeration problems, which is formulated as follows. Given parameters n and r, find the number of distinct networks on n leaves with hybridization number r. When considering the class of tree-based networks, there exist cherry covers for such networks that contain $n-1$ cherry shapes and r reticulated cherry shapes by Corollary 8—can we somehow count all possible arrangements of these shapes to enumerate the space of both network classes? Perhaps, for non-binary networks, this line of attack will be too complicated due to shapes being able to cover the same edges. However, for binary networks this may be viable, as each edge of the network is covered exactly once in a cherry decomposition by Lemma 49.

On the algorithmic front, one may find a cherry cover for a tree-based network and an acyclic cherry cover for an orchard network in polynomial-time. For orchard networks, we may find reducible pairs, cover the edges involved using the steps outlined in the proof of Theorem 18, reduce the shape, and continue until an acyclic cherry cover is obtained. Since we may pick reducible pairs from orchard networks in any order (Theorem 9), this bottom-up approach provides a polynomial-time algorithm for finding an acyclic cherry cover of an orchard network. For tree-based networks, we first find a base tree in polynomial-time with the matching approach used in the proof of Theorem 3.4 in [6]. Then, one may follow the steps outlined in the proof of Theorem 17 of this chapter to convert the cherry cover of this base tree to a cherry cover of the network in polynomial-time. Without the base tree however, it is not clear if there is a systematic way of obtaining a cherry cover. Indeed, it is not enough to naively cover the edges in any fashion (e.g., bottom-up), as shown in Figure 5.10. We wonder if it would be possible to directly obtain a cherry cover of a tree-based network without first having to find a base tree; and if this is the case, can it be done faster than the algorithm presented in [6]?

The HGT-consistent labelling gives a biologically meaningful characterization of binary orchard networks, and outlines the motivation for studying them apart from computational reasons. In particular, we have shown that binary orchards networks are those that can be obtained from trees by adding horizontal edges. More still needs to be done in this area, especially for characterizing non-binary orchard networks. We have tried two labellings to characterize non-binary orchard networks: one where all arcs added to the base tree are horizontal, and one where only one arc, per reticulation, added to the base tree is horizontal. We saw in Section 5.5.2 that these two labellings do not fully capture the network class. Nevertheless, while a direct characterization may not have been found yet, we may characterize non-binary orchards using time labels by first finding a binary resolution of the network (which exists by Lemma 58), and then by finding an HGT-consistent labelling of the resolution.

REFERENCES

[1] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A unifying characterization of tree-based networks and orchard networks using cherry covers,* Advances

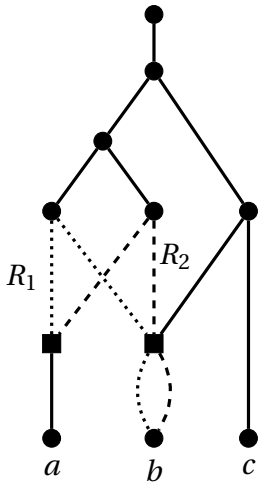


Figure 5.10: The bulged version of the tree-based network from Figure 5.3, in which we cover some of the edges with arbitrary reticulated cherry shapes R_1 and R_2. Since the edge incident to the leaf a can no longer be covered by any reticulated cherry shape, there exists no cherry cover that contains both R_1 and R_2.

in Applied Mathematics **129**, 102222 (2021).

[2] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications* (Cambridge University Press, 2010).

[3] W. F. Martin, *Early evolution without a tree of life,* Biology direct **6**, 36 (2011).

[4] A. R. Francis and M. Steel, *Which phylogenetic networks are merely trees with additional arcs?* Systematic biology **64**, 768 (2015).

[5] L. Zhang, *On tree-based phylogenetic networks,* Journal of Computational Biology **23**, 553 (2016).

[6] L. Jetten and L. van Iersel, *Nonbinary tree-based phylogenetic networks,* IEEE/ACM transactions on computational biology and bioinformatics **15**, 205 (2018).

[7] A. Francis, C. Semple, and M. Steel, *New characterisations of tree-based networks and proximity measures,* Advances in Applied Mathematics **93**, 93 (2018).

[8] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, and C. Semple, *Rooting for phylogenetic networks,* arXiv preprint arXiv:1906.07430 (2019).

[9] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[10] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[11] B. M. Moret, L. Nakhleh, T. Warnow, C. R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme, *Phylogenetic networks: modeling, reconstructibility, and accuracy,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **1**, 13 (2004).

[12] M. Baroni, C. Semple, and M. Steel, *Hybrids in real time,* Systematic biology **55**, 46 (2006).

[13] C. McDiarmid, C. Semple, and D. Welsh, *Counting phylogenetic networks,* Annals of Combinatorics **19**, 205 (2015).

[14] M. Hayamizu, *A structure theorem for tree-based phylogenetic networks,* arXiv preprint arXiv:1811.05849 (2018).

6

ENCODING BY NUMBER OF PATHS TO A LEAF

In Chapters 3 and 4, we saw network encodings from displayed subnetworks as well as from cherry-picking sequences. In this chapter, we show a third type of encoding called μ-representations. The μ-vector of each vertex has the number of paths from it to the ith leaf for its ith element. The set of all μ-vectors for each vertex in the network is called the μ-representation (and it is equivalent to a notion called 'ancestral profiles'). It has been shown that non-binary tree-child networks are encoded by their μ-representation. It has also been shown that binary stack-free orchard networks are strongly encoded by their μ-representation. Here, we introduce the class of tree-clone-free networks, which are networks whose tree vertices all have a unique μ-vector. We show preliminary results for tree-clone-free networks regarding the structure of such networks, and conjecture that binary tree-clone-free networks are encoded by their μ-representations. Recently, it was claimed that stack-free orchard networks (not necessarily binary) were encoded by their μ-representations within the class of all stack-free networks. We also provide a counter-example to this claim, and layout the intuition for why this is.

6.1. INTRODUCTION

THE number of evolutionary paths from a non-extant species (internal vertex of a network) to an extant species (leaf) denotes the different possible ways in which genetic material has passed. For trees, this number is binary; either the vertex is or is not an ancestor of the leaf. For networks however, the existence of reticulation vertices allows for more than one path to the same taxon.

For each vertex in the network, we denote the number of paths from it to each leaf in the network as a vector called the μ-vector. The multiset of μ-vectors of a network is called the μ-representation of the network. For trees, the μ-representation of the network corresponds to its set of clusters. We know that a tree is reconstructible from their

set of clusters [1, Theorem 9.2], and therefore from their μ-representation. When then, is a network reconstructible from its μ-representation? Prior results have shown that tree-child networks and time-consistent tree-sibling networks are reconstructible from their μ-representations [2, 3]. It is also known that binary stack-free orchard networks are strongly encoded by their μ-representations [4].

The initial motivation for considering μ-representations was to construct a distance metric on some class of networks [2]. Comparing networks is necessary for assessing the accuracy of network construction methods [5]; starting with a network N, obtain some form of building block from it. This could be in the form of a sequence alignment, a set of reticulate-edge-deleted subnetworks (what we saw in Chapter 3), or a distance matrix (we will see this in Chapter 7). Construct a network N' from the building block(s), and compute the distance between N and N'. Ideally N' is isomorphic to N, which is the case if and only if the distance between them is 0 for a distance metric.

Note that the existence of an encoding for a class of networks immediately implies the existence of a distance metric for the class. Let $\mathscr{C}$ be a class of networks that are encoded by some type of building block S, where the set of building blocks for a network $N \in \mathscr{C}$ is denoted by $S(N)$. Define a distance metric between two networks N, N' in $\mathscr{C}$ by taking the size of the symmetric difference between the two building blocks $|S(N) \triangle S(N')|$. Since no two distinct networks have the same building blocks, and because the triangle inequality holds for the size of symmetric differences, it follows that this yields a distance metric within the class. This means that showing that a class of networks is encoded by the μ-representations immediately provides a distance metric for the class.

An advantage of defining distance metrics based on encodings is that they are much faster to compute compared to some other distance metrics. Generally, the building blocks for a network in a certain class can be computed in polynomial-time. Finding the symmetric difference between two multisets can also be done quickly. On the other hand, consider rearrangement moves, which are graph operations that remove and reattach parts of the network (for an overview on rearrangement moves, see, for example [6, 7]). These are popular methods in traversing the space of phylogenetic networks to obtain optimal solutions in a Bayesian, a maximum likelihood, or a maximum parsimony framework. The distance between two networks can be obtained by taking the number of moves required to transform one into the other; unfortunately, computing this distance is known to be NP-hard for most known types of moves [8].

Recently, it was claimed that semi-binary stack-free orchard networks are strongly encoded by their μ-representations [9]. In this chapter, we give a counter-example for this statement, and we conjecture that the non-uniqueness is caused perhaps by tree clones, which are two tree vertices with the same μ-vector. Interestingly, orchard networks do not contain tree clones; due to this, we consider the class of tree-clone-free networks, and we show preliminary results on the class. Furthermore, we show that the automorphism group on tree-clone-free networks contains only the trivial element. This automorphism result could have applications in Bayesian methods, where current methods do not take into account the internal symmetries of a phylogenetic network. This can lead to undersampling of symmetric networks within Bayesian methods in PhyloNet [10].

Structure of the chapter: In the next section we formally define the notion of a μ-representation, and we list all known reconstructibility results for μ-representations. In Section 6.3, we give a counter-example to Theorem 3.1 of Bai et al. [9]. In Section 6.4, we show that the clones in a tree-clone-free network must either be contained in the same reticulation equivalence class, or they are parent-child pairs for a reticulation and a tree vertex (Lemma 64). Furthermore, we show that the semi-binary tree-clone-free networks have only one automorphism (Theorem 25). In Section 6.5, we layout the roadmap for future research in this area.

6.2. PRELIMINARIES

In this chapter we consider only semi-binary rooted networks. Furthermore, we consider the version of networks in which roots have indegree-0 and outdegree-2 (essentially contract the edge between the root and the first tree vertex). Recall that in a semi-binary network, every tree vertex is of total degree 3 and every reticulation vertex is of total degree ≥ 3. The *stack identification* of a network N, denoted by $id(N)$, is the semi-binary network obtained by contracting every reticulation edge with a reticulation vertex tail [9]. Observe that a network N is stack-free if and only if $N \cong id(N)$. We say that two reticulations u, v are *equivalent* if there exists a reticulation r such that there is an rr-path from u to r and an rr-path from v to r. Recall that an rr-path is a path consisting only of reticulation vertices (see Section 4.2). This forms an equivalence class on the reticulation vertices of networks.

Let N be a network on the taxa set X, and let V denote the vertex set of N. Since X is chosen arbitrarily, we may assume without loss of generality that $X = [n] = \{1, 2, \ldots, n\}$ for some positive integer n. For each vertex $v \in V$, let $m_i(v)$ denote the number of paths from v to the leaf i. As every vertex has a path to itself, namely the empty path, we have that $m_i(i) = 1$ for all leaves $i \in [n]$. The *μ-vector* of $v \in N$ is the vector

$$\mu_N(v) = (m_1(v), m_2(v), \ldots, m_n(v)).$$

The *μ-representation* of N is the multiset of every μ-vector in N

$$M_N = \{\mu_N(v) : v \in V\}.$$

Let $v_1, v_2, \ldots, v_\ell$ be an arbitrarily fixed labelling of the vertices V in N. The *ancestral tuple* of $i \in [n]$ is the vector

$$\sigma(i) = (m_i(v_1), m_i(v_2), \ldots, m_i(v_\ell)).$$

The *ancestral profile* of N is the set of ordered pairs

$$\Sigma_N = \{(i, \sigma(i)) : i \in [n]\}.$$

Note that in the original definition, only the non-leaf vertices were included in the ancestral tuple, and therefore in the ancestral profile [4]. The addition of the leaf vertices does not make use of any additional information, however, it does make the proof of the following observation more simple.

Observation 13. *Let N be a network on X with vertex set V. Then the μ-representation M_N can be obtained from the ancestral profile Σ_N and vice versa.*

Proof. Consider an $X \times V$ matrix where the (i, v)-th element is the number of paths from the vertex v to the leaf i. The multiset of the column vectors is precisely the μ-representation of N. The set of ordered pairs with a leaf as first element and the corresponding row vector as the second element is precisely the ancestral profile of N (with the ordering on V determined by how V is ordered in the matrix columns). One can therefore derive the μ-representation of N from its ancestral profile. □

We say vertices u, v in a network are *clones* if $\mu(u) = \mu(v)$. If clones u, v are both tree vertices, we say that u, v are *tree clones*. We say that a network is *tree-clone-free* if it contains no tree clones. We call a stack-free tree-clone-free network a *STCF network*.

6.2.1. KNOWN RECONSTRUCTIBILITY RESULTS FOR ANCESTRAL PROFILES

For definitions of the following network classes, we refer the reader to Section 2.1.4. Some network classes are known to be reconstructible from their μ-representations.

Theorem 21 (Theorem 1 of [2])**.** *Let N, N' be semi-binary tree-child networks. Then $N \cong N'$ if and only if $\mu(N) = \mu(N')$.*

This was also shown to be true for time-consistent tree-sibling networks.

Theorem 22 (Proposition 8 of [3])**.** *Let N, N' be semi-binary time-consistent tree-sibling networks. Then $N \cong N'$ if and only if $\mu(N) = \mu(N')$.*

Recently, the result was also shown for binary orchard networks by [4, 9].

Theorem 23. *Let N be a binary stack-free orchard network, and let N' be a binary stack-free network. Then $N \cong N'$ if and only if $\mu(N) = \mu(N')$.*

Note that this third theorem differs from the first two theorems, as the first two results give encoding results where both networks must be within the same restricted classes (defined as *weak encoding* in Section 1.3.2). The third theorem gives a strong encoding result within a larger class.

6.3. COUNTER-EXAMPLE FOR BAI ET AL.'S THEOREM

THE following was claimed in a recent paper by Bai et al.

Conjecture 5 (Theorem 3.1 of [9])**.** *Let N be a semi-binary stack-free orchard network on X, and let N' be a semi-binary stack-free network. Then $N \cong N'$ if and only if $\mu(N) = \mu(N')$.*

Figure 6.1 shows that this conjecture cannot be true. The network N is a binary stack-free orchard network; network N' is a semi-binary stack-free network that is not orchard. Both networks have the same μ-representations, yet they are not isomorphic. The issue arises from the fact that we cannot discern between reticulated cherries with differing

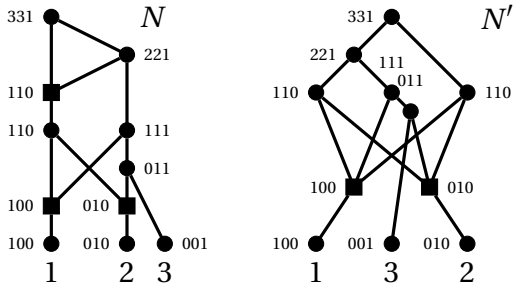


Figure 6.1: Two stack-free networks N and N' on the same leaf set $\{1,2,3\}$. The network N is a binary orchard network with the cherry-picking sequence $(2,3)(1,2)(1,3)(2,3)(2,3)$. The network N' is semi-binary and is not orchard. The μ-vector labels each vertex (the leaves therefore have two labels: the leaf label as well as the μ-vector). Both networks have the same μ-representation $\{(331,1),(221,1),(111,1),(110,2),(011,1),(100,2),(010,2),(001,1)\}$, where each element in the set has a μ-vector as the first coordinate and its multiplicity as the second coordinate. The networks N and N' act as a counter-example to Conjecture 5.

indegrees on the reticulation parent. In Figure 6.1, both networks contain the reticulated cherry $(2,3)$, where in N, the indegree of the parent of 2 is 2, and that in N' is 3. Therefore, we obtain the following theorem.

Theorem 24. *Semi-binary stack-free orchard networks are not encoded by their μ-representations.*

In this particular example, another key difference between N and N' is that N contains 3 reticulation vertices whereas N' contains 2 reticulation vertices. The μ-vector of one of the reticulations in N, namely 110, appears as a μ-vector of a tree vertex in N'. Because 110 is also a μ-vector of a tree vertex in N, this leads to N' containing a tree clone for the μ-vector 110. The existence of this tree clone essentially 'allows' for the root vertex to choose between the two tree clones as its children, leading to the creation of the counter-example. Interestingly, it is known that orchard networks contain no tree clones (Lemma 4.4 of [9]). In an effort to extend Theorem 23 to a more general class of networks, it is natural then, to consider the class of tree-clone-free networks.

Further observation of N' in Figure 6.1 shows that the tree clones exist as a result of a *crown* in the network. However, tree clones are not necessarily the product only of crowns. Figure 6.2 gives two non-isomorphic networks with the same μ-representations, with no crown. Therefore the symmetric difference between the class of crown-free networks and the class of tree-clone-free networks is non-empty. Furthermore, this shows that crown-free networks are not encoded by their μ-representations.

The reason for delving into these two classes is because we know that orchard networks are necessarily crown-free and that they are also tree-clone-free (Lemma 4.4 of [9]). Furthermore, the above example exemplifies the point that simply considering crown-free networks is not enough. In the next section we look at a few preliminary results on tree-clone-free networks.

6.4. STRUCTURE OF TREE-CLONE-FREE NETWORKS

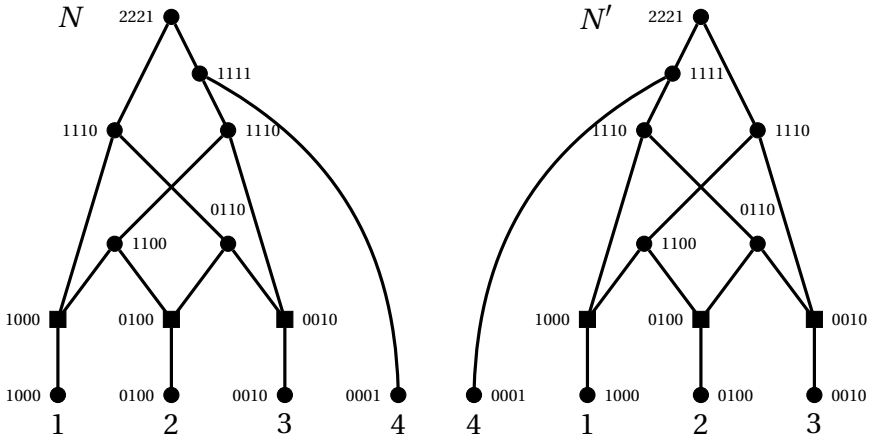


Figure 6.2: Two non-isomorphic binary stack-free non-orchard networks with the same μ-representation. Both networks contain a tree clone on the μ-vector 1110, however, they do not contain crowns.

We recall the following key lemma which states that the μ-vector of a vertex is the sum of the μ-vectors of all its children.

6

Lemma 62 (Lemma 4b of [2])**.** *Let N be a network. Let v be a vertex of N with children $c_1, \ldots, c_k$. Then $\mu(v) = \sum_{i \in [k]} \mu(c_i)$.*

We say that two reticulations r_1, r_2 are in the same *reticulation equivalence class* if there exists a reticulation r such that there exists an rr-path from r_1 to r, and an rr-path from r_2 to r. It is easy to check that this forms an equivalence class on the reticulations of N.

Observation 14. *Let N be a network. Let r be a reticulation of N. Then there exists a tree vertex or a leaf t that can be reached from r by a path whose internal vertices consist only of reticulations. For any such t, it holds that $\mu(r) = \mu(t)$.*

Proof. Consider the lowest reticulation v contained in the same reticulation equivalence class as r. As v is lowest, the child t of v must be a tree vertex or a leaf. Furthermore by Lemma 62, $\mu(v) = \mu(t)$. Since r and v are contained in the same reticulation equivalence class, and as v is the lowest such reticulation, there must exist an rr-path from r to v. Furthermore, the μ vector of all vertices on this path is the same by Lemma 62; in particular, $\mu(r) = \mu(v)$. Therefore there exists a path from r to t consisting only of reticulation internal vertices, such that $\mu(r) = \mu(t)$. □

Lemma 63. *Let N be a tree-clone-free network, and let v_i, v_j be reticulations in N. Then $\mu(v_i) = \mu(v_j)$ if and only if v_i and v_j are contained in the same reticulation equivalence class.*

Proof. Suppose first that v_i and v_j are contained in the same reticulation equivalence class. Then there exists a reticulation v such that v_i and v_j have disjoint paths to v consisting only of reticulations. If v_i is above v_j, then v is the vertex v_j (and also vice

versa). Since the μ-vector of a reticulation is equal to the μ-vector of its only child by Lemma 62, we must have that $\mu(v_i) = \mu(v) = \mu(v_j)$.

Now suppose that $\mu(v_i) = \mu(v_j)$. Suppose for a contradiction that v_i and v_j are contained in different reticulation equivalence classes. Since reticulations in the same equivalence classes have the same μ-vectors (by what we have just shown in the first part of this proof), we may, without loss of generality, assume that v_i and v_j are the lowest reticulations in their respective classes. This means that the child c_i of v_i and the child c_j of v_j are either tree vertices or leaves. By Lemma 62, we must have $\mu(c_i) = \mu(v_i) = \mu(v_j) = \mu(c_j)$. As N is a tree-clone-free network by assumption, c_i and c_j cannot both be tree vertices. So one of these must be a leaf, say c_i. The vertex c_j cannot also be a leaf as network leaves are uniquely labelled. So vertex c_j must be a tree vertex; however, this would mean that c_j either has a path to a leaf that is not c_i, or c_j has more than one path to c_i. In both cases, we would have that $\mu(c_i) \neq \mu(c_j)$, which is a contradiction. Therefore v_i and v_j must be contained in the same reticulation equivalence class. □

The following is a generalization of Lemma 4.4 in [9] from orchard networks to tree-clone-free networks.

Lemma 64. *Let N be a tree-clone-free network. Any two non-leaf vertices v_i, v_j in N are clones if and only if one of the following holds:*

1. *v_i and v_j belong to the same reticulation equivalence class; or*
2. *exactly one of v_i and v_j is a reticulation, say v_i, and there is a path from v_i to v_j where the internal vertices are all reticulations.*

Proof. If one of the two conditions holds, then the vertices v_i and v_j are clones by Lemmas 63 and 62. Therefore suppose that v_i and v_j are clones. Since N is tree-clone-free, v_i and v_j cannot both be tree vertices. If v_i and v_j are both reticulations, then by Lemma 63, they must be contained in the same reticulation equivalence class. So it remains to show that the claim holds when one is a tree vertex and the other a reticulation.

Suppose without loss of generality that v_i is a tree vertex and that v_j is a reticulation. By Observation 14, there exists a tree vertex t that can be reached from v_j by a path whose internal vertices consist only of reticulations, such that $\mu(v_j) = \mu(t)$. Because N is tree-clone-free, we must have that $t = v_i$. This proves the claim. □

From Lemma 64, the next corollary follows immediately for STCF networks.

Corollary 10. *If an STCF network has μ-representation M, then the multiplicity of every μ-vector is at most 2 in M.*

6.4.1. HYBRIDIZATION NUMBER OF STCF NETWORKS FROM μ-REPRESENTATIONS

In this section, we show that the reticulation number and the hybridization number of a semi-binary STCF network can be inferred from its μ-representation. Recall that the hybridization number of a network is the total indegree of all reticulation vertices minus the total number of reticulation vertices. Following this, we show that semi-binary networks with at most one reticulation vertex are encoded by their μ-representations, within the class of semi-binary STCF networks.

Lemma 65. *Let N be a semi-binary STCF network, and let M be its μ-representation. Let v be a vector in M with multiplicity 2. Let r, c be vertices of N such that $\mu(r) = \mu(c) = v$. Then exactly one of r, c, say r, is a reticulation vertex and c is a non-reticulation vertex such that rc is an edge of N. Conversely, for every reticulation r in N, the multiplicity of the vector $\mu(r)$ in M is 2.*

Proof. By Lemma 64, either r and c belong in the same reticulation equivalence class, or r is a reticulation in the same equivalence class as the parent of c and c is a non-reticulation vertex. As N is stack-free, every reticulation equivalence class is of size 1. This means in particular that the first case is not possible. For the second case, this implies that r is the parent of c, which is what we wanted.

To prove the second statement, the child c of r is a non-reticulation since N is stack-free, and we also have that $\mu(r) = \mu(c)$ by Observation 14. So the multiplicity of $\mu(r)$ is at least 2 in M. By Corollary 10, every μ-vector in M is of multiplicity at most 2. Therefore, the multiplicity of $\mu(r)$ is exactly 2. □

Corollary 11. *Let N be a semi-binary STCF network, and let M be its μ-representation. Let u denote the number of vectors in M of multiplicity 2. Then N contains u reticulation vertices.*

Lemma 66. *Let N be a semi-binary STCF network, and let M be its μ-representation. Let n denote the number of leaves in N, and let f denote the number of vectors with multiplicity 1 in M. Then the hybridization number of N is*

$$f - 2n + 2.$$

Proof. By Corollary 10, every vector in M is of multiplicity at most 2. By Lemma 65, every vector in M of multiplicity 2 corresponds to a parent child pair, where the parent is a reticulation and the child is a non-reticulation vertex. Every vector in M of multiplicity 1 therefore corresponds either to a tree vertex or a leaf. Together, this means that the number of vectors with multiplicity 1 in M, denoted by f, is the number of tree vertices and leaves in N. By Lemma 51, we know that a semi-binary network with n leaves and t tree vertices has hybridization number $t - n + 2$. Since $f = t + n$, this means that the hybridization number of N is $f - 2n + 2$. □

At the start of Section 6.3, we commented on why Conjecture 5 cannot hold by giving the counter-example of Figure 6.1. The existence of the counter-example stemmed from the fact that one cannot discern between reticulated cherries with reticulations of differing indegrees. One can indeed identify a reticulated cherry from the μ-representations of a stack-free network by Lemma A.1 of [9]; it is just not possible to comment on the indegree of the reticulation involved. This is the step that is currently lacking in the proof of an encoding result for semi-binary stack-free orchard networks from their μ-representations. Lemma 4.3 of [9] gives a way of reducing reticulated cherries and a way of adjusting the μ-representation accordingly. If one can infer the indegree of each reticulation vertex of every reticulated cherry from the μ-representations, then we would have the right ingredients to prove Conjecture 5. For now, we can do this as long as the network contains exactly one reticulation vertex, i.e., in the case that the indegree

of the reticulation vertex is exactly the number of vectors of multiplicity 2 in the μ-representation. In fact, we show the following in the larger class of semi-binary STCF networks.

Lemma 67. *Let N be a semi-binary network with at most one reticulation vertex, and let N' be a semi-binary STCF network. Then $N \cong N'$ if and only if $\mu(N) \cong \mu(N')$.*

Proof. Note first that N is stack-free and orchard. By Lemma 4.4 of [9], this means that N is tree-clone-free, so we may use the results from above. One direction, that if $N \cong N'$ then $\mu(N) \cong \mu(N')$ is trivial, so suppose that $\mu(N) \cong \mu(N')$.

We prove the claim by induction on the number of vertices $|N|$ in N. If $|N| = 1$ or $|N| = 3$, we either have a network on a single leaf or a network with a cherry, respectively. It is not possible to have a network on 2 vertices. These cases follow trivially, and so we assume that $|N| \geq 4$.

By definition, orchard networks contain either a cherry or a reticulated cherry. Identifying a cherry from $\mu(N)$ is possible by Corollary 3.2 of [4]; identifying a reticulated cherry from $\mu(N)$ is possible by Lemma A.1 of [9]. If there were to be a reticulated cherry, we know from Corollary 11 that the network realising $\mu(N)$ contains at most one reticulation vertex; we can also infer the hybridization number of the network realising $\mu(N)$ by Lemma 66. Since there is at most one reticulation vertex, its indegree, should it exist, must be the hybridization number itself. Suppose that N contains a reducible pair (x, y). Then, we may use Lemma 4.3 of [9] to reduce (x, y) from N and adjust the μ-representation accordingly. Call the reduced network N_1 and the adjusted μ-representation $\mu(N_1)$. The network N_1 is semi-binary, contains fewer vertices than that of N, and it still contains at most one reticulation vertex. By the induction hypothesis, N_1 is the unique network within the class of semi-binary STCF networks, up to isomorphism, whose μ-representation is $\mu(N_1)$.

We now look at N'; we are in the case where the μ-representation of N' is $\mu(N)$. The reducible pair (x, y) of N is also a reducible pair of N' by Corollary 3.2 of [4] and Lemma A.1 of [9][1]. Since N' is STCF, it also follows from Corollary 11 and Lemma 66 that N' must contain at most one reticulation vertex, and its indegree can be inferred. But this means that when reducing (x, y) from N' using Lemma 4.3 of [9], we get a network N_1' with the μ-representation $\mu(N_1)$. By the induction hypothesis, we must have that $N_1 \cong N_1'$. Adding the pair (x, y) to N_1 can only be done in one way, since we require the resulting network to be semi-binary stack-free (Lemma 20). We must then have that $N \cong N'$. □

6.4.2. AUTOMORPHISM GROUP OF NETWORKS

Let N be a network. An automorphism f on N is a bijective mapping from N to N, such that uv is an edge of N if and only if $f(u)f(v)$ is an edge of N, and leaves are mapped to leaves of the same label. An example of a non-trivial automorphism is given in Figure 6.3.

Observation 15. *Let N be a network and let f be an automorphism on N. Then f maps every vertex to a vertex of the same type (root, tree vertex, reticulation, or leaf) and we have $\mu(v) = \mu(f(v))$ for every vertex v in N.*

[1] This is where we require N' to be stack-free.

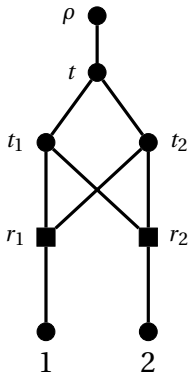


Figure 6.3: A network on two taxa $\{1,2\}$ with an automorphism group of size 2. Apart from the trivial automorphism, we obtain a non-trivial automorphism by mapping t_1 and t_2 to one another, and mapping the other elements to themselves.

Proof. Since f is an automorphism, we have that uv is an edge of N if and only if $f(u)f(v)$ is an edge of N. For the indegrees and outdegrees of vertices to coincide, we must have that vertices of a certain type are mapped to vertices of the same type. To prove the second statement, observe that distinct paths are mapped to distinct paths under f. Therefore, the number of paths to a leaf is preserved under f. □

Ancestral profiles are not only useful in showing encoding results for networks; they can be used to show results on automorphisms.

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

Proof. Let f be an automorphism of N. We shall prove that f is the identity map. As leaves are uniquely labelled, leaves are uniquely mapped to leaves of the same labels under f. By Observation 15, tree vertices must be mapped to tree vertices of the same μ-vector. As N contains no tree clones, tree vertices are mapped to themselves under f. Therefore it remains to show that reticulations must also be mapped to themselves under f.

Suppose for a contradiction that this was not the case, i.e., that a reticulation is mapped to another reticulation under f. Let r_1 and r_2 denote such two distinct reticulations, where $f(r_1) = r_2$. In particular, choose r_1 to be a highest such reticulation. Because f is an automorphism, we have that uv is an edge of N if and only if $f(u)f(v)$ is an edge of N. This means that the parents of r_1 must also be mapped to the parents of r_2. Since we chose r_1 to be the highest such reticulation, the parents of r_1 must both be tree vertices. If a parent of r_1 was a reticulation, then we have $f(r_3) = r_3$ by choice of r_1, at which point we would have that $r_1 = r_2$. Consequently, the parents of r_2 must also both be tree vertices by Observation 15. But tree vertices are mapped to themselves, as N contains no tree clones. It follows then that r_1 and r_2 have the same parents. As they have at least two parents, there exist at least two tree vertices t_1, t_2 where $\mu(t_1) = \mu(r_1) + \mu(r_2) = \mu(t_2)$ by Lemma 62. This means that t_1 and t_2 are tree clones. But this contradicts the fact

that N is tree-clone-free. Thus reticulations must be mapped to themselves under f, and therefore we conclude that f is the identity map. □

6.5. DISCUSSION

IN this chapter, we showed that binary stack-free orchard networks are not reconstructible from their μ-representations within the class of semi-binary stack-free networks. Noting that the reason for this, in the particular example, stemmed from the existence of tree clones within the network, we turned our attention to tree-clone-free networks. We showed that within tree-clone-free networks, two vertices are clones if and only if they are a parent-child pair where the parent is a reticulation and the child is a tree vertex, or if they are reticulations in the same equivalence class. We also showed that tree-clone-free networks only have one automorphism.

There are a multitude of ways in which reconstructibility results from μ-representations can be explored. A natural step is to try to show the reconstructibility of semi-binary tree-clone-free networks or to show the reconstructibility of semi-binary stack-free orchard networks from their μ-representations. This can of course be investigated for reconstructibility within their respective classes (weak encodings), or within a larger space of networks (strong encodings). The networks in Figure 6.1 show that for orchards, this can not be done within the class of semi-binary stack-free networks.

The difficulty in showing reconstructibility results for semi-binary tree-clone-free networks is that the traditional approach of reducing either a cherry or reticulated cherry and arguing by induction no longer works. There is no guarantee that tree-clone-free networks contain either a cherry or a reticulated cherry. Indeed, upon finding a lowest tree vertex, it is easy to see that it could be contained in a double-reticulated cherry (see Section 4.8). The problem with double-reticulated cherries, is that upon deleting a reticulation edge from such structures, there may be more than one possibility for reattaching the tail of the edge. Perhaps this can be remedied for tree-clone-free networks by looking at the μ-vectors. However, the network obtained by deleting this reticulation edge does not need to be tree-clone-free. This is problematic in an inductive argument, and thus a new approach is needed (or one that carefully chooses an edge, so that the resulting network remains tree-clone-free).

Showing that semi-binary stack-free orchard networks are reconstructible requires a result on distinguishing between reticulated cherries where the reticulation vertices have different indegree. Existing results show that reticulated cherries for stack-free networks can be identified from the μ-representations of networks (Lemma 3.3 of [4], with fixes from [9]). Furthermore we know how to obtain the μ-representation of a network upon reducing a cherry or a reticulated cherry (in all reticulation indegree cases) (Lemma 4.3 of [9]). Finally, because we know that a network obtained by reducing a cherry or a reticulated cherry from an orchard network is orchard (see Chapter 4), the missing piece for an inductive argument is the ability to distinguish between reticulated cherries with reticulations of distinct indegree. Perhaps the following conjecture may be the required first step in proving this reconstructibility result.

Conjecture 6. *Let N and N' be semi-binary stack-free orchard networks with the same μ-*

representation. Let r, r' be reticulation vertices in N, N', respectively, with the same μ-vectors. Then the indegree of r and the indegree of r' are the same.

REFERENCES

[1] A. Dress, K. T. Huber, J. Koolen, V. Moulton, and A. Spillner, *Basic phylogenetic combinatorics* (Cambridge University Press, 2012).

[2] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[3] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, *A distance metric for a class of tree-sibling phylogenetic networks,* Bioinformatics **24**, 1481 (2008).

[4] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[5] L. Nakhleh, J. Sun, T. Warnow, C. R. Linder, B. M. Moret, and A. Tholse, *Towards the development of computational tools for evaluating phylogenetic network reconstruction methods,* in *Biocomputing 2003* (World Scientific, 2003) pp. 315–326.

[6] J. Klawitter, *Spaces of phylogenetic networks,* Ph.D. thesis, University of Auckland (2020).

[7] R. Janssen, *Rearranging Phylogenetic Networks,* Ph.D. thesis, Delft University of Technology (2021).

[8] M. Bordewich and C. Semple, *On the computational complexity of the rooted subtree prune and regraft distance,* Annals of combinatorics **8**, 409 (2005).

[9] A. Bai, P. L. Erdős, C. Semple, and M. Steel, *Defining phylogenetic networks using ancestral profiles,* Mathematical Biosciences **332**, 108537 (2021).

[10] D. Wen, Y. Yu, and L. Nakhleh, *Bayesian inference of reticulate phylogenies under the multispecies network coalescent,* PLoS genetics **12**, e1006006 (2016).

7

ENCODINGS OF LEVEL-k NETWORKS FROM SHORTEST AND LONGEST DISTANCES

Recently it was shown that a certain class of unrooted phylogenetic networks, called level-2 networks, are not encoded by their induced shortest distance matrices. In this chapter, we show that they can be reconstructed from their induced shortest and longest distance matrices. That is, if two level-2 networks without branch lengths induce the same shortest and longest distance matrices, then they must be isomorphic. We further show that level-2 networks are reconstructible from their shortest distance matrices if and only if they do not contain a subgraph from a certain family of graphs. We also show that networks with, in a certain sense, leaves everywhere, are reconstructible from their induced shortest distance matrix.

7.1. INTRODUCTION

In this chapter we look at a third type of substructure that could be used for encoding phylogenetic networks. Where previous chapters considered subnetworks and cherry-picking sequences, this chapter studies whether some unrooted network classes can be encoded by their inter-taxa distances. Distance matrices labelled by a set of taxa X describe the genetic distance between pairs of taxa.

Because networks contain undirected cycles, there can be many paths between two leaves (leaves are labelled by unique taxa, so leaves and taxa will be used interchangeably). This is in contrast to trees which contain only one path between each leaf pair. So while a tree induces only one metric, networks may induce many. A matrix consisting of inter-leaf shortest distances, and a matrix consisting of inter-leaf longest distances

The contents of this chapter have been published in [1] and in [2].

are two examples of a metric induced by a network. To date, different matrices have been considered for phylogenetic network reconstruction. These include the shortest distances (the traditional distance matrix), the sets of distances [3], and the multisets of distances [1, 4], where the latter two are not distance matrices in the traditional sense of the term. An element of the multisets of distances is a multiset of all distances between a pair of leaves, together with the multiplicities associated to each length. The set of distances can be obtained from the multisets of distances by ignoring the multiplicities[1].

Both the sets and multisets of distances were first introduced to prove reconstructibility results for distance matrices induced by particular phylogenetic network classes [3, 4]; shortest distances have also been used to prove reconstructibility results (split networks [6]; cactus graphs [7]). Reconstructibility results have also been proven on other distance metrics (TOM-networks) [8]. Recent results have shown unique realizability from sets and multisets of distances for certain rooted networks (tree-child and normal networks) [9, 10] and for certain unrooted networks (level-k networks) [1].

In particular, the paper "Reconstructibility of unrooted level-k phylogenetic networks from distances" showed that unweighted binary level-2 networks are reconstructible from their multisets of distances [1]. They also showed that level-1 networks are reconstructible from their shortest distances, but that level-2 networks were not reconstructible in general from their shortest distances (Figure 7.1). It was also shown that level-k networks for $k > 2$ are not reconstructible from their multisets of distances, in general. The results presented in this chapter are stronger than those presented in [1], and therefore we do not include the contents of that paper in this chapter.

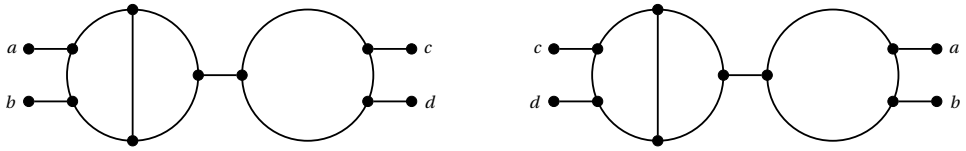


Figure 7.1: Two level-2 networks with the same shortest distances between any pair of leaves. The shortest distance can be worked out by taking the length of the shortest path between a pair of leaves, where each edge is of length 1.

This is our starting point for this chapter. Here we investigate the level-2 networks that are reconstructible from their shortest distances, and in the case it is not possible, the additional information that is needed to be able to do so. There are two main motivations behind this. Firstly, not much is known regarding the unique reconstructibility of level-k unrooted networks from their induced distance matrices. Of the reconstructibility results discussed in the previous paragraph, only two focus specifically on level-k unrooted networks [1, 7]. To add to this, reconstructibility implies an existence of an algorithm for constructing level-2 networks from a distance matrix (a unique network if

[1] The motivation for considering multisets of distances is mostly combinatorial. It is not clear how such distances can be obtained from a multiple sequence alignment. A possibility is to divide the alignment into *blocks* depending on the parts of the chromosome responsible for encoding a particular gene, or by optimizing constraints such as the homoplasy score [5]. Treating each of these blocks as an alignment yields a distance matrix for each block, which can be collated to give multisets of distances between pairs of taxa. However, every multiset of distances between leaf pairs would be of the same size (in particular the number of blocks), which is not always the case in the results where these multisets are used.

given the right input). While we do not explicitly give the algorithm in this chapter, it is possible to infer it from the results provided here. Secondly, we have seen that multisets of distances have been used to show reconstructibility results both for rooted and unrooted networks. The problem is that it is not clear how such distances can be obtained from a multiple sequence alignment. A possibility is to divide the alignment into *blocks* depending on the parts of the chromosome responsible for encoding a particular gene, or by optimizing constraints such as the homoplasy score [5]. Treating each of these blocks as an alignment yields a distance matrix for each block, which can be collated to give multisets of distances between pairs of taxa. However, every multiset of distances between leaf pairs would be of the same size (in particular the number of blocks), which is rarely the case for multisets of distances induced by networks. Therefore we stay away from multisets of distances, and we show that the shortest and longest distances suffice in showing that level-2 networks are reconstructible. Note that these two distance metrics can be obtained from the outlined procedure above (in particular, the set of distances can be obtained).

Here, we answer three open problems for binary networks from [1] on unique realizability of certain distance matrices.

1. Networks with a leaf on every generator side are reconstructible from their induced shortest distance matrix in the space of all unrooted networks (Theorem 26);

2. Level-2 networks are reconstructible from their induced shortest and longest distance matrix in the space of unrooted level-2 networks (Theorem 28);

3. We characterize subgraphs of level-2 networks that are responsible for the class to not be reconstructible from their induced shortest distance matrix (Theorem 29).

Structure of the chapter In Section 7.2, we give formal definitions of phylogenetic terms. In Section 7.3, we show that networks with a leaf on every generator side (i.e., every vertex on the network is at most shortest distance-2 away from a leaf) are reconstructible from their shortest distances (Theorem 26). In Section 7.4, we show that level-2 networks are reconstructible from their sl-distance matrices (Theorem 28). This is proven by first showing that the splits of the network (cut-edges that induce a partition on the labelled leaves) are determined by the shortest distances that they realize (Theorem 27). In Section 7.5, we show a construction for obtaining pairs of distinct level-2 networks from a binary tree that realize the same shortest distance matrix. We show that having such a network as a subgraph renders a level-2 network to be non-reconstructible from their shortest distances, thereby characterizing the family of subgraphs that are responsible for the non-reconstructibility (Theorem 29). We close with a discussion in Section 7.7, presenting ideas for possible future directions.

7.2. PRELIMINARIES

A *blob* of a network is a maximal 2-connected subgraph with at least three vertices. Recall that a network is a *level-k network*, with $k \geq 0$, if at most k edges must be

deleted from every blob to obtain a tree. We call a cut-edge *trivial* if the edge is incident to a leaf, and *non-trivial* otherwise. Given a cut-edge uv we say that a leaf x can be *reached from* u (via uv) if, upon deleting the edge uv without suppressing degree-2 vertices, x is in the same component as v in the resulting subgraph. We say that a leaf is *contained* in a blob if the neighbour of the leaf is a vertex of the blob. We say that a chain is *contained* in a blob if any of the leaves of the chain are contained in the blob (and therefore all leaves of the chain are contained in the blob). An edge is *incident* to a blob if exactly one of the endpoints of the edge is a vertex of the blob. A blob is *pendant* if there is exactly one non-trivial cut-edge that is incident to the blob. We say that a leaf x can be *reached* from a blob B via a cut-edge uv if u is a vertex of B and x can be reached from u via uv. In this case, we also say that uv or u *separates* x from B.

Letting X be a set of taxa, a *split* on X is a partition $\{A, B\}$ of X. We denote a split which induces the partition $\{A, B\}$ of X by $A|B$ where the order in which we list A and B does not matter. Observe that some cut-edges of a network on X naturally induce a split as there are exactly two parts of the network separated by the edge. We call this a *cut-edge induced split*. We call a split $A|B$ *non-trivial* if both A and B contain at least two elements. Otherwise we call a split *trivial*. Observe that non-trivial cut-edges induce non-trivial splits, and that trivial cut-edges induce trivial splits.

In this chapter, we assume the restriction that every cut-edge must induce a unique split. Firstly, such a restriction eliminates the possibility for networks to contain *redundant* blobs, which are pendant blobs that contain no leaves. Secondly, the restriction removes all blobs that do not contain leaves, that are incident only to two non-trivial cut-edges. Such blobs can be interpreted as higher-level analogues of parallel edges.

The *generator* $G(N)$ of a network N is the multi-graph obtained by deleting all pendant subtrees (i.e., deleting all leaves from N) and suppressing degree-2 vertices. The generator may contain loops and parallel edges. A vertex of N that is not deleted or suppressed in the process of obtaining $G(N)$ is called a *generator vertex*. We call the edges of $G(N)$ the *sides* of N. Observe that the sides of N correspond to paths of N. Let s be a side of N, and let $e_0 v_1 v_2 \cdots v_k e_1$ with $k \geq 0$ denote the path in N corresponding to s, where e_0 and e_1 are vertices of the generator. If $k = 0$, then the path is simply the edge $e_0 e_1$. We call e_0 and e_1 the *boundary vertices of side* s. We say that a leaf x *is on* side s if x is a neighbour of v_i for some $i \in [k]$. We say that a chain *is on* side s if all leaves of the chain are on the side s. Observe that a leaf of a chain is on a side if and only if the chain is on the side. Observe also that $v_1 \cdots v_k$ is a spine of some chain on s. We say that a side is *empty* if no leaves are on the side. A *side of a blob* B is an edge of $G(N)$ which corresponds to a path in B. Observe that level-2 blobs contain exactly two vertices that are not cut-vertices. We call these the *poles* of the blob. There are exactly three edge-disjoint paths between the two poles. We call these three paths in N the *main sides* of B. The vertices in a main side s of B that are adjacent to the endpoints of s are called the *main end-spine vertices*.

We adopt the following notation for pendant level-2 blobs from [1]. Let B be a pendant level-2 blob, and let a, b, c, d denote the four chains contained in B of lengths $k, \ell, m, n \geq 0$, respectively, such that chains c and d are on the same main side of B as the non-trivial cut-edge. Then we say that B is of the form (a, b, c, d) (see Figure 7.2). The order of the first two elements a, b, and the order of the last two elements c, d do not matter. For

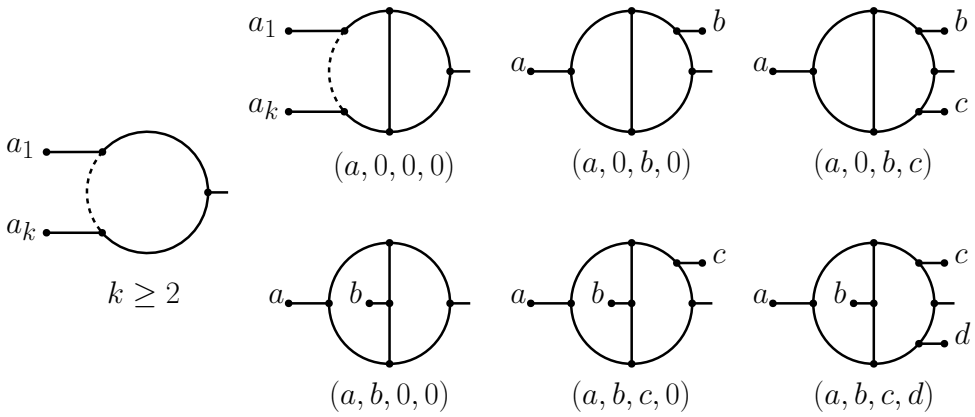


Figure 7.2: The seven possible pendant blobs in a level-2 network. The leftmost pendant blob is level-1, and the other pendant blobs are all level-2. Observe that for the pendant level-1 blob and the pendant level-2 blob of the form $(a,0,0,0)$, the length of the chain a must be of length at least 2. This is due to the fact that networks do not contain parallel edges nor level-2 blobs with only two cut-edges incident to it. The dashed edges in these two blobs indicate that the chain can be longer than 2. For the five other pendant blobs, each of the leaves a,b,c,d indicates a chain.

ease of notation, a side without leaves is seen as a length-0 chain. Note that since every cut-edge induces a unique split, it is not possible to obtain the pendant blob of the form $(1,0,0,0)$.

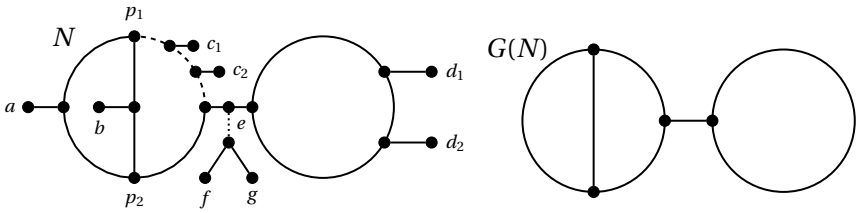
7.2.1. DISTANCES

For a network N on X, we let $d_m^N(x,y)$ and $d_l^N(x,y)$ denote the length of a shortest and a longest path between two vertices x,y in N, respectively. We exclude the superscript N when there is no ambiguity on the network at hand. Let $a = \{a_1,\ldots,a_k\}$ be a set of vertices in N, and let u be a vertex in N that is not in a. Then we define the shortest distance from u to a as the shortest distance from u to any of the vertices in a, that is, $d_m^N(a,u) = \min\{d_m(a_i,u) : i \in [k] = \{1,\ldots,k\}\}$[2]. Similarly, define the longest distance from u to a as the longest distance from u to any of the vertices in a, that is, $d_l^N(a,u) = \max\{d_l(a_i,u) : i \in [k]\}$.

The *shortest distance matrix* $\mathscr{D}_m(N)$ of N is the $|X| \times |X|$ matrix whose (x,y)-th entry is $d_m^N(x,y)$. A network N *realises* the shortest distance matrix $\mathscr{D}_m$ if $\mathscr{D}_m(N) = \mathscr{D}_m$. *reconstructible* within a class $\mathscr{C}$ from its shortest distance matrix if N is the only network in $\mathscr{C}$, up to isomorphism, that realises $\mathscr{D}_m(N)$. Here, we say that two networks N and N' on X are *isomorphic* if there exists a bijection f from the vertices of N to the vertices of N', such that uv is an edge of N if and only if $f(u)f(v)$ is an edge of N', and the leaves of N are mapped to leaves of N' of the same label. Similarly, we define the *sl-distance matrix (shortest longest - distance matrix)* $\mathscr{D}(N)$ as the $|X| \times |X|$ matrix whose (x,y)-th entry is $d^N(x,y) = \{d_m^N(x,y), d_l^N(x,y)\}$. We say that a network N realizes the sl-distance matrix $\mathscr{D}$ if $\mathscr{D}(N) = \mathscr{D}$. A network N is *reconstructible* within a class $\mathscr{C}$ from its sl-distance matrix if N is the only network in $\mathscr{C}$, up to isomorphism, that realises $\mathscr{D}(N)$.

[2] For consistency later on the section, we let $[0] = \emptyset$, the empty set.

7



	a	b	c_1	c_2	d_1	d_2	f	g
a		$(4,8)$	$(4,8)$	$(5,7)$	$(7,12)$	$(7,12)$	$(6,10)$	$(6,10)$
b			$(4,8)$	$(5,7)$	$(7,12)$	$(7,12)$	$(6,10)$	$(6,10)$
c_1				$(3,7)$	$(7,10)$	$(7,10)$	$(6,8)$	$(6,8)$
c_2					$(6,11)$	$(6,11)$	$(5,9)$	$(5,9)$
d_1						$(3,4)$	$(5,6)$	$(5,6)$
d_2							$(5,6)$	$(5,6)$
f								$(2,2)$
g								

Figure 7.3: A level-2 network N on the taxa set $\{a,b,c_1,c_2,d_1,d_2,f,g\}$, its generator $G(N)$, and its sl-distance matrix. N contains a cherry $\{f,g\}$ and four chains $(a),(b),(c_1,c_2)$ and (d_1,d_2). N contains two pendant blobs: the leftmost is a level-2 blob of the form $((a),(b),(c_1,c_2),\emptyset)$, and the rightmost is a level-1 blob containing the leaves d_1 and d_2. The poles of the pendant level-2 blob are labelled by p_1 and p_2. The dotted cut-edge e induces the non-trivial split $\{a,b,c_1,c_2,d_1,d_2\}|\{f,g\}$. The blob side indicated by the dashed path contains the chain (c_1,c_2). The chains (a) and (c_1,c_2) are adjacent once. The chains (a) and (b) are adjacent twice. The sl-matrix has ij-th elements of the form (x,y), where x and y denote the shortest and longest distances between i and j in N. The diagonal elements, which are all $(0,0)$, and the lower triangular elements are omitted as the matrix is symmetric.

7.2.2. REDUCING CHERRIES

By definition, we may identify cherries from shortest distance matrices.

Observation 16. *Let $\mathscr{D}_m$ be a shortest distance matrix. A network N on X that realises $\mathscr{D}_m$ contains a cherry $\{x,y\}$ if and only if $d_m(x,y)=2$.*

Reducing a cherry $\{x,y\}$ to a leaf z from N is the action of deleting both leaves x,y and labelling the remaining unlabelled degree-1 vertex as z, assuming that $z \notin X$ (this vertex was the neighbour of x and y in N). As a result of reducing the cherry $\{x,y\}$, observe that the shortest distance between two leaves that are both not z are unchanged; the shortest distance between z and another leaf $l \in X-\{x,y\}$ is exactly one less than that of x and l in N.

Observation 17. *Let N be a network on X containing a cherry $\{x,y\}$. Upon reducing the cherry to a leaf z, we obtain a network N' on $X'=X\cup\{z\}-\{x,y\}$ such that the shortest*

distance matrix for N' contains the elements

$$d_m^{N'}(a,b) = \begin{cases} d_m^N(a,b) & \text{if } a,b \in X - \{x,y\} \\ d_m^N(a,x) - 1 & \text{if } a \in X - \{x,y\} \text{ and } b = z. \end{cases}$$

In the setting of Observation 17, one may obtain a network that is isomorphic to N from N' by adding two labelled vertices x and y, adding the edges zx and zy, and unlabelling the vertex z. We call this *replacing z by a cherry $\{x,y\}$*.

Observation 18. *Let N be a network on X, and let z be a leaf in N. Let $x,y \notin X$ be leaf labels that do not appear in N. Then upon replacing z by a cherry $\{x,y\}$, we obtain a network M on $Y = X \cup \{x,y\} - \{z\}$ that realises the shortest distance matrix with entries*

$$d_m^M(a,b) = \begin{cases} d_m^N(a,b) & \text{if } a,b \in Y - \{x,y\} \\ d_m^N(a,z) + 1 & \text{if } a \in Y - \{x,y\} \text{ and } b \in \{x,y\} \\ 2 & \text{if } a = x \text{ and } b = y. \end{cases}$$

It is easy to see that replacing a leaf by a cherry and reducing a cherry are inverse operations of one another.

Lemma 68. *Let N be a network with a cherry $\{x,y\}$, and let N' denote the network obtained by reducing the cherry from N to a leaf z. Then N is reconstructible from its shortest distance matrix if and only if N' is reconstructible from its shortest distance matrix.*

Proof. Suppose first that the network N is reconstructible from its shortest distance matrix. Suppose for a contradiction that the shortest distance matrix $\mathscr{D}_m(N')$ of N' is also realised by a network N'' that is not isomorphic to N'. Consider the networks M' and M'' obtained from N' and N'', respectively, by replacing z by a cherry $\{x,y\}$. By Observation 18, the two distinct networks M' and M'' realise the same shortest distance matrix. However, this shortest distance matrix is precisely $\mathscr{D}_m(N)$, since M' is isomorphic to N. This contradicts the fact that N is reconstructible from its shortest distance matrix. Therefore N' must be reconstructible from its shortest distance matrix.

Now suppose that the network N' is reconstructible from its shortest distance matrix. If there were two distinct networks N and M realising $\mathscr{D}_m(N)$, then these networks must both contain the cherry $\{x,y\}$. Reducing this cherry to a leaf z, we see by Observation 17 that both reduced networks, which are distinct, realise the same shortest distance matrix, which is exactly $\mathscr{D}_m(N')$. However, this is not possible, as N' is reconstructible from its shortest distance matrix. Therefore N is also reconstructible from its shortest distance matrix. □

Let N be a network. *Subtree reduction* refers to the action of reducing cherries of N until it is no longer possible to do so. We refer to the resulting network as the *subtree reduced version* of N. Note that the subtree reduced version of N is unique, and the order in which the cherries are reduced does not matter. The following corollary follows immediately by applying Lemma 68 to every cherry that is reduced in the subtree reduction.

Corollary 12. *A network N is reconstructible from its shortest distance matrix if and only if the subtree reduced version of N is reconstructible from its shortest distance matrix.*

Note that Observations 17 and 18, Lemma 68, and Corollary 12 can naturally be extended to the sl-distances, with a single tweak for Observations 17 and 18, where the longest distances are adjusted exactly the same as done for the shortest distances (replace d_m by d_l wherever possible). This means we may assume for the rest of the chapter, that all networks have undergone subtree reduction, and therefore that all networks contain no cherries.

7.2.3. CHAINS

Upon reducing all cherries from our networks, we may identify unique chains from shortest distance matrices. Recall that chains are written as sequences $a = (a_1, \ldots, a_k)$ for some $k \geq 1$. We shall sometimes write these as (a, k). In what follows, we will often require a way of referring to leaves of the network that are not in a particular chain. So while a is a sequence of leaves, we shall sometimes treat a as a set of leaves, e.g., $X - a = \{l \in X : l \neq a_i \text{ for } i \in [k]\}$.

Observation 19. *Let $\mathscr{D}_m$ be a shortest distance matrix. A network N on X that realises $\mathscr{D}_m$ contains a chain $a = (a_1, \ldots, a_k)$ where $k \geq 1$ if and only if $d_m(a_i, a_{i+1}) = 3$ for all $i \in [k-1]$ and there exists no leaf $l \in X - a$ such that $d_m(a, l) = 3$.*

Observation 19 implies that the leaves in a network without cherries can be partitioned into chains. Indeed, no leaf can be contained in two distinct chains, as otherwise the chains would be non-maximal. Let (a, k) and (b, ℓ) be two distinct chains. We say that (a, k) and (b, ℓ) are *adjacent* if $d_m(a_i, b_j) = 4$ for some combination of $i \in \{1, k\}$ and $j \in \{1, \ell\}$. Observe that adjacent chains of a network N can be identified from shortest distance matrices, by first partitioning the leaf set of N into chains and then checking for chain end-leaves that are distance-4 apart. We say that two chains (a, k) and (b, ℓ) are *adjacent once* if exactly one distinct pair of (a, k) and (b, ℓ) are distance-4 apart. We say that the chains (a, k) and (b, ℓ) are *adjacent twice* if two distinct pairs of (a, k) and (b, ℓ) end-leaves are distance-4 apart. Since we assume networks to be binary, two chains may be adjacent at most twice. In the special case where $k = \ell = 1$, we can only tell whether the chains are adjacent from the shortest distances. We cannot tell whether they are adjacent twice. This can however be inferred from the sl-distance matrix.

7.2.4. KNOWN RESULTS

The following results appeared in [1].

Lemma 69 (Theorem 4.2; Lemma 4.4; Lemma 5.3 of [1])**.** *Let N be a level-2 network on $|X|$. Then N is reconstructible, within the class of level-2 networks, from its shortest distance matrix if N is also level-1, if $|X| < 4$, or if N contains only one blob.*

Therefore we may assume that the networks we consider are always at least level-2 on at least four leaves, and that the network contains at least two blobs. Furthermore, from Section 7.2.2, we may assume that the networks contain no cherries.

7.3. LEAF ON EACH GENERATOR SIDE

In this section, we consider networks with at least one leaf on each generator side, and show that such networks are reconstructible from their shortest distance matrices, regardless of level. Let N be one such network. Since we may assume that N has no cherries, each side of N can be determined by the chain contained therein. Furthermore, two sides are adjacent (i.e., the sides share a common endpoint in $G(N)$) if and only if the chains on the sides are adjacent. Since chains partition the leaf set of N, this implies that the structure of the generator $G(N)$, and therefore the structure of the network N is determined by the chains of N and their adjacency in N.

Every vertex in N is either a leaf, a spine vertex of some chain, or a generator vertex. Since networks considered here are binary, exactly two or three generator sides may be incident to the same vertex in $G(N)$ (as per conventional graph theory, we say that an edge is *incident* to its endpoints). If a vertex is incident to exactly two sides in $G(N)$, then one of these sides must be a loop. Loops in $G(N)$ correspond to pendant level-1 blobs in N. Suppose that (a,k) is a chain (recall that this is a chain of length k) and is adjacent to exactly one chain (b,ℓ) twice, and that (a,k) is not adjacent to any other chains. Then (a,k) is contained in a pendant level-1 blob, since we may assume that N is a level-2 network with at least two blobs. Note that $k \geq 2$ as N contains no parallel edges. In such a case, we call the pair (a,b) the *bulb* of a and b. We say that a is contained in the bulb as the *petal*. We say that N *contains* the petal (a,b).

If a generator vertex is incident to three sides, then the three distinct chains in the network, corresponding to these three sides must be pairwise adjacent. Now consider three pairwise adjacent distinct chains $(a,k),(b,\ell),(c,m)$ in N. Since we may assume N is not a level-2 network with a single blob (as we know such networks are reconstructible from their shortest distances by Lemma 69), any three chains may be pairwise adjacent at most once. In particular, the end-leaves of a,b,c that are adjacent are unique. In this case, we say that (a,b,c) forms a *pairwise adjacent triple* (see Figure 7.4 for examples of pairwise adjacent triples and petals). Therefore, if (a,b,c) is a pairwise adjacent triple, then there is a generator vertex that is incident to three sides such that one side contains chain a, one chain b and one chain c. We say that N *contains* the pairwise adjacent triple (a,b,c).

Note that bulbs and pairwise adjacent triples both consist of three leaves. In what follows, the notion of a *median* vertex will be important. Given three vertices a,b,c of a network, a median of a,b,c is a vertex that belongs to a shortest path between each pair of a,b,c. A median may not always exist for any three vertices (consider, for example, a cycle on three vertices). However, for our purposes, we shall consider medians of three leaves, which will always exist. Moreover, a median of three vertices is not necessarily unique, as there may be more than one shortest path between a pair of vertices in a network.

Lemma 70. *Let $\mathscr{D}_m$ be a shortest distance matrix. Then $\mathscr{D}_m$ can only be realised by a network N with leaves on each side of the generator, where N is not a single level-2 blob if and only if each chain of $\mathscr{D}_m$ is contained in either*

(i) two distinct pairwise adjacent triples; or

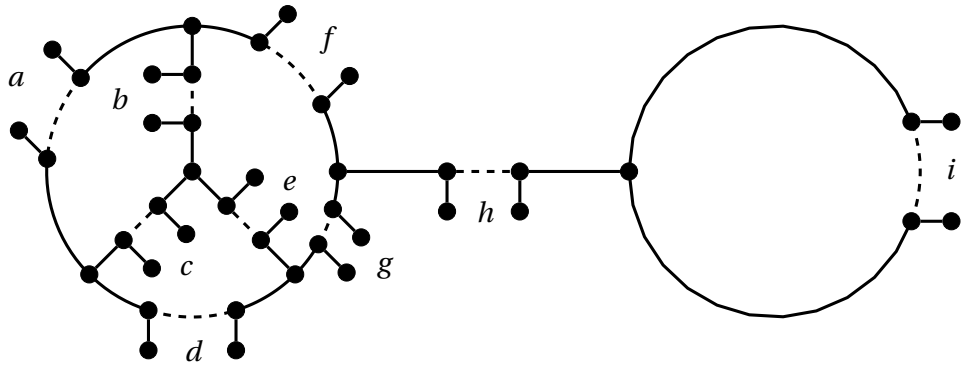


Figure 7.4: A level-3 network with leaves on every generator side. Each letter represents a chain on a side of the network. The network contains the pairwise adjacent triples $(a,b,f),(a,c,d),(b,c,e),(d,e,g),(f,g,h)$ and the bulb (i,h) with i as the petal. The dashed edges indicate how each of the nine chains is of length at least 1.

(ii) one pairwise adjacent triple and one bulb as a non-petal; or

(iii) one bulb as the petal; or

(iv) two bulbs as non-petals.

Proof. Let N be a network with leaves on each generator side and suppose that N realises $\mathscr{D}_m$. Then each generator vertex of N is the median of three end-leaves of (not necessarily distinct) chains, such that these end-leaves are pairwise shortest distance-4 apart. Any three chains may be pairwise adjacent at most once (unless N is a network with a single level-2 blob, but we have specifically excluded this case in the statement of the lemma), and a chain contained in a pendant level-1 blob is adjacent twice to exactly one other chain. As N is binary, these median vertices encode either pairwise adjacent triples or bulbs. By *encode*, we mean that for every three end-leaves that are pairwise distance-4 apart, the median vertex corresponding to it is unique. Each chain is contained in exactly two of such constructs, where being contained in a bulb as the petal counts as two, since each side has two boundary vertices (except for the loop). The result follows immediately.

To show the other direction of the lemma, we prove the contrapositive. Let N be a network that has at least one empty generator side, and suppose that N realises $\mathscr{D}_m$. We want to show that at least one chain of N does not satisfy any of the four properties $(i)-(iv)$ as stated in the statement of the lemma. Find adjacent sides s_1 and s_2 of N, such that s_1 contains a chain c while s_2 is empty. Clearly, c cannot be contained in a bulb as its petal, since s_2 contains no chains to which c can be adjacent twice (cannot satisfy (iii)). So we may assume that c is not contained in a pendant level-1 blob, and therefore that the boundary vertices e_0,e_1 of s_1 are distinct. We may assume without loss of generality that e_0 is the boundary vertex of s_2. Since s_2 is empty, e_0 cannot be a median of three distinct end-leaves of chains. This implies that c can only be contained in exactly one pairwise adjacent triple, or in exactly one bulb as a non-petal (which is encoded by e_1) (cannot satisfy $(i),(ii)$, nor (iv)). □

Lemma 71. *Let N and N' be networks with a leaf on each generator side, such that neither N nor N' are level-2 and contain precisely one level-2 blob. Then N and N' are isomorphic if and only if they contain the same chains, the same pairwise adjacent triples, and the same bulbs.*

Proof. Suppose first that N and N' contain the same chains, the same pairwise adjacent triples, and the same bulbs. Then the networks must contain the same leaves and the same spine vertices (and also the edges therein). The remaining vertices in N and N' are their generator vertices, and the remaining edges are those incident on generator vertices and the end-spine vertices.

We show first that $G(N) = G(N')$. Every edge in the generator is a side that contains a chain. Since N and N' have the same chains, the number of edges in $G(N)$ is the same as that in $G(N')$. Every vertex in the generator is a median of end-leaf vertices of three (not-necessarily distinct) chains. These generator vertices uniquely encode a pairwise adjacent triple or a bulb, since N and N' are not level-2 networks that contain precisely one level-2 blob. Since N and N' have the same pairwise adjacent triples and the same bulbs, $G(N)$ and $G(N')$ must have the same number of vertices. To see that $G(N) = G(N')$, observe that each generator vertex that encodes the pairwise adjacent triple (a,b,c) or a bulb (a,b) links the generator edges that contain the chains a,b,c or a,b, respectively. This means that two generator edges share a common endpoint if and only if the chains that they contain are in the same pairwise adjacent triple or bulb.

To see that N is isomorphic to N', simply attach all chains to their corresponding generator sides, noting that the placement of the end-leaves are determined by the composition of the pairwise adjacent triples. Since N and N' contain the same pairwise adjacent triples and bulbs, they must be isomorphic.

Conversely, if two networks are isomorphic, then they must have the same chains, the same pairwise adjacent triples, and the same bulbs. □

Theorem 26. *Given a network N with a leaf on each generator side, N is the only network that realizes $\mathscr{D}_m(N)$. That is, networks with leaves on each generator side are reconstructible, within the class of all unrooted networks, from their shortest distances.*

Proof. If N is a level-2 network with a single blob, then N is reconstructible from its shortest distances by Lemma 69. Therefore we may assume N is not a level-2 network on a single blob, and therefore we may call Lemmas 70 and 71.

Let N be a network with a leaf on each generator side. This means that every chain in N satisfies one of properties $(i)-(iv)$ of Lemma 70. As before, let $\mathscr{D}_m(N)$ be the shortest distance matrix of N. Suppose that N' is another network that realises $\mathscr{D}_m(N)$. Because each chain of $\mathscr{D}_m(N)$ satisfies one of the four properties $(i)-(iv)$ of Lemma 70, N' must be a network with a leaf on each generator side. Furthermore, any network realising $\mathscr{D}_m(N)$ must contain the same chains as N, by Observation 19. Therefore N and N' have the same chains. To see that N and N' also have the same generator vertices, observe that N and N' contain the same pairwise adjacent triples and the same bulbs; these can indeed be inferred from chain adjacencies, which can be inferred from $\mathscr{D}_m(N)$ by definition of adjacent chains. It follows by Lemma 71 that N and N' must be isomorphic. □

7.4. LEVEL-2 RECONSTRUCTIBILITY FROM SL-DISTANCE MATRIX

As was pointed out in [1], level-2 networks are in general not reconstructible from their induced shortest distance matrix. Figure 7.1 illustrates two distinct level-2 networks on four leaves with the same shortest distance matrices (Figure 2 of [1]). In this section, we show that level-2 networks are reconstructible from their sl-distance matrix.

7.4.1. CUT-EDGES

First, we show that for a level-2 network, we may obtain all the cut-edge induced splits from its shortest distance matrix. Though the section is concerned with sl-distance reconstructibility, we show that the shortest distance matrix suffices in obtaining the cut-edge induced splits.

Theorem 27. *All cut-edge induced splits of a level-2 network N may be obtained from its shortest distance matrix $\mathscr{D}_m(N)$. A split $A|B$ is induced by a cut-edge of N if and only if for all $a, a' \in A$ and $b, b' \in B$,*

(i) $d_m(a,b) + d_m(a',b') = d_m(a,b') + d_m(a',b)$*; and*

(ii) $d_m(a,a') + d_m(b,b') \leq d_m(a,b) + d_m(a',b') - 2$.

Proof. The first statement of the theorem follows from the second statement of the theorem. Here, we prove the second statement.

Let N be a level-2 network on X. Suppose first that $A|B$ is a split induced by some cut-edge uv in N. Let $a, a' \in A$ and $b, b' \in B$ be arbitrarily chosen. Since every shortest path from a leaf of A to a leaf of B contains the edge uv, we must have that

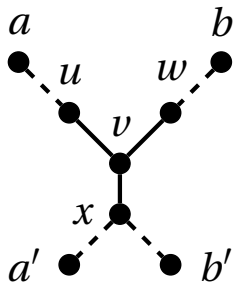
$$\begin{aligned} d_m(a,b) + d_m(a',b') &= d_m(a,u) + d_m(u,b) + d_m(a',u) + d_m(u,b') \\ &= d_m(a,b') + d_m(a',b). \end{aligned}$$

So property (i) holds. Since the length of the edge uv is 1, property (ii) also holds because

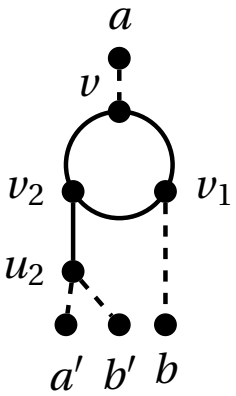
$$\begin{aligned} d_m(a,b) + d_m(a',b') &= d_m(a,u) + 1 + d_m(v,b) + d_m(a',u) + 1 + d_m(v,b') \\ &\geq d_m(a,a') + d_m(b,b') + 2, \end{aligned}$$

where in particular, we obtain equality if there exist a shortest path between a and a' and a shortest path between b and b' containing the vertices u and v, respectively.

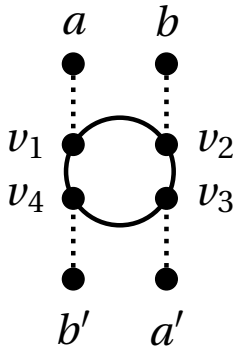
Now suppose that properties (i) and (ii) hold for a partition $A|B$ of the leaf-set of N. We shall show that $A|B$ is induced by a cut-edge of N. Let $a \in A$, and let $e = uv$ be a cut-edge in N that is farthest from a, such that e induces a split which separates a from B. If e induces the split $A|B$, then we are done. So suppose that there exists an $a' \in A$ such that e induces a split that separates a from $B \cup \{a'\}$ (in particular, we may assume that $|A| \geq 2$ as every trivial split is clearly induced by a cut-edge). Without loss of generality, suppose that u is closer to a than to v. We consider several cases (see Figure 7.5 for an illustration of the cases).



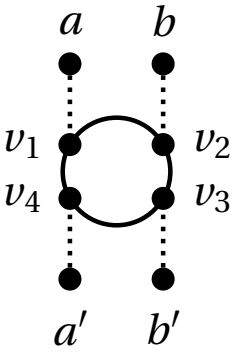
(a) Case 1.



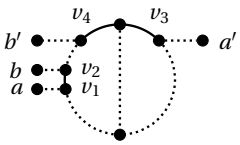
(b) Case 2. (a), when two leaves from the two partitions (a' and b') are reachable from the blob via the same cut-edge.



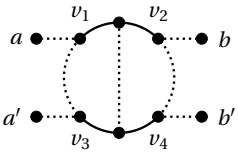
(c) Case 2. (a) i.



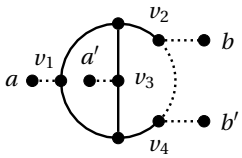
(d) Case 2. (a) ii.



(e) Case 2. (b) i. The bottom-left dashed edge is a potential type-B cut-edge. This edge may or may not exist in the network (does not affect the case).



(f) Case 2. (b) ii. A.



(g) Case 2. (b) ii. B.

Figure 7.5: All cases examined in the proof of Theorem 27. The dotted edges represent a path between the two vertices.

1. ***v* is not in a blob:** Let w, x denote the two neighbours of v that are not u. By our choice of e, there must be a leaf $b \in B$ that can be reached from the edge vw, and a leaf $b' \in B$ that can be reached from the edge vx. Without loss of generality, assume that a' can be reached from the edge vx. But this means that

$$\begin{aligned} d_m(a,b) + d_m(a',b') &\leq d_m(a,v) + d_m(v,b) + d_m(a',x) + d_m(x,b') \\ &= [d_m(a,v) + d_m(v,x) + d_m(a',x)] - d_m(v,x) \\ &\quad + [d_m(b,v) + d_m(v,x) + d_m(x,b')] - d_m(v,x) \\ &= d_m(a,a') - d_m(v,x) + d_m(b,b') - d_m(v,x) \\ &= d_m(a,a') + d_m(b,b') - 2 \\ &< d_m(a,a') + d_m(b,b'), \end{aligned}$$

   where the first inequality may be strict since the shortest path between a' and b' may not pass through x. This contradicts the second condition of the claim.

2. ***v* is a vertex of a blob *C*:** The blob C must be incident to at least two cut-edges e_1, e_2 other than uv, for which there must be elements b and b' in B that are reachable from e_1 and e_2 respectively. Otherwise, as before, this would contradict our choice of a farthest uv. We claim that if a' can be reached from either e_1 or e_2, then we would reach a contradiction. Without loss of generality, suppose that a' can be reached from e_2. Letting $e_1 = u_1v_1$ and $e_2 = u_2v_2$ where v_1 and v_2 are vertices on C, we have that

$$\begin{aligned} d_m(a,b) + d_m(a',b') &< d_m(a,v) + d_m(v,v_1) + d_m(v_1,b) + d_m(a',v_2) + d_m(v_2,b') \\ &= d_m(a,a') - d_m(v,v_2) + d_m(b,b') - d_m(v_1,v_2) + d_m(v,v_1) \\ &\leq d_m(a,a') + d_m(b,b'), \end{aligned}$$

   where the first inequality follows as the shortest path between a' and b' does not contain v_2, and the final inequality follows from the triangle inequality. This contradicts the second condition of the claim. Therefore, we may assume from now that there are at least four cut-edges incident to the blob C and that no leaves from A and B can be reached from the same cut-edge incident to C. That is, every cut-edge incident to C induces a split that has either a subset of A or a subset of B as one of its parts. We refer to these as *type-A cut-edges* and *type-B cut-edges*, respectively.

   We have another case that is common both for the instances when C is either a level-1 or a level-2 blob. Suppose first that there exist two pairs of cut-edges e_1, e_2, and e_3, e_4 incident to C, whose endpoints are adjacent, respectively, such that all four edges are distinct and a, b, a', b' are reachable from e_1, e_2, e_3, e_4 respectively. We let v_i denote the vertices of C that are endpoints of e_i for $i = 1, 2, 3, 4$, respectively. Then we have

$$\begin{aligned} d_m(a,b) + d_m(a',b') &= d_m(a,v_1) + d_m(v_1,v_2) + d_m(v_2,b) + d_m(a',v_3) \\ &\quad + d_m(v_3,v_4) + d_m(v_4,b') \\ &= d_m(a,a') + d_m(b,b') - d_m(v_1,v_3) - d_m(v_2,v_4) + 2 \\ &\leq d_m(a,a') + d_m(b,b'), \end{aligned}$$

where the second equality follows as $d_m(v_1, v_2) = d_m(v_3, v_4) = 1$. This contradicts the second inequality of the claim.

If C is a level-1 blob, then the above case always applies. Indeed, there must be at least four cut-edges incident to C, of which at least two are type-A and the remaining edges are type-B. If there was only one type-B edge, then such a cut-edge induces the split $A|B$, and we are done. So this implies that there are always two distinct pairs of type-A and type-B edges, whose endpoints on C are adjacent. Thus we may assume that C is a level-2 blob.

We may assume that each main side of C contains only type-A cut-edges or only type-B cut-edges, or a combination of the two, for which such a main side contains one type of cut-edges, a single cut-edge of the other type, and possibly cut-edges of the first type. For example, a path corresponding to a main side of B may be $e_0 v_1 \cdots v_k e_1$ where $k \geq 2$ and e_0, e_1 are boundary vertices. For some integer $j \leq k$, we have $v_1, v_2, \ldots, v_{j-1}, v_{j+1}, \ldots, v_k$ are incident to type-A cut-edges, and v_j is incident to a type-B cut-edge. We call such a main side a *combination side*. Observe that a combination side contains either one type-A or one type-B cut-edge. Also note that the blob C contains at most one combination side as otherwise there would be two distinct pairs of type-A and type-B edges, whose endpoints on C are adjacent.

(a) ***C* contains one combination side *s*:** Suppose without loss of generality that s is a combination side containing exactly one type-A cut-edge. Let v_1 denote the endpoint of this cut-edge on C, and let v_2 be an adjacent vertex on C that is incident to a type-B cut-edge. Since C is incident to at least two type-A cut-edges, there must be another main side s' of C that is incident to only type-A cut-edges. Similarly, since C is incident to at least two type-B cut-edges, there must be another type-B cut-edge e_4 that is incident to C. We may assume in particular that an endpoint v_4 of e_4 is a main end-spine vertex incident either to s or to the third main side of C. Either way, there must exist an end-spine vertex v_3 on s' such that $d_m(v_3, v_4) = 2$. Observing that v_3 is an endpoint of a type-A cut-edge, we may assume that the leaves a, b, a', b' are separated from C by v_1, v_2, v_3, v_4, respectively. Then,

$$\begin{aligned} d_m(a,b) + d_m(a',b') - 2 &= d_m(a,v_1) + d_m(v_1,v_2) + d_m(v_2,b) + d_m(a',v_3) \\ &\quad + d_m(v_3,v_4) + d_m(v_4,b') - 2 \\ &= d_m(a,a') + d_m(b,b') + d_m(v_1,v_2) + d_m(v_3,v_4) \\ &\quad - d_m(v_1,v_3) - d_m(v_2,v_4) - 2 \\ &\leq d_m(a,a') + d_m(b,b') + 1 + 2 - 2 - d_m(v_1,v_3) \\ &\quad - d_m(v_2,v_4) \\ &< d_m(a,a') + d_m(b,b'), \end{aligned}$$

since $d_m(v_1, v_3) \geq 1$ and $d_m(v_2, v_4) \geq 1$. This leads to a contradiction of the second condition.

(b) **Each main sides of *C* contain cut-edges of the same type:** Observe that at least one main side must contain at least two cut-edges, since there are at

least four cut-edges incident to C and C has three main sides. Without loss of generality, suppose that there is a main side s with at least 2 type-B edges.

i. **There is another main side s' with at least two type-A edges:** Then choose v_1, v_3 and v_2, v_4 to be the main end-spine vertices of s' and s, respectively, such that $d_m(v_1, v_2) = 2$ and $d_m(v_3, v_4) = 2$. Supposing that the leaves a, b, a', b' are separated from C by v_1, v_2, v_3, v_4, respectively, we have that

$$\begin{aligned} d_m(a,b) + d_m(a',b') &= d_m(a',b) + d_m(a,b') - d_m(v_1,v_4) - d_m(v_2,v_3) \\ &\quad + d_m(v_1,v_2) + d_m(v_3,v_4) \\ &\leq d_m(a',b) + d_m(a,b') - 3 - 3 + 2 + 2 \\ &< d_m(a',b) + d_m(a,b'), \end{aligned}$$

since $d_m(v_1, v_4) \geq 3$ and $d_m(v_2, v_3) \geq 3$, where this inequality is strict when the third main side contains no cut-edges. This clearly contradicts the first condition of the claim.

ii. **The other two main sides contain exactly one type-A edge each:** Choose v_1, v_3 to be the two possible vertices incident to the type-A cut-edges, and v_2, v_4 to be the main end-spine vertices of s. Supposing that the leaves a, b, a', b' are separated from C by v_1, v_2, v_3, v_4, respectively, we have that

$$\begin{aligned} d_m(a,b) + d_m(a',b') &= d_m(a,a') + d_m(b,b') + d_m(v_1,v_2) \\ &\quad + d_m(v_3,v_4) - d_m(v_1,v_3) - d_m(v_2,v_4) \\ &= d_m(a,a') + d_m(b,b') + 4 - 2 - d_m(v_2,v_4) \\ &< d_m(a,a') + d_m(b,b') + 2, \end{aligned}$$

which contradicts the second condition of the claim.

This covers all possible cases, for which we have obtained a contradiction in each case. Therefore $A|B$ must be a cut-edge induced split of the network. □

Note that Theorem 27 does not hold for networks of level at least 3 (see Figure 7.6). Let us call a split $A|B$ *minimal* if there exists no non-trivial split $A'|B'$ of the same network such that A' and B' are proper subsets of A, or such that one of A' and B' is a proper subset of A or B. We say that A and B are *minimal parts* of $A|B$, respectively. Note that minimal parts of a split may not be unique as two pendant blobs may be connected by a non-trivial cut-edge e, for which both parts of the split are minimal parts.

Lemma 72. *Let N be a level-2 network on X with at least two pendant blobs. Then N contains a pendant blob containing the set of leaves A if and only if $A|B$ is a minimal cut-edge induced non-trivial split where A is a minimal part.*

Proof. Suppose first that N contains a pendant blob C containing the set of leaves A. Then there exists exactly one non-trivial cut-edge e incident to C, which induces the non-trivial split $A|B$ (where $B = X - A$). To see that A is a minimal part, observe that

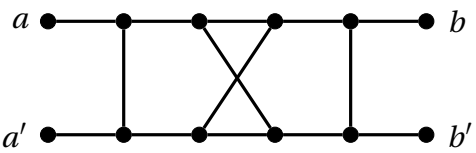


Figure 7.6: A level-3 network on leaf-set $\{a, a', b, b'\}$. Observe that the conditions for Theorem 27 are satisfied for $A = \{a, a'\}$ and $B = \{b, b'\}$, but there is no cut-edge that induces the split $A|B$.

for every cut-edge induced split $A'|B'$ where $A' \subseteq A$, we have $|A'| = 1$, since every cut-edge incident to C other than e is trivial. Therefore, $A|B$ is a minimal split, where A is a minimal part.

Suppose now that $A|B$ is a minimal non-trivial split induced by e, where A is a minimal part. Suppose for a contradiction that N did not contain a pendant blob with the set of leaves A. Because we may assume N contains no cherries, the part of N corresponding to the split part A (i.e., the graph obtained by deleting e and taking the component with the leaves from A) must contain a pendant blob C. Such a pendant blob contains the set of leaves A', where $A' \subseteq A$. The non-trivial cut-edge incident to C induces the split $A'|B'$, where $B' = X - A'$. By definition, $A'|B'$ must be a non-trivial split. But this contradicts the fact that $A|B$ was minimal. Therefore, N must contain a pendant blob with the set of leaves A. □

7.4.2. SL-DISTANCE RECONSTRUCTIBILITY

We show now that we can identify pendant blobs of level-2 networks from their sl-distance matrices.

Lemma 73. *Let N be a level-2 network on X with at least two blobs. Let $A|B$ be a non-trivial split of N where A is the minimal part. Then N contains a pendant blob containing the set of leaves A, and if A contains*

- *1 chain (a, k), then N contains*
  - *a pendant level-1 blob containing (a, k) if and only if*
    - *$2 \leq k \leq 3$, $d_m(a_1, a_k) = k + 1$, and $d_l(a_1, a_k) = 4$.*
    - *$k \geq 4$ and $d_m(a_1, a_k) = 4$.*
  - *a pendant level-2 blob of the form $(a, 0, 0, 0)$ if and only if*
    - *$2 \leq k \leq 3$, $d_m(a_1, a_k) = k + 1$, and $d_l(a_1, a_k) = 6$.*
    - *$k \geq 4$ and $d_m(a_1, a_k) = 5$.*
- *2 chains (a, k) and (b, ℓ), then N contains*
  - *a pendant level-2 blob of the form $(a, b, 0, 0)$ if and only if for all $x \in X - (a \cup b)$, we have $d_m(a, x) = d_m(b, x)$.*
  - *a pendant level-2 blob of the form $(a, 0, b, 0)$ if and only if for all $x \in X - (a \cup b)$, we have $d_m(a, x) = d_m(b, x) + 1$.*

7

- 3 *chains* $(a,k),(b,\ell)$, *and* (c,m), *then* N *contains*
  - *a pendant level-2 blob of the form* $(a,b,c,0)$ *if and only if for all* $x \in X-(a\cup b\cup c)$, *we have* $d_m(a,x)=d_m(b,x)=d_m(c,x)+1$.
  - *a pendant level-2 blob of the form* $(a,0,b,c)$ *if and only if for all* $x \in X-(a\cup b\cup c)$, *we have* $d_m(a,x)=d_m(b,x)+\min\{\ell,m\}+1=d_m(c,x)+\min\{\ell,m\}+1$.
- 4 *chains* $(a,k),(b,\ell),(c,m)$, *and* (d,n) *then* N *contains a pendant level-2 blob of the form* (a,b,c,d) *if and only if* (a,b,c) *and* (a,b,d) *are both pairwise adjacent triples.*

Proof. The fact that N contains a pendant blob containing the set of leaves A follows from Lemma 72.

Suppose first that N contains either a pendant level-1 blob or a pendant level-2 blob of the form (a,b,c,d), where a,b,c,d could be empty chains. Then it is easy to see by inspection that these distances hold and also that the pairwise adjacent triple statement holds in the case of 4 chains (see Figure 7.2).

To show the other direction, note that within a level-2 network, there is one possible level-1 pendant blob, and there are six possible level-2 blobs. We know that N contains a pendant blob with the leaves of A; it remains to show that if the conditions on the distances are satisfied, then N must contain the corresponding pendant blob. From the sl-distance matrix, we can infer the number of distinct chains contained in A, as well as their adjacencies. Then, we can infer the type of this pendant blob by looking at the distance from the leaves of A to some leaf that is not in A. We give one example here for the case when A consists of exactly three chains. The proof for the other cases follow in an analogous fashion.

We give a proof for the case when A contains 3 chains $(a,k),(b,\ell)$, and (c,m). Pendant level-1 blobs contain exactly 1 chain; thus the pendant blob must be level-2. Level-2 pendant blobs have three main sides, one of which contains the endpoint of the incident non-trivial cut-edge. This main side, say s, contains at least 1 chain and at most 2 chains, whilst the other two main sides contain at most 1 chain. Let $x \in X-a\cup b\cup c$ be an arbitrary leaf. Two of the chains, say a and b, have the same minimal distance to x, and the other chain c has different minimal distance. If the odd one out is shorter, say, then we know that c must be contained in the main side s of B, and we have a pendant level-2 blob of the form $(a,b,c,0)$. On the other hand, if the odd one out is longer, then we know that a and b must be contained in the main side s of B, and we have a pendant level-2 blob of the form $(c,0,a,b)$. □

Observe that in the proof of Lemma 73, the longest distance information was used only to distinguish the pendant level-1 blob with a chain (a,k) and the pendant level-2 blob of the form $(a,0,0,0)$ for $k \in \{2,3\}$. In other words, using only the shortest distances, the pendant level-1 blob containing 2 leaves cannot be distinguished from the pendant level-2 blob also containing 2 leaves on the same side; the pendant level-1 blob containing 3 leaves cannot be distinguished from the pendant level-2 blob of containing the same leaves on the same side. We shall denote these four subgraph structures as *bad blobs*. That is, we say that a level-1 blob is *bad* if it is incident to exactly three or four cut-edges. We say that a level-2 blob B is *bad* if, of the three main sides s_1,s_2,s_3 of B,

the main side s_1 is incident to a single cut-edge, s_2 is incident to no cut-edges, and s_3 is incident to exactly two or three cut-edges.

The reason why we cannot discern these bad blobs is because the shortest distance between the end-leaves of the chain uses the path containing the spine of the chain, which is the same length for both pendant level-1 and pendant level-2 blobs. Whenever these chains contain at least 4 leaves, a shortest path no longer contains the spine; since such paths differ in distance for pendant level-1 and pendant level-2 blobs with a single chain, we are able to identify such pendant blobs. We later show that level-2 networks that do not contain bad blobs are reconstructible from their shortest distances (Corollary 13).

The following lemma states that if we can identify certain structures within level-2 networks, then we may replace them by a leaf, and we may obtain the distance matrix of the reduced network.

Lemma 74. *Let N be a level-2 network on X with a pendant blob B, and replace B by a leaf $z \notin X$ to obtain the network N'. Letting Y denote the set of leaves contained in B, we have that the sl-distance matrix of N' contains the elements*

$$d^{N'}(p,q) = d^N(p,q)$$

for all pair of leaves $p, q \in X - Y$. Now, for all $p \in X - Y$, we have the following.

- ***B is a pendant level-1 blob with the chain (a,k):***

$$d^{N'}(p,z) = \{d_m^N(p,a) - 2, d_l^N(p,a) - (k+1)\}$$

- ***B is a pendant level-2 blob of the form F:***

$$d^{N'}(p,z) = \begin{cases} \{d_m^N(p,a) - 3, d_l^N(p,a) - (k+3)\} & \text{if } F = (a,0,0,0) \\ \{d_m^N(p,a) - 3, d_l^N(p,a) - (k+\ell+3)\} & \text{if } F = (a,b,0,0) \\ \{d_m^N(p,c) - 2, d_l^N(p,c) - (k+m+3)\} & \text{if } F = (a,0,c,0) \\ \{d_m^N(p,c) - 2, d_l^N(p,c) - (\max\{k,\ell\}+m+3)\} & \text{if } F = (a,b,c,0) \\ \{d_m^N(p,c) - 2, d_l^N(p,c) - (k+m+n+3)\} & \text{if } F = (a,0,c,d) \\ \{d_m^N(p,c) - 2, d_l^N(p,c) - (\max\{k,\ell\}+m+n+3)\} & \text{if } F = (a,b,c,d) \end{cases}$$

Proof. To obtain the inter-taxa distances for N', it suffices to simply subtract the shortest / longest distances from the vertex of the pendant blob incident to the non-trivial cut-edge to an end-spine leaf of a chain. These distances are easy to obtain as we know exactly what the pendant blobs are in all cases, due to Lemma 73. □

The above two lemmas will now be combined to prove the following result.

Theorem 28. *Level-2 networks are reconstructible within the class of level-2 networks, from their sl-distance matrix.*

Proof. We prove by induction on the size of the network. For the base case, we know by Lemma 69 that a network on a single blob is reconstructible from its shortest distances. So suppose that we are given a level-2 network N with $|E(N)|$ edges, and that the result holds for all level-2 networks with at most $|E(N)|-1$ edges.

We may assume that N contains at least two pendant blobs. By the results in Section 7.2.3, we can partition the leaves into chains, and adjacency between chains can be obtained from sl-distance matrices. By Theorem 27, we can obtain all cut-edge induced splits of N from its shortest distance matrix; by Lemma 73, we can identify all pendant blobs from these splits, by using the sl distance matrix. We can also replace one of these pendant blobs P by a leaf z to obtain a smaller level-2 network N', for which its shortest and longest inter-taxa distances can be obtained by Lemma 74. By induction hypothesis, N' is reconstructible. Then, we can obtain a network isomorphic to N by replacing the leaf z with the pendant blob that was originally present.

To see that this network is unique, consider another network M that is not isomorphic to N such that M induces the same sl-distance matrix as N. Note that M must also contain a pendant blob P, and upon replacing P in M by a leaf z, we get by the induction hypothesis that the resulting network M' must be isomorphic to N'. We obtain a network isomorphic to M by replacing the leaf z by P in M': but this operation yields a network that is also isomorphic to N. It follows that N and M must be isomorphic.

Therefore, level-2 networks are reconstructible from their sl-distance matrices. □

As stated before, it is possible to distinguish all pendant blobs from the shortest distances matrices if the networks do not contain the bad blobs. It follows then that the proof of Theorem 28 can be adapted to prove the following corollary, when we look at restricted level-2 networks.

Corollary 13. *Let N be a level-2 network containing no bad blobs. Then N is reconstructible, within the class of level-2 networks, from its shortest distance matrix.*

A direct consequence of Theorem 28 and Corollary 13, for restricted level-2 networks, is that by iteratively reducing pendant subtrees and pendant blobs from a network, it is possible to reconstruct the network from its sl-distance matrix and shortest distance matrix, respectively. Note that subtree reduction may be necessary after a few iterations of reducing pendant blobs from a network, as it is possible to obtain cherries from such reductions. Therefore the above results implicitly give an algorithm for reconstructing level-2 networks from their sl-distance matrices.

Completely excluding all bad blobs is quite restrictive. There can indeed exist networks that contain bad blobs that are still reconstructible from their shortest distances. For example, take a network in which there is exactly one bad blob. By reducing all cherries and all other pendant blobs before we reduce the bad blob, we are able to obtain a network on a single blob (which is necessarily the bad blob). Since networks on single blobs are reconstructible by Lemma 69, it follows then that the original network is also reconstructible. Therefore, in an effort to weaken the restriction of completely disallowing bad blobs, we next aim to characterize level-2 networks that are not reconstructible from their shortest distances.

7.5. CHARACTERIZATION OF LEVEL-2 NETWORKS THAT CANNOT BE RECONSTRUCTED FROM THEIR SHORTEST DISTANCES

In this section we show that level-2 networks that cannot be reconstructed from their shortest distances can be categorized by a type of subgraph that they must contain. We only consider shortest distances in this section; we use use $d^N(x,y)$ to denote the shortest distance between two vertices x and y in a network N.

7.5.1. ALT-PATH STRUCTURES

Let T be any binary tree with labelled leaves. Two-color the vertices of T with colors black and red. Let G denote a graph obtained by

- replacing each black internal vertex by a certain level-2 blob. That is, for each internal vertex v with neighbours u_i for $i \in [3]$, delete v, add vertices v_i, n_v, s_v and edges $u_i v_i, n_v v_i, s_v v_i$ for $i \in [3]$;
- replacing each black leaf by a pendant level-2 blob of the form $(2,0,0,0)$ or $(3,0,0,0)$; and
- replacing each red leaf by a pendant level-1 blob with two or three leaves.

The leaves of G are unlabelled. We call G an *alt-path structure* of T. We may obtain another alt-path structure H of T by swapping the roles of the red and black vertices in the blob replacement step. We say that H is *similar* to G if every pendant blob of H that replaces a leaf l of T contains the same number s of leaves as that of G that replaces l. See Figure 7.7 for an example of obtaining two similar alt-path structures from the same binary tree. Note that every binary tree T on at least two leaves gives rise to exactly two alt-path structures, and these are similar to each other.

We say that a network *contains* an alt-path structure of some tree if the alt-path structure is a subgraph of the network up to deleting leaf labels. Suppose that N contains an alt-path structure G of some tree T.

We define the operation of *replacing* G by its similar alt-path structure. For a moment, consider labelling the leaves of G. Let H be the similar alt-path structure to G, and label the leaves of H, such that its leaves in a pendant blob have the same labels as the leaves in the pendant blob of G that replaced the same leaf of T. Label the vertices of N that correspond to the leaves of G with the same labels, and delete the edges in G that are incident to the leaves of G. This operation creates two components. Take the component that contains the leaves of N, and append H by taking the union of the vertices (where vertices with the same label are counted as one vertex) and the union of the edges. Remove labels from non-leaf vertices to obtain a network N'. We call N' the network obtained by *replacing* G by its similar alt-path structure.

Lemma 75. *Similar alt-path structures of a given binary tree realise the same shortest distance matrix.*

Proof. Let N be a level-2 network that is an alt-path structure G of some binary tree T. Let N' be a network obtained from N by replacing G by its similar alt-path structure.

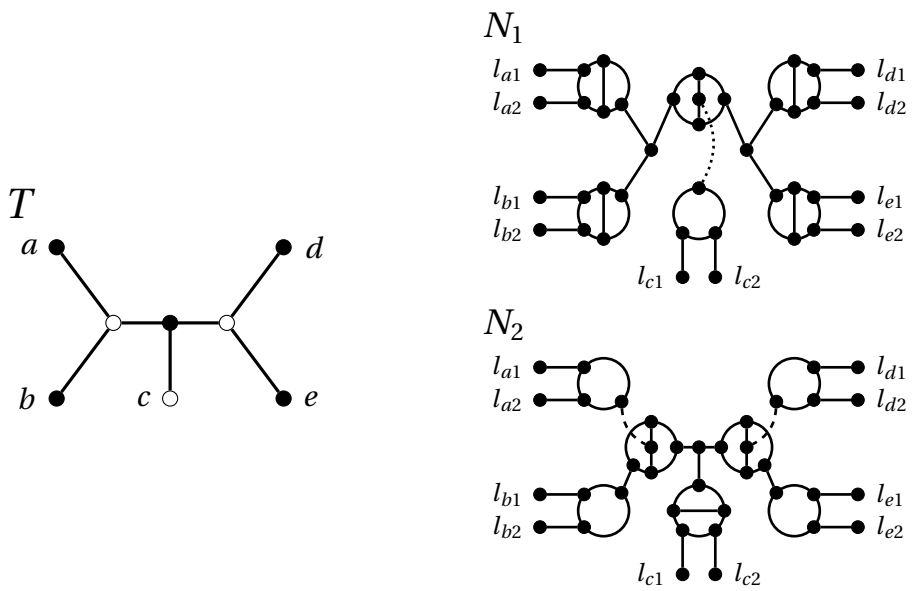


Figure 7.7: An example of obtaining two alt-path structures from a binary tree T. The network N_1 is obtained by replacing all filled internal vertices by a level-2 blob with each cut-edge subdividing the three different sides, filled leaf vertices by a pendant level-2 blob of the form $(k,0,0,0)$ for $k=2$ (note that this can also be $k=3$), and unfilled leaf vertices by a pendant level-1 blob of 2 or 3 leaves. The similar alt-path structure N_2 is obtained by the same construction, with the roles of filled and unfilled vertices reversed. Observe that N_1 and N_2 has the same shortest distance matrix, as stated in Lemma 75.

Let x and y denote two leaves in N. The two networks N and N' both contain the same chains by construction. Furthermore, each chain is of length at most 3. Thus we have that $d^N(x,y) = d^{N'}(x,y)$ if x and y are contained in the same chains. So we may assume that x and y are contained in different chains. We wish to show that $d^N(x,y) = d^{N'}(x,y)$. Consider the leaves l_x and l_y of T that were replaced by the pendant blobs containing x and y, respectively. If $d_T(l_x,l_y)$ is odd, then there is an even number of internal vertices in the path between l_x and l_y in T. This means that N and N' contain the same number of non-pendant level-2 blobs and the same number of non-leaf vertices not contained in blobs in the shortest path between x and y. Moreover, due to parity, exactly one of the two leaves will be contained in a pendant level-1 blob in N and the other leaf in a pendant level-2 blob. The reverse is true for N'. Thus it follows that if $d_T(l_x,l_y)$ is odd, then $d^N(x,y) = d^{N'}(x,y)$. Now if $d_T(l_x,l_y)$ is even, the number of non-pendant level-2 blobs in the shortest path between x and y will be greater by one in either N or in N'. Without loss of generality, suppose that N has this property. But this difference is offset by the fact that the pendant blobs in this path are both level-1 in N, whereas they are both level-2 in N'. Therefore $d^N(x,y) = d^{N'}(x,y)$. □

Corollary 14. *Let N be a level-2 network containing an alt-path structure G of some binary tree T as a subgraph. Then N is not reconstructible from its shortest distance matrix.*

Proof. Let N' denote the network obtained from N by replacing G by its similar alt-path structure. We claim that the distinct networks N and N' must realise the same shortest distance matrix, thereby proving that N is not reconstructible from its shortest distance matrix. Consider any two leaves x and y of N, and let P denote a shortest path between x and y in N. If P does not contain any edges of G, then a path between x and y on the same length must exist in N', since only the subgraph G of N was changed to obtain N'. On the other hand, if P contains an edge of G, then P must contain exactly one path of G that starts and ends at two leaves l_1, l_2 of G. Since only the subgraph G of N was changed to obtain N', we have that $d^N(x,l_1) = d^{N'}(x,l_1)$, and that $d^N(l_2,y) = d^{N'}(l_2,y)$. By Lemma 75, we have that $d^N(l_1,l_2) = d^{N'}(l_1,l_2)$. So a path between x and y on the same length must also exist in N'. It follows that $d^{N'}(x,y) \leq d^N(x,y)$.

Now consider a shortest path Q between x and y in N'. By applying the same arguments to Q, but with the alt-path structure that is similar to G, we conclude that $d^N(x,y) \leq d^{N'}(x,y)$. This proves that $d^N(x,y) = d^{N'}(x,y)$. □

It is now easy to explain why the two networks in Figure 7.1 realise the same shortest distance matrix; they contain similar alt-path structures of a binary tree on two leaves. We show in the next subsection that the converse of Corollary 14, that a level-2 network containing no alt-path structure is reconstructible, is also true.

7.5.2. LEVEL-2 NETWORKS WITHOUT ALT-PATH STRUCTURES ARE RECONSTRUCTIBLE

We introduce some more terminology. Let N be a level-2 network. A *blob tree* of N is the graph obtained by contracting all edges of blobs, deleting all labelled leaves, and suppressing all degree-2 vertices. A vertex of a blob-tree is called a *blob-vertex*. We define the *connection* of a pendant blob as the endpoint of the non-trivial cut-edge incident to

the blob that is not on the blob. We say that a blob B *contains* a pendant blob C if the connection of C is a vertex of B. For any blob B in N, we let $l(B)$ denote the level of B.

Let P_1 be a bad pendant blob on two leaves l_1, l_2, and let u be a vertex of N that is not a neighbour of l_1 nor l_2. We let $d^N(P_1, u) = d^N(l_1, u)$ denote the shortest distance between P_1 and u. This is well-defined for all bad pendant blobs containing exactly two leaves, since the shortest distance from either of the two leaves to any other vertex in the network is the same. Let P_2 be another bad pendant blob on two leaves l_1', l_2'. Then we may similarly define the shortest distance between P_1 and P_2 by $d^N(P_1, P_2) = d^N(l_1, l_1')$. This again is well-defined as the two bad pendant blobs both contain two leaves.

Let P_1 and P_2 be two pendant blobs that are contained in the same blob B. Let p_1 and p_2 be the connections of P_1 and P_2, respectively. We say that P_1 and P_2 are *adjacent* if p_1 and p_2 are adjacent. Let l be a leaf that is not contained in P_1. We say that l and P_1 are *adjacent* if the neighbour of l is adjacent to p_1. We say that P_1 is *adjacent* to a chain of leaves (a, k) if P_1 is adjacent to an end-leaf of (a, k).

Lemma 76. *Let N be a level-2 network on X containing a bad pendant blob with 3 leaves (a_1, a_2, a_3), and let N' denote the network obtained by deleting a_2 from N. Then the shortest distance matrix realized by N' is given by*

$$d^{N'}(x, y) = \begin{cases} d^N(x, y) & \text{if } x, y \in X - \{a_2\} \text{ and } \{x, y\} \neq \{a_1, a_3\}; \\ 3 & \text{if } \{x, y\} = \{a_1, a_3\}. \end{cases}$$

Proof. The only shortest paths containing the edges incident to the neighbour of a_2 in N were those involving a_2, or the path between a_1 and a_3. Since a_2 is no longer a leaf in N', the only path that is affected in the leaf deletion is the path between a_1 and a_3, which is now of length 3 in N'. □

We are now ready to prove the main theorem of the section. Because the proof exhaustively checks for contradictions within each case, it is rather long, and so we split the three main cases of the proof into subsubsections. In each subsubsection, a short summary will be given to clarify the proof steps.

Theorem 29. *Let N be a level-2 network with no alt-path structure, and let N' be a level-2 network. Then N and N' realize the same shortest distance matrix if and only if they are isomorphic. That is, a level-2 network containing no alt-path structure is reconstructible from its shortest distances within the class of level-2 networks.*

Proof. Suppose for a contradiction that there exist distinct networks N and N' with no alt-path structures that realise the same shortest distance matrix. In particular, choose N and N' to be minimal counter-examples with respect to the size of the networks. This means that every pendant blob of N and N' must be a bad blob; indeed, all other pendant blobs are identifiable from the shortest distance matrix by Lemma 74, and thus can be reduced otherwise, allowing for smaller counter-examples to exist. We may further assume that all pendant bad blobs contain exactly two leaves, as otherwise we can find a smaller counter-example by calling Lemma 76. Finally, observe that if N contains a pendant level-1 blob with some leaves l_1, l_2, then N' must contain a pendant level-2 blob of the form $(2, 0, 0, 0)$ containing the leaves l_1, l_2, as again we would be able to obtain a

smaller counter-example otherwise. Similarly, if N contains a pendant level-2 blob of the form $(2,0,0,0)$ with leaves l_1, l_2 then N' must contain a pendant level-1 blob with leaves l_1, l_2. If N contains a pendant blob P_i, we say that N' contains the *corresponding* pendant blob P'_i on the same leaves such that $l(P_i) \neq l(P'_i)$. For each pendant blob P_i, P'_i in N, N', we let p_i, p'_i denote their connections, respectively.

Since N and N' realize the same shortest distance matrix, the two networks have the same cut-edge induced splits by Theorem 27. Since each cut-edge in our networks induces a unique split, this implies that their blob-trees must be identical. This follows as every edge of the blob tree is a cut-edge of the network, and because trees are uniquely determined by their induced splits [11]. Note that the blob-vertices of the blob-trees correspond to either a degree-3 vertex, a level-1 blob, or a level-2 blob of the network. Observe that the blob-tree of N must contain at least 3 blob-vertices, as otherwise N would be a level-2 network with a single blob – which is reconstructible from their shortest distances by Lemma 69 – or a level-2 network with two pendant blobs, which implies that N must contain an alt-path structure as N contains only bad pendant blobs, or N and N' cannot realize the same shortest distance matrix.

Consider the blob-vertex u whose neighbours are all leaves, except possibly for one neighbour; let uv denote the edge in the blob-tree to this one neighbour. Since every edge of the blob-tree corresponds to a non-trivial cut-edge in the network, the edge uv must be incident to a degree-3 vertex / blob which correspond to u in N and N'. Let B, B' denote the corresponding structures in N, N', respectively. Then B contains at least one pendant blob, possibly some chain of leaves, and the cut-edge uv incident to it, with u as a vertex of B. The same can be said for B', but we use u' instead of u to be the vertex of B' incident to the cut-edge for clarity. Note that it is possible for u and u' to be a connection of some pendant blob. Let $\bar{B}$ be the graph obtained from N by deleting the edge uv and taking the component containing u. Similarly define $\bar{B}'$ as the graph obtained from N' by deleting the edge $u'v$ and taking the component containing u'. Since we have deleted the edges corresponding to the same edge in the blob-tree, $\bar{B}$ and $\bar{B}'$ contain the same chains, and $\bar{B}$ contains a pendant blob if and only if $\bar{B}'$ contains the corresponding pendant blob. Observe that $\bar{B}$ and $\bar{B}'$ are not networks because they contain a degree-2 vertex u and u', respectively. However, to avoid having to introduce new notation, we shall still use terms defined for networks, such as blobs containing pendant blobs, pendant blobs being adjacent to one another in $\bar{B}$ and $\bar{B}'$.

The rest of the proof will be as follows. We consider the cases where B is a degree-3 vertex, a level-1 blob, or a level-2 blob. Based on the graph $\bar{B}$ in comparison with the graph $\bar{B}'$, we seek a contradiction with regards to the networks realizing the shortest distance matrix and the choice of the minimum counter-example. Because the results are symmetric, it is worth mentioning that once we have proven the case for when B is a degree-3 vertex, then we may assume that B' is also not a degree-3 vertex. After we prove the case for when B is a level-1 blob, then we may assume that B' is also not a level-1 blob.

The following claim will be used extensively throughout the proof.

Claim 8. *Let x, y be leaves in $\bar{B}$. Then*

$$d^N(x,u) - d^{N'}(x,u') = d^N(y,u) - d^{N'}(y,u').$$

Proof. Let z be a leaf of N that is not in $\bar{B}$. Such a leaf must exist by our choice of B, and in particular, z must be reachable from B via uv. Then we have

$$d^N(z,x) = d^N(z,u) + d^N(u,x)$$
$$d^N(z,y) = d^N(z,u) + d^N(u,y),$$

which gives

$$d^N(z,x) - d^N(z,y) = d^N(x,u) - d^N(y,u).$$

Similarly we have

$$d^{N'}(z,x) - d^{N'}(z,y) = d^{N'}(x,u') - d^{N'}(y,u').$$

Since the shortest distance matrices of N and N' are the same, we have

$$d^N(x,u) - d^{N'}(x,u') = d^N(y,u) - d^{N'}(y,u').$$

□

***B* IS A DEGREE-3 VERTEX IN *N*:**

Suppose first that a leaf l is a neighbour of u in $\bar{B}$, and let P_1 be a pendant blob in $\bar{B}$ whose connection is u. Since N' has the same cut-edge induced splits, $\bar{B}'$ must either be a degree-3 vertex or a blob that contains l and the corresponding pendant blob P_1'. But then

$$d^N(P_1, l) = 4,$$

whereas

$$d^{N'}(P_1', l) \geq 5,$$

which contradicts the fact that N and N' must realize the same shortest distance matrix.

So now suppose that u is the connection of two pendant blobs P_1 and P_2 in $\bar{B}$. We check the three possible scenarios with regards to the levels of P_1 and P_2.

1. **$l(P_1) = 1$ and $l(P_2) = 1$:** Then we have $d^N(P_1, P_2) = 6$. But since $l(P_1') = l(P_2') = 2$, we must have that $d^{N'}(P_1', P_2') \geq 8$. This contradicts the fact that N and N' realize the same shortest distance matrix.

2. **$l(P_1) = 1$ and $l(P_2) = 2$:** Then we have $d^N(P_1, P_2) = 7$. Since $l(P_1') = 2$ and $l(P_2') = 1$, we have that $d^{N'}(P_1', P_2') \geq 7$, where equality is achieved whenever u' is a degree-3 vertex. But this would mean that

   $$d^N(P_1, u) - d^{N'}(P_1', u') = -1,$$

   whereas

   $$d^N(P_2, u) - d^{N'}(P_2', u') = 1,$$

   which contradicts Claim 8.

3. **$l(P_1) = 2$ and $l(P_2) = 2$ (see Figure 7.8 for an illustration of the cases):** Then $d^N(P_1, P_2) = 8$. Since $l(P'_1) = l(P'_2) = 1$, and because N' contains a split with one of the sets containing exactly the leaves of P'_1 and P'_2, B' must be a level-2 blob. In particular, P'_1 and P'_2 cannot be adjacent. There are two possibilities for this – u' is a neighbour of one of p'_1 or p'_2 but not the other, or all three vertices u', p'_1, p'_2 are pairwise non-adjacent (i.e., they all lie on different main sides of $\bar{B}'$). In the former case, we have that, assuming without loss of generality that u' is adjacent to p'_1,

$$d^N(P_1, u) - d^{N'}(P'_1, u') = 4 - 4 = 0,$$

whereas

$$d^N(P_2, u) - d^{N'}(P'_2, u') = 4 - 5 = -1,$$

which contradicts Claim 8. For the latter case, we claim that there is a smaller counter-example. We replace $\bar{B}$ in N by a pendant level-1 blob P_3 containing two leaves l_1, l_2. We replace $\bar{B}'$ in N' by a pendant level-2 blob P'_3 of the form $(2,0,0,0)$ with the same leaves l_1, l_2. Then, we may adjust the shortest distance matrix by first deleting elements containing the leaves of P_1 and P_2. And for all other leaves z in the network, we add the elements

$$d^N(l_i, z) = d^N(P_1, z) - 2,$$

and

$$d^{N'}(l_i, z) = d^N(P'_1, z) - 2$$

for $i = 1, 2$. All other matrix elements remain the same. Note that before the replacement of the blobs,

$$d^N(P_1, u) = d^N(P_2, u) = 4,$$

and

$$d^{N'}(P'_1, u') = d^{N'}(P'_2, u') = 5.$$

The replacement of $\bar{B}$ and $\bar{B}'$ by pendant level-1 and level-2 blobs, respectively ensures that the distance differences are preserved. Therefore the modified networks both must satisfy this new reduced shortest distance matrix. These modified networks N and N' still contain no alt-path structures, as otherwise the original networks also must have contained alt-path structures; all other parts of the networks remain unchanged, and the two leaves l_1 and l_2 are contained in pendant blobs of different level in N and N'. Therefore, this gives a counter-example on fewer leaves than that of N and N', contradicting our original choice of N and N'.

B IS A LEVEL-1 BLOB IN N:

We may now also assume that B' is either a level-1 or a level-2 blob. Note that either $\bar{B}$ or $\bar{B}'$ must contain a pendant level-1 blob, and that we shall obtain a contradiction in each of those cases. Before we do so, we prove a claim that will be used in many of the arguments to come. In the following claim, we assume that B is either a level-1 or a level-2 blob.

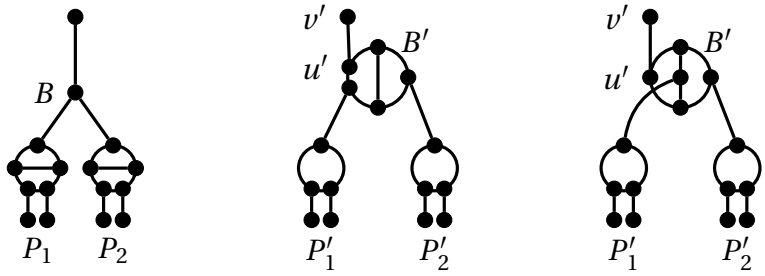


Figure 7.8: Subsubsection 7.5.2 case 3 in the proof of Theorem 29. The left figure is the part of the network N containing the internal vertex B and its neighbouring pendant blobs. The middle and the right figures are the two subcases for what N' could look like. In the middle network, two cut-edges are incident to the same side; in the right network, the three cut-edges are incident to distinct sides of B'.

Claim 9. *Suppose $l(B), l(B') \in \{1,2\}$, and suppose that $\bar{B}$ contains a pendant level-1 blob P_1. Then,*

(i) P_1 cannot be adjacent to a chain of leaves (a,k) in $\bar{B}$;

(ii) P_1 cannot be adjacent to another pendant level-1 blob in $\bar{B}$;

(iii) P_1 is adjacent to a pendant level-2 blob P_2 in $\bar{B}$ if and only if P'_1 and P'_2 are adjacent in $\bar{B}'$;

(iv) P_1 is adjacent to at most one pendant level-2 blob in $\bar{B}$. In particular, this means that every pendant level-2 blob in $\bar{B}$ is adjacent to at most one pendant level-1 blob.

(v) P_1 can be shortest distance 6 away from at most two end-leaves of distinct chains in $\bar{B}$.

Proof. (i) If P_1 is adjacent to a chain of leaves (a,k), then one of the end-leaves of (a,k) must be shortest distance 5 away from P_1. Without loss of generality, suppose that $d^N(P_1,a_1) = 5$. In B', since $l(P') = 2$, we must have that $d^{N'}(P'_1,a_1) \geq 6$, which contradicts the fact that N and N' must realize the same shortest distance matrix.

(ii) If P_1 is adjacent to another pendant level-1 blob P_2, then $d^N(P_1,P_2) = 7$. In $\bar{B}'$, both corresponding pendant blobs are of level-2. This means that $d^{N'}(P'_1,P'_2) \geq 8$, which contradicts the fact that N and N' must satisfy the same shortest distance matrix.

(iii) If P_1 is adjacent to P_2, then we have $d^N(P_1,P_2) = 8$. Since N and N' satisfy the same shortest distance matrix, one must also have $d^{N'}(P'_1,P'_2) = 8$. The shortest distance from P'_1 to its connection p'_1, and that from P'_2 to its connection p'_2 is 3 and 4, respectively. Since $l(B') \in \{1,2\}$, the vertices p'_1 and p'_2 cannot be the same. Therefore to satisfy P'_1 and P'_2 being shortest distance 8 from one another in $\bar{B}'$, one must have that p'_1 and p'_2 are adjacent, which means that P'_1 and P'_2 must be adjacent. The converse follows by symmetry.

(iv) Suppose for a contradiction that P_1 is adjacent to two pendant level-2 blobs P_2 and P_3 in $\bar{B}$. Then $d^N(P_2, P_3) = 10$. By 3., we know that P_1' must be adjacent to both P_2' and P_3' in $\bar{B}'$. We also know that $l(P_2') = l(P_3') = 1$. It follows that $d^{N'}(P_2', P_3') = 8$. But this contradicts the fact that N and N' must realize the same shortest distance matrix. If a pendant level-2 blob is adjacent to two pendant level-1 blobs, then the corresponding pendant level-1 blob in $\bar{B}'$ is adjacent to two pendant level-2 blobs, which we have just shown cannot be true.

(v) Suppose l is an end-leaf of a chain such that $d^N(P_1, l) = 6$. Since $l(P_1') = 2$, and since N and N' realize the same shortest distance matrix, the leaf l must be adjacent to P_1' in $\bar{B}'$. So every leaf that is shortest distance 6 away from P_1 in $\bar{B}$ must be adjacent to P_1' in $\bar{B}'$. Note that P_1' can be adjacent to at most two chains. These chains must be distinct in $\bar{B}'$, since u' is contained in $\bar{B}'$. This implies that in $\bar{B}$, P_1 can be shortest distance 6 away from at most two end-leaves of distinct chains.

□

It follows that each main side of $\bar{B}$ (and $\bar{B}'$) may contain at most two pendant level-1 blobs. Either $\bar{B}$ or $\bar{B}'$ must contain a pendant level-1 blob.

1. **$\bar{B}$ has a pendant level-1 blob P_1:** Since N contains no parallel edges, and since leaves cannot be adjacent to pendant level-1 blobs, P_1 must be adjacent to a pendant level-2 blob P_2 in $\bar{B}$. By Claim 9 (iv), P_1 can be adjacent to at most one pendant level-2 blob in $\bar{B}$. This implies that p_1 must be adjacent to u. In N', the corresponding pendant blobs P_1' and P_2' are adjacent by Claim 9 (iii). Then we have that

$$d^N(P_1, u) - d^N(P_2, u) \leq 4 - 5 = -1.$$

Observe that $d^{N'}(P_1', p_1') = 4$ and $d^{N'}(P_2', p_1') = 4$ since $l(P_1') = 2$ and $l(P_2') = 1$. This implies that

$$\begin{aligned} d^{N'}(P_1', u') - d^{N'}(P_2', u') &\geq d^{N'}(P_1', p_1') + d^{N'}(p_1', u') - d^{N'}(P_2', p_1') - d^{N'}(p_1', u') \\ &= 4 - 4 \\ &= 0, \end{aligned}$$

which is a contradiction to Claim 8.

2. **$\bar{B}'$ has a pendant level-1 blob P_1':** If $l(B') = 1$, then we are done by symmetry via case 1. So suppose that $l(B') = 2$, and suppose in addition that $\bar{B}$ contains no pendant level-1 blobs. This implies that $\bar{B}'$ contains no pendant level-2 blobs. We claim that $\bar{B}'$ also contains no pendant level-1 blobs other than P_1'. Suppose for a contradiction that it did, so that $\bar{B}'$ contains another pendant level-1 blob P_2'. Because $\bar{B}'$ contains no pendant level-2 blobs, and since leaves cannot be adjacent to pendant level-1 blobs by Claim 9, we must have that p_1' and p_2' are adjacent to the same pole, or that they must both be adjacent to u'. In any case, we must have $d^{N'}(P_1', P_2') = 8$. But $l(P_1) = l(P_2) = 2$, and therefore $d^N(P_1, P_2) \geq 9$, which is a contradiction. So P_1' is the only pendant blob in $\bar{B}'$.

We now consider two possible cases: either p'_1 is or is not adjacent to u'. Either way, at least one main side of B' that does not contain p'_1 must contain a chain of leaves, as otherwise B' contains parallel edges, or B' is a level-2 blob with only two cut-edges incident to it.

(a) ***p'_1 is adjacent to u':*** One of the main sides of $\bar{B}'$ must contain a chain, since N' does not contain parallel edges. So there must exist a leaf l that is shortest distance 6 away from P'_1. Then

$$d^{N'}(P'_1, u') - d^{N'}(l, u') \leq 4 - 3 = 1.$$

On the other hand, in $\bar{B}$, the leaf l is adjacent to P_1; then $d^N(P_1, p_1) = 4$ and $d^N(l, p_1) = 2$. It follows that

$$\begin{aligned} d^N(P_1, u) - d^N(l, u) &\geq d^N(P_1, p_1) + d^N(p_1, u) - d^N(l, p_1) - d^N(p_1, u) \\ &= 2, \end{aligned}$$

which contradicts Claim 8.

(b) ***p'_1 is not adjacent to u':*** Observe that $\bar{B}'$ has five sides: two sides s'_1, s'_2 which have p'_1 as one of their boundary vertices; two sides s'_3, s'_4 which have u' as one of their boundary vertices; and one side s'_5 that has neither p'_1 nor u' as a boundary vertex. By Claim 9 (i) and (v), the sides s'_1 and s'_2 are empty, and at most two of the three remaining sides of $\bar{B}'$ may contain chains. In particular, at least one of these remaining three sides must contain a chain. We first show that s'_5 must be empty. Suppose not, and let (a, k) denote the chain contained in s'_5. Note that both end-leaves of (a, k) are shortest distance 6 away from P'_1, and so by Claim 9 (v), this chain must be of length $k = 1$. Since either s'_3 or s'_4 must be empty,

$$d^{N'}(a_1, u') - d^{N'}(P'_1, u') = 3 - 5 = -2.$$

In $\bar{B}$, the blob P_1 and the leaf a_1 must be adjacent. The vertex u must be adjacent to the neighbour of a_1, since $\bar{B}$ contains no other pendant blobs. So we have

$$d^N(a_1, u) - d^N(P_1, u) \leq 2 - 5 = -3,$$

which contradicts Claim 8. So s'_5 is empty.

Now suppose that s'_3 and s'_4 contain the chains (b, ℓ) and (c, m), respectively, where at least one of $\ell \geq 1$ or $m \geq 1$ holds. If $\ell = 1$ and $m = 0$, then $\bar{B}'$, together with the edge incident to u' is an alt-path structure of a binary tree on two leaves. This contradicts our choice of N'. By symmetry, the case $m = 1$ and $\ell = 0$ is also not possible. So we may assume that $\ell \geq 1$ and $m \geq 1$. Suppose the chains are arranged such that

$$d^{N'}(b_1, u') = d^{N'}(c_1, u') = 2.$$

If $\ell > 1$ or $m > 1$, then $d^{N'}(b_\ell, c_m) = 5$, whereas $d^N(b_\ell, c_m) = 4$. This contradicts the fact that N and N' realize the same shortest distance matrix. So we

must have $\ell = m = 1$. But then $\bar{B}'$, together with the edge incident to u' is an alt-path structure of a binary tree on two leaves. This contradicts our choice of N'.

***B* IS A LEVEL-2 BLOB IN *N*:**
Our only remaining case is if B and B' are both level-2 blobs. The proofs of Claims 10-12 are given in the next section.

Claim 10. *Two pendant level-2 blobs cannot be adjacent to one another in $\bar{B}$ and in $\bar{B}'$.*

An immediate consequence of Claim 10 is that distinct pendant level-1 blobs in $\bar{B}$ or $\bar{B}'$ must be distance at least 10 apart. In particular, they cannot be adjacent by Claim 9 and they cannot be shortest distance-8 apart, since two pendant level-2 blobs are shortest distance at least 9 apart. The following claim dictates the placement of pendant blobs and leaves in $\bar{B}$.

Claim 11. *A pendant level-2 blob may not be adjacent to both a pendant level-1 blob and a leaf simultaneously in $\bar{B}$ and in $\bar{B}'$.*

Pendant level-1 blobs may be adjacent to at most one pendant level-2 blob by Claim 9. Pendant level-2 blobs cannot be adjacent to other pendant level-2 blobs by Claim 10. Pendant level-1 blobs cannot be adjacent to a leaf by Claim 11. So the main side of $\bar{B}$ (and $\bar{B}'$) that contains u (u') contains at most two pendant level-1 blobs; the other two main sides of $\bar{B}$ (and $\bar{B}'$) contain at most one pendant level-1 blob.

Claim 12. *$\bar{B}$ and $\bar{B}'$ contain at most one pendant level-1 blob.*

We have now arrived at the two final cases for this proof. In summary, the current setting is as follows. Both $\bar{B}$ and $\bar{B}'$ are level-2 blobs, and they both contain at most one pendant level-1 blob. In fact, this implies that $\bar{B}$ and $\bar{B}'$ also contain at most one pendant level-2 blob. We split into the cases for when $\bar{B}$ does not, or does contain a pendant level-2 blob.

1. ***$\bar{B}$* contains no pendant level-2 blob:** By assumption, $\bar{B}$ must contain a pendant level-1 blob P_1. Let s denote the main side of B containing P_1. Note that s may contain at most one chain. Indeed, P_1 cannot be adjacent to a chain of leaves by Claim 9 (i), so p_1 must be adjacent to a pole of $\bar{B}$; if p_1 is adjacent to u, then s can contain a chain of leaves (a,k) such that an end-spine vertex of (a,k) is adjacent to u. The other two main sides of $\bar{B}$ may contain at most one chain of leaves each. So in total, $\bar{B}$ may contain at most three chains.

   For each chain contained in $\bar{B}$, we have that one end-leaf of a chain is shortest distance-6 from P_1. This means that each chain contained in $\bar{B}$ must be adjacent to P_1' in $\bar{B}'$. But P_1' may be adjacent to at most two chains. So $\bar{B}$ may contain up to two chains. This also implies that $\bar{B}'$ only contains leaves on the main side that contains p_1'. Since $\bar{B}'$ contains no parallel edges, we must then have that u' lies on a main side that does not contain p_1'.

   Note that $\bar{B}$ must contain at least one chain, as otherwise B would be a level-2 blob incident only to two cut-edges. We now split into subcases depending on the location of u.

(a) ***u* is adjacent to p_1:** Then $d^N(P_1,u)=4$. Since u' is not on the same main side as that containing P_1', we have $d^{N'}(P_1',u')\geq 6$. Now there exists a leaf l such that $d^{N'}(l,u')=3$. Noting that $d^N(l,u)\geq 2$, we have

$$d^N(P_1,u)-d^N(l,u)\leq 2$$

and

$$d^{N'}(P_1',u')-d^{N'}(l,u')\geq 3,$$

which contradicts Claim 8.

(b) ***u* is not adjacent to p_1:** We let s,s_1,s_2 denote the three main sides of $\bar{B}$ such that s contains p_1, s_1 contains u, and s_2 contains neither p_1 nor u. We claim first that s_2 contains no chains. Suppose for a contradiction that it did contain some chain (a,k). Note first that a_1 and a_k are both shortest distance 6 from P_1; this implies that P_1' is adjacent to both a_1 and a_k in $\bar{B}'$, implying that a_1 and a_k are in different chains in $\bar{B}'$. But this is not possible, so we require $k=1$. Note also that since $\bar{B}$ contains at most two chains, u must be adjacent to a pole; this implies that $d^N(P_1,u)=5$ and $d^N(a_1,u)=3$. However in B', we have $d^{N'}(P_1',u')\geq 6$ and $d_{N'}(a_1,u')=3$, which contradicts Claim 8 as

$$d^N(P_1,u)-d^N(a_1,u)=5-3=2$$

whereas

$$d^{N'}(P_1',u')-d^{N'}(a_1,u')\geq 6-3=3.$$

So the main side s_2 contains no chains; this leaves only s_1 to contain chains. The main side s_1 may contain two chains, (b,ℓ) and (c,m), such that $\ell,m\geq 0$ and $d^N(b_1,u)=d^N(c_1,u)=2$, whenever $\ell>0$ and $m>0$, respectively. We require $\ell+m\geq 3$, as otherwise $\bar{B}$ with the edge incident to u is an alt-path structure that can be obtained from a binary tree on two leaves. We fall into two subcases depending on the value of ℓ.

i. **$\ell=0$:** Then $m\geq 3$, and so

$$d^N(P_1,c_1)=7,$$

whereas

$$d^{N'}(P_1',c_1)=8,$$

which contradicts the fact that N and N' realize the same shortest distance matrix.

ii. **$\ell\neq 0$:** By symmetry, we may assume $m\neq 0$. Now,

$$d^N(b_1,c_1)=4,$$

whereas

$$d^{N'}(b_1,c_1)=5,$$

since $\ell+m\geq 3$. This contradicts the fact that N and N' realize the same shortest distance matrix.

2. **$\bar{B}$ contains one level-2 blob:** If $\bar{B}$ did not contain a pendant level-1 blob, then we are done by applying the arguments from the previous case to $\bar{B}'$. So suppose that $\bar{B}$ contains a pendant level-1 blob P_1 and a pendant level-2 blob P_2. We first show that P_1 and P_2 cannot be adjacent in $\bar{B}$. Suppose that they were adjacent. Then the vertex on $\bar{B}$ that is shortest distance-2 from p_1 must be a pole of $\bar{B}$ or u. Indeed, it cannot be a neighbour of some leaf l; this would mean that $d^N(P_1, l) = 6$, implying that P_1' must be adjacent to l. But this is not possible by Claim 11. In particular, it cannot be a connection since $\bar{B}$ contains only the pendant blobs P_1 and P_2.

   Now, if u was adjacent to p_2, then this would mean that the two main sides of $\bar{B}$ would be empty, resulting in parallel edges in $\bar{B}$. So u must either be adjacent to p_1 or u must be contained in one of the two other main sides of $\bar{B}$. Either way, we have

$$d^N(P_1, u) - d^N(P_2, u) \leq -1.$$

   In $\bar{B}'$, we have $d^{N'}(P_1', p_1') = d^{N'}(P_2', p_1') = 4$. So

$$d^{N'}(P_1', u') - d^{N'}(P_2', u') \geq 0,$$

   which contradicts Claim 8. So we may assume that P_1 and P_2 are not adjacent in $\bar{B}$. We split into cases depending on the position of u in $\bar{B}$.

   (a) **u is on the same main side as p_1:** Then u must be adjacent to p_1, since P_1 cannot be adjacent to a chain by Claim 9, and since P_1 is not adjacent to the only pendant level-2 blob in $\bar{B}$. Then $d^N(P_1, u) = 4$ and $d^N(P_2, u) \geq 5$. We split into subcases depending on the position of u' in $\bar{B}'$.

      i. **u' is on the same main side as p_2':** Then u' must be adjacent to p_2'. So we have $d^{N'}(P_2', u') = 4$ and $d^{N'}(P_1', u') \geq 5$, which contradicts Claim 8.

      ii. **u' is not on the same main side as p_2':** If u' is adjacent to an end-spine vertex of some chain, then there exists a leaf l such that $d^{N'}(l, u') = 2$. We also have $d^{N'}(P_1', u') \geq 5$. But in $\bar{B}$, we have $d^N(l, u) \geq 2$. This contradicts Claim 8, as

$$d^N(P_1, u) - d^N(l, u) \leq 4 - 2 = 2,$$

      whereas

$$d^{N'}(P_1', u') - d^{N'}(l, u') \geq 5 - 2 = 3.$$

      So u' cannot be adjacent to an end-spine vertex of some chain, which means that $d^{N'}(P_2', u') = 5$. But $d^N(P_2, u) \geq 5$, and so

$$d^N(P_1, u) - d^N(P_2, u) \leq 4 - 5 = -1,$$

      whereas

$$d^{N'}(P_1', u') - d^{N'}(P_2', u') \geq 0,$$

      which contradicts Claim 8.

   (b) **u is not on the same main side as p_1:** We may assume that u' is not on the same main side as p_2' by symmetry (apply the previous case to $\bar{B}'$).

i. **u and p_2 are not on the same main side:** We let s_u, s_1, s_2 denote the three main sides of $\bar{B}$ such that s_u contains u and s_i contains p_i for $i = 1,2$. Note that s_u may contain up to two chains: denote these chains as (a,k) and (b,ℓ) where $d^N(a_1,u) = d^N(b_1,u) = 2$, if $k > 0$ and $\ell > 0$, respectively. If $k > 0$ and $\ell > 0$, then $d^N(P_2,a) \geq 7$ and $d^N(P_2,b) \geq 7$. Since $d^N(P_1,a_k) = d^N(P_1,b_\ell) = 6$, the pendant blob P'_1 must be adjacent to the two chains a and b in $\bar{B}'$. Depending on the placement of u', at least one, and at most two of the chain endpoints a_1, b_1 are shortest distance 6 away from P'_2. But this contradicts that N and N' realize the same shortest distance matrix. Therefore, s_u contains at most one chain; this means that $d^N(P_1,u) = 5$. We also have $d^N(P_2,u) \geq 6$.
The same argument can be used in the case when u' and p'_1 are not on the same main side of $\bar{B}'$. In that case, the main side of $\bar{B}'$ containing u' contains at most one chain. We now split into subcases depending on the position of u'.

   A. **u' and p'_1 are not on the same main side of $\bar{B}'$:** Then, the main side of $\bar{B}'$ that contains u' contains at most one chain. In particular, this means that u' must be adjacent to a pole in $\bar{B}'$. So $d^{N'}(P'_2,u') = 5$. Furthermore, $d^{N'}(P'_1,u') \geq 6$. But this contradicts Claim 8.

   B. **u' and p'_1 are on the same main side of $\bar{B}'$:** Consider the generator side s' of B' that contains u' and a pole of $\bar{B}'$ as its boundary vertices. We claim that s' is empty. If not, then s' contains a chain (a,k) such that $d^{N'}(a_1,u') = 2$. But then $d^{N'}(P'_2,a_k) = 6$, meaning that a_k must be adjacent to P_2 in B. This further implies that $d^N(P_1,a_1) = 6$, which leads to a contradiction as a_1 is clearly not adjacent to P'_1 in B' (we have $d^{N'}(P'_1,a_1) \geq 7$). Thus s' must be empty. But then

$$d^N(P_1,u) - d^N(P_2,u) \leq 5 - 6 = -1,$$

   whereas

$$d^{N'}(P'_1,u') - d^{N'}(P'_2,u') \geq 5 - 5 = 0,$$

   which contradicts Claim 8.

ii. **u and p_2 are on the same main side of $\bar{B}$:** We may assume also that u' and p'_1 are on the same main side of $\bar{B}'$. Consider the main side s of $\bar{B}$ that does not contain p_1 nor p_2. We claim that s is empty. Suppose not, and suppose that s contains a chain (a,k). In $\bar{B}'$, we require P'_1 to be adjacent to both a_1 and to a_k. For this to be possible, since (a,k) must also be a chain in $\bar{B}'$, we require $k = 1$. Note that in $\bar{B}'$, the leaf a_1 is contained in the side with boundary vertices p'_1 and u'. If it had been contained in the other side of $\bar{B}'$ with p'_1 as its boundary vertex, then P_2 must be adjacent to a_1 in B, which is clearly not the case. Observe that a shortest path from P_1 to u contains the same pole contained in a shortest path from a_1 to u. Noting that the shortest distance from P_1 to this pole, and the shortest distance from a_1 to this pole are 4 and 2, respectively, it

follows that

$$d^N(P_1,u)-d^N(a_1,u)=4-2=2,$$

whereas

$$d^{N'}(P_1',u')-d^{N'}(a_1,u')=6-2=4,$$

which contradicts Claim 8. So s is empty, and we may assume that the main side of $\bar{B}'$ that does not have p_1' nor p_2' is empty.
If all three other sides of B are empty, then B contains an alt-path structure formed by a binary tree on two leaves, and we get a contradiction on the choice of N. In particular, the three sides must contain at least two leaves. Let s_1,s_2,s_3 denote the sides of B that has p_2 but not u, p_2 and u, and u but not p_2 as its boundary vertices, respectively. Similarly let s_1',s_2',s_3' denote the sides of B' that has p_1' but not u', p_1' and u', and u but not p_1' as its boundary vertices, respectively. It is easy to see that a chain contained in s_1 must be contained in s_1'; a chain contained in s_2 must be contained in s_3'; a chain contained in s_3 must be contained in s_2'.

A. **the side s_1 is non-empty:** Let (a,k) denote the chain contained in s_1, such that $d^N(a_1,P_2)=d^N(a_k,P_1)=6$. We claim that s_2 and s_3 must both be empty. If s_2 is non-empty, then it contains a chain (b,ℓ), such that $d^N(b_1,P_2)=6$. So the chains (a,k) and (b,ℓ) are adjacent. In $\bar{B}'$, the chain (b,ℓ) is contained in the side s_3'. But then (a,k) and (b,ℓ) cannot be adjacent in N', which contradicts the fact that N and N' satisfy the same shortest distance matrix. So s_2 must be empty. By applying the same argument to B', we see that s_2' must also be empty; therefore, the side s_3 must be empty. So s_2 and s_3 must both be empty.
We require $k\geq 2$, as otherwise B contains an alt-path structure obtained from a tree on two leaves. But then

$$d^N(a_1,u)-d^N(a_k,u)=3-4=-1,$$

whereas

$$d^{N'}(a_1,u')-d^{N'}(a_k,u')=4-3=1,$$

since a_1 is adjacent to P_2 in $\bar{B}$ and a_k is adjacent to P_1' in $\bar{B}'$. This contradicts Claim 8.

B. **the side s_1 is empty:** Let (b,ℓ) and (c,m) denote the chains contained in s_2 and s_3, respectively, such that $d^N(P_2,b_1)=6$ and $d^N(b_\ell,c_m)=4$, whenever $\ell>0$ and $m>0$, respectively. Note that $\ell+m\geq 2$, as otherwise $\bar{B}$ together with the edge incident to u is an alt-path structure formed by a binary tree on two leaves. In particular, at least one of ℓ or m must be non-zero. By symmetry, we may assume without loss of generality that $m>0$. Then in $\bar{B}'$, the blob P_1' is adjacent to c_1. This means that

$$d^{N'}(P_2',c_1)=7.$$

In $\bar{B}$, there are three paths from c_1 to P_2. One uses the empty main side and is of length 8. The second uses the main side with p_1 and is of length 9. These two paths cannot be altered by deleting leaves. The third path contains the spine of the chain (b, ℓ), and is of length $\ell + m + 6$. Since $m \geq 1$, we only obtain $d^N(P_2, c_1) = 7$ if and only if $m = 1$ and $\ell = 0$. But this is not possible, since we require $\ell + m \geq 2$. So the networks N and N' cannot satisfy the same shortest distance matrix, which is a contradiction.

Therefore we reach a contradiction for the case when B and B' are both level-2 blobs. □

The following corollary follows immediately from Corollary 14 and Theorem 29.

Corollary 15. *A level-2 network is reconstructible if and only if it does not contain an alt-path structure.*

7.6. PROOF OF CLAIMS FROM SECTION 7.5.2:

To prove these claims, we will use the following observation.

Observation 20. *Let P_1 be a pendant level-1 blob contained in $\bar{B}$. Suppose that $p'_2p'_1p'_3p'_4$ is a path in $\bar{B}'$, such that each p'_i is a connection for a pendant blob P'_i for $i \in [4]$. Suppose also that $l(P'_1) = l(P'_3) = 1$ and $l(P'_2) = l(P'_4) = 2$. A vertex p on the blob $\bar{B}$ such that $d^N(p_1, p) = 2$ must either be u or a pole of $\bar{B}$.*

Proof. Suppose for a contradiction that p is either a neighbour of a leaf or a connection of some pendant blob. Suppose first that p is a neighbour of a leaf l. Then $d^N(P_1, l) = 6$. Since N and N' have the same shortest distance matrix, this means that P'_1 must be adjacent to l in $\bar{B}'$. But this is not possible as P'_1 is already adjacent to P'_2 and P'_3. So suppose that p is a connection of a pendant blob P_5. Suppose first that $l(P_5) = 1$. Then $d^N(P_1, P_5) = 8$. Since $l(P'_1) = l(P'_5) = 2$, such a distance can be realized in N' if and only if $p'_1 = p'_5$. But this is impossible as B' is a level-2 blob. Finally suppose that $l(P_5) = 2$. Then since $d^N(P_1, P_5) = 9$, the corresponding blob P'_5 must be adjacent to P'_2 or to P'_4, which is not possible as pendant level-1 blobs cannot be adjacent to one another by Claim 9. □

Claim 3. *Two pendant level-2 blobs cannot be adjacent to one another in $\bar{B}$ (and in $\bar{B}'$).*

Proof. Suppose for a contradiction that $\bar{B}$ contains two adjacent pendant level-2 blobs P_1 and P_2 on the main side s. Then B' contains two corresponding pendant level-1 blobs P'_1 and P'_2 on the same leaves. We split into cases depending on the locations of p'_1 and p'_2 in B'.

1. **p'_1 and p'_2 are on the same main side s' of B':** Since the shortest distance between p'_1 and p'_2 must be exactly 3, there must be at least two vertices on the same main side in the path Q' between p'_1 and p'_2. This path Q' may contain u' as a vertex; we split into cases again.

(a) ***u′ is a vertex of Q′:*** Observe first that if both P'_1 and P'_2 are adjacent to level-2 blobs P'_3 and P'_4 respectively, then $d^{N'}(P'_3, P'_4) \geq 10$. However, the counterparts of these blobs in $\bar{B}$ must be adjacent to P_1 and P_2. Since P_1 and P_2 are adjacent, this implies that $d^N(P_3, P_4) = 9$, which contradicts the fact that N and N' satisfy the same shortest distance matrix. Therefore only one of P'_1 or P'_2 can be adjacent to a pendant level-2 blob. Note that at least one of P'_1 or P'_2 must be adjacent to a pendant level-2 blob, such that its connection is on the path Q'. So without loss of generality, suppose that P'_1 is adjacent to a pendant level-2 blob P'_3, such that p'_3 is a vertex in Q'. At this point, we have that P_3, P_1 and P_1, P_2 are adjacent in $\bar{B}$.

We claim that P'_3 cannot be adjacent to a leaf or to pendant blobs other than P'_1 in $\bar{B}'$. Firstly, if P'_3 was adjacent to a leaf l, then $d^{N'}(P'_1, l) \in \{5, 6\}$. The distance $d^{N'}(P'_1, l) = 5$ is impossible as $l(B) = 2$; if $d^{N'}(P'_1, l) = 6$, then P_1 must be adjacent to l in $\bar{B}$. But this is impossible as P_1 is already adjacent to P_2 and P_3. This is a contradiction. The blob P'_3 cannot be adjacent to a pendant level-1 blob other than P'_1, as this would contradict Claim 9. Finally, we claim that P'_3 cannot be adjacent to a pendant level-2 blob P'_4. Since this would mean that $d^{N'}(P'_1, P'_4) = 9$, we must in $\bar{B}$ that either P_2 or P_3 is adjacent to P_4. The former is not possible as P'_2 is not adjacent to P'_4 in B'; the latter is not possible as two pendant level-1 blobs cannot be adjacent by Claim 9. So P'_3 cannot be adjacent to a leaf or to pendant blobs in $\bar{B}'$.

By Claim 9, the connection p'_1 must be adjacent to a pole of B'. Since P'_2 cannot be adjacent to a leaf or to a pendant level-1 blob by Claim 9, and it also cannot be adjacent to any pendant level-2 blobs by assumption, p'_2 must also be adjacent to the other pole of B' and to u'. Since p'_3 must be adjacent to u', it follows that the main side s' is the path $p'_1 p'_3 u' p'_2$.

Now B' must contain another leaf on one of the other two main sides since they cannot contain parallel edges. We claim that such a leaf cannot exist, thereby reaching a contradiction. Let l denote such a leaf that is on one of these two main sides, whose neighbour (if l is not in a pendant blob) / connection (if l is in a pendant blob) is shortest distance-2 to p'_1. Suppose first that l is not in a pendant blob. Then

$$d^{N'}(P'_1, l) = 6,$$

and so P_1 must be adjacent to l in $\bar{B}$, which is not possible as P_1 is already adjacent to P_2 and to P_3. So now suppose that l is contained in a pendant level-1 blob. Then

$$d^{N'}(P'_1, l) = 8.$$

But the level of the corresponding pendant blob in $\bar{B}$ is 2, and since $l(P_1) = 2$, we must have that $d^N(P'_1, l) \geq 9$, which contradicts the fact that N and N' have the same shortest distance matrix. Finally, if l is contained in a pendant level-2 blob P'_4, then

$$d^{N'}(P'_1, l) = 9.$$

This means that in $\bar{B}$, the corresponding blob P_4 must be adjacent either to P_2 or P_3. The former is not possible as P'_2 is not adjacent to P'_4 in $\bar{B}'$; the latter is not possible as two pendant level-1 blobs cannot be adjacent by Claim 9.

(b) ***u' is not a vertex of Q':*** Let p'_3 and p'_4 denote the neighbours of p'_1 and p'_2 in this path Q', respectively. Since $l(P'_1) = l(P'_2) = 1$, the vertices p'_3 and p'_4 must be connections of pendant level-2 blobs P'_3 and P'_4, respectively. At this point, we have that P_3, P_1; P_1, P_2; P_2, P_4 are adjacent in B. Now suppose that there is another vertex p'_5 that is a neighbour of p'_3 that is not p'_4 in B'. By Observation 20, the vertex p'_5 cannot be a neighbour of a leaf nor a connection of some pendant blob. By nature of B', p'_5 must be the vertex u', but this would contradict our assumption that the path Q' does not include u'. Therefore p'_3 must be adjacent to p'_4. Thus, P'_1, P'_3; P'_3, P'_4; and P'_4, P'_2 are adjacent in B'.

By Claim 9, since pendant level-1 blobs may not be adjacent to leaves and they may be adjacent to at most one pendant level-2 blob, either p'_1 or p'_2 must be adjacent to a pole of B'. Suppose without loss of generality that p'_1 is adjacent to a pole of B'. Consider the two main sides of B' that are not s'. Since N contains no parallel edges, one of the two main sides must contain a vertex. In particular, there must exist a vertex that is distance-2 away from p'_1. By Observation 20, such a vertex cannot be a neighbour of a leaf nor a connection of a pendant blob in $\bar{B}'$. Then such a vertex must be u'. By invoking Observation 20 again, for both p'_1 and p'_2, it is easy to see that these two main sides cannot contain any leaves in $\bar{B}'$. So $\bar{B}$ and $\bar{B}'$ must contain only the eight leaves of these four pendant blobs on the same main sides, with the vertices u and u' on a different main side, respectively. But we find that

$$d^N(P_1, u) - d^{N'}(P'_1, u') = 7 - 5 = 2,$$

whereas

$$d^N(P_3, u) - d^{N'}(P'_3, u') = 5 - 7 = -2,$$

which contradicts Claim 8.

2. ***p'_1 and p'_2 are incident to different main sides of B':*** Let s'_1, s'_2 denote the main sides of B' that contains p'_1, p'_2, respectively. Let s'_3 denote the third main side of $\bar{B}'$. The vertex u' is contained either in s'_1, s'_2, or in s'_3. The first two cases are equivalent by symmetry, so we split into two cases.

(a) ***u' is in s'_1:*** If P'_1 and P'_2 are both not adjacent to a pendant level-2 blob, then $d^{N'}(P'_1, P'_2) = 8$ and we reach a contradiction as we have $d^N(P_1, P_2) = 9$. Therefore P'_1 or P'_2 must be adjacent to a pendant level-2 blob on this distance-8 path. If P'_1 and P'_2 are both adjacent to level-2 blob P'_3 and P'_4, respectively, then $d^{N'}(P'_3, P'_4) \geq 10$. But since P_3, P_1; P_1, P_2; and P_2, P_4 would be adjacent in $\bar{B}$, we must have $d^N(P_3, P_4) = 9$, which contradicts the fact that N and N' satisfy the same shortest distance matrix. Therefore, exactly one of P'_1 or P'_2 must be adjacent to a pendant level-2 blob.

i. **P'_1 is adjacent to a pendant level-2 blob P'_3:** Note that p'_1 is adjacent to u' and to p'_3. In particular, P'_1 must be adjacent to u' to make sure that the shortest path between P'_1 and P'_2 is of length 9. Then we have

$$d^{N'}(P'_1, u') - d^{N'}(P'_3, u') = 4 - 6 = -2.$$

But in $\bar{B}$, we have that $d^N(P_1, p_1) = d^N(P_3, p_1) = 4$. This implies that

$$\begin{aligned} d^N(P_1, u) - d^N(P_3, u) &\geq d^N(P_1, p_1) + d^N(p_1, u) - d^N(P_3, p_1) - d^N(p_1, u) \\ &= 0, \end{aligned}$$

which contradicts Claim 8.

ii. **P'_2 is adjacent to a pendant level-2 blob P'_4:** Observe that p'_4 must be placed on s'_2 such that $d^{N'}(p'_1, p'_4) = 2$. Then we have that

$$d^{N'}(P'_4, u') - d(P'_1, u') = 7 - 4 = 3.$$

In $\bar{B}$, P_2 must be adjacent to both P_1 and P_4 Then $d^N(P_1, p_1) = 4$, whereas $d^N(P_4, p_1) = 5$. We have that

$$d^N(P_4, u) - d^N(P_1, u) \leq 1,$$

which contradicts Claim 8.

(b) **u' is in s'_3:** Then to ensure that the leaves of P'_1 and the leaves of P'_2 are shortest distance-9 apart, we require P'_1 and P'_2 to be adjacent to level-2 blobs P'_3 and P'_4, respectively. But then

$$d^{N'}(P'_3, P'_4) \geq 10.$$

In $\bar{B}$, the pendant blobs P_3, P_1; P_1, P_2; and P_2, P_4 are adjacent. So we have

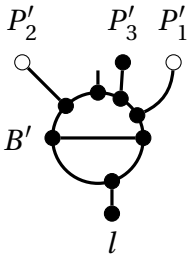
$$d^N(P_3, P_4) = 9,$$

which contradicts Claim 8.

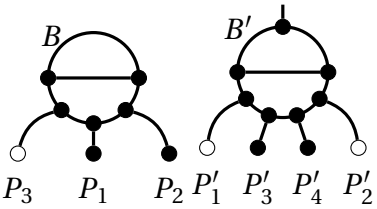
This covers all cases for whenever two level-2 blobs are adjacent. In all cases, we were able to find a contradiction with regards to the inter-taxa distances or to Claim 8. □

Claim 4. *A pendant level-2 blob may not be adjacent to both a pendant level-1 blob and a leaf simultaneously in $\bar{B}$ (and in $\bar{B}'$).*

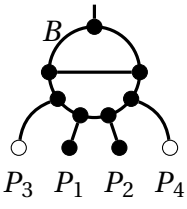
Proof. Suppose that a pendant level-1 blob P_1 is adjacent to a pendant level-2 blob P_2, and some leaf l is adjacent to P_2 in $\bar{B}$. To realize these distances in $\bar{B}'$, we must have that P'_1 and P'_2, the pendant level-2 and the level-1 blobs that correspond to P_1 and P_2 must be adjacent, and that l must be adjacent to P'_1. Since level-1 blobs may be adjacent to at most one level-2 blob, in both B and B', P_1 and P'_2 must be adjacent to a pole or u or u'. We now split into cases depending on the position of u in $\bar{B}$.

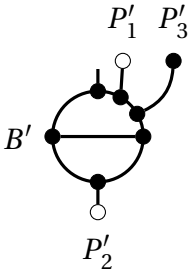


(a) Case 1. (a)

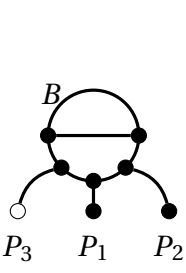


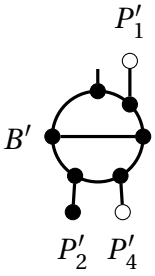
(b) Case 1. (b)

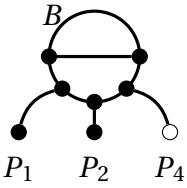




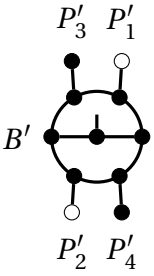
(c) Case 2. (a) i.

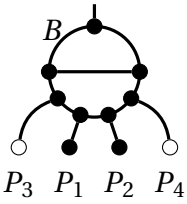






(d) Case 2. (a) ii.





(e) Case 2. (b)

Figure 7.9: The different cases from the proof of Claim 10. Pendant blobs are indicated by filled and unfilled leaves with the label P_i for some i. The filled vertices indicate a pendant level-2 blob of the form $(2,0,0,0)$, whereas unfilled vertices indicate a pendant level-1 blob on two leaves.

1. ***u* is adjacent to p_1:** We have

$$d^N(P_1, u) - d^N(P_2, u) = 4 - 6 = -2.$$

In $\bar{B}'$, we have $d^{N'}(P'_1, p'_1) = d^{N'}(P'_2, p'_1) = 4$. It follows that

$$d^{N'}(P'_1, u') - d^{N'}(P'_2, u') \geq 0,$$

which contradicts Claim 8.

2. ***u* is not adjacent to p_1:** Let v be a vertex in $\bar{B}$ such that $d^N(p_1, v) = 2$ and v is not the neighbour of l. We claim that v is either a pole or equal to u.

Note that p_1 must be adjacent to a pole since P_1 can be adjacent to at most one pendant level-2 blob, and P_1 cannot be adjacent to leaves or other pendant level-1 blobs by Claim 9. The shortest path from p_1 to v must contain this pole. Suppose for a contradiction that v is either a neighbour of a leaf or that v is a connection of some pendant blob. If v is a neighbour of a leaf l', then

$$d^N(P_1, l) = 6,$$

meaning that P'_1 must be adjacent to l' in $\bar{B}'$. But this is impossible since P'_1 is adjacent to P'_2 and l. Secondly, if v is a connection of a pendant level-1 blob P_3, then

$$d^N(P_1, P_3) = 8,$$

but this is impossible since two pendant level-1 blobs must have shortest distance at least 10 by Claim 10. Finally, if v is a connection of a pendant level-2 blob P_4, then

$$d^N(P_1, P_4) = 9,$$

which means that the corresponding pendant level-1 blob must be adjacent either to P'_2 or l in $\bar{B}'$. But both of these are forbidden by Claim 9. Therefore v must be a pole or u.

But this means that if u was not placed in the main side of $\bar{B}$ that did not contain p_1, then $\bar{B}$ would have parallel edges. So u must be contained in one of these two main sides. This means that

$$d^N(P_1, u) - d^N(P_2, u) = 5 - 7 = -2.$$

In B', as in the previous case, we have that

$$d^{N'}(P'_1, u') - d^{N'}(P'_2, u') \geq 0,$$

which clearly contradicts Claim 8

These cover all possibilities for a pendant level-2 blob to be adjacent to a pendant level-1 blob and to a leaf. In all cases, we reach a contradiction. □

Claim 5. *$\bar{B}$ (and $\bar{B}'$) contain at most one pendant level-1 blob.*

Proof. Suppose for a contradiction that $\bar{B}$ contained two pendant level-1 blobs P_1 and P_2. We know that they must be shortest distance at least 10 apart. Since $d^N(P_1, p_1) = d^N(P_2, p_2) = 3$, we require that $d^N(p_1, p_2) \geq 4$.

1. **p_1 and p_2 are contained in the same main side s of $\bar{B}$:** Consider the path from p_1 to p_2 that contains only the vertices of the main side s. Since we require $d^N(p_1, p_2) \geq 4$, we need at least three vertices in this path excluding p_1 and p_2. By Claims 9 and 11, these three vertices must be two connections p_3, p_4 of pendant level-2 blobs and the vertex u. In particular, p_1 and p_2 must be adjacent to p_3 and p_4, and p_3 and p_4 must be adjacent to u. It follows from Claim 9 that s contains only these five vertices.

   Now consider the two main sides s_1 and s_2 of B that is not s. If these main sides are both empty, then $d^N(P_1, P_2) = 9$, which contradicts the fact that $d^N(P_1, P_2) \geq 10$. So they must both contain vertices. Let v_i denote the vertex in s_i such that $d^N(v_i, p_1) = 2$, for $i = 1, 2$. Firstly, if v_1 is a neighbour of some leaf l, then $d^N(P_1, l) = 6$, meaning that l must be adjacent to P_1' in N'. But this would imply that P_1' is adjacent to a leaf l and a pendant level-2 blob P_3' in $\bar{B}'$, which contradicts Claim 11. Secondly, if v_1 is a connection of a pendant level-1 blob P_5, then $d^N(P_1, P_5) = 8$. But two pendant level-1 blobs must be shortest distance at least 10 apart. So v_1 must be a connection of a pendant level-2 blob. Similarly, v_2 must be a connection of a pendant level-2 blob. But this implies that $\bar{B}'$ contains four pendant level-1 blobs.

   The main side of $\bar{B}'$ with u' contains exactly two pendant level-1 blobs; the other two main sides of $\bar{B}'$ contains exactly one pendant level-1 blob each. Consider these two latter main sides. By Claims 9, 10, and 11 these two main sides may contain an additional pendant level-2 blob each, but no other leaves. This means that the pendant level-1 blobs in these two main sides are shortest distance at most 9 to one another, which is a contradiction. Therefore, the vertices v_i for $i = 1, 2$ cannot be neighbours of leaves / connections of pendant blobs, meaning that $\bar{B}$ contains parallel edges, which is a contradiction.

2. **p_1 and p_2 are not contained in the same main side of $\bar{B}$:** Let s_1 and s_2 denote the main sides of $\bar{B}$ that contain p_1 and p_2, respectively. If s_1 and s_2 do not contain the vertex u, then $d^N(P_1, P_2) \leq 9$ and we are done.

   So suppose without loss of generality that s_1 contains the vertex u. Since we require $d^N(p_1, p_2) \geq 4$, considering the path between P_1 and P_2 that uses only the edges from s_1 and s_2, without using the vertex u, we see that P_1 and P_2 must be adjacent to pendant level-2 blobs P_3, P_4, respectively. In particular, this path contains the subpath $p_1 p_3 v p_4 p_2$ for some pole v of $\bar{B}$. Therefore p_1 must be adjacent to u in $\bar{B}$. Then,

$$d^N(P_1, u) - d^N(P_3, u) = 4 - 6 = -2.$$

   In $\bar{B}'$, we have $d^{N'}(P_1', p_1') = d^{N'}(P_3', p_1') = 4$. It follows that

$$d^{N'}(P_1', u') - d^{N'}(P_3', u') \geq 0,$$

   which is a contradiction.

This covers all cases for when $\bar{B}$ contains more than one pendant level-1 blob, which all result in a contradiction. Therefore the claim follows. □

7.7. DISCUSSION

THE results of this chapter build on, and answer three open problems presented in the paper by van Iersel et al. [1]. We have shown that networks with a leaf on each generator side are reconstructible from their shortest distance matrix (Theorem 26). We have shown that level-2 networks are reconstructible from their sl-distance matrix (Theorem 28). We have characterized the level-2 networks that are not reconstructible from their shortest distances (Theorem 29).

Previously, it was known from [1] that level-2 networks were reconstructible from their multisets of distances, the full collection of lengths of all inter-taxa distances together with their multiplicities. An algorithm based on this result was recently presented and implemented in the Bachelor Thesis of Riche Mol [12], where a major bottleneck originated from having to adjust the large multisets of distances upon identifying and reducing a particular pendant structure. As a result, the theoretical running time of the algorithm is exponential in the number of leaves in the network (though it is polynomial in the size of the input, the multisets of distances). The results presented in this chapter point to a possibility of an alternative algorithm for constructing level-2 networks from their sl-distance matrix; since updating the sl-distance matrix can be done much quicker than for multisets of distances, we wonder if this could culminate in a polynomial-time algorithm with respect to the number of leaves in the network. It would be of great interest to see the speed-up both theoretically and in practice.

In this chapter, we have excluded all blobs incident to exactly two cut-edges. One of the consequences of excluding such blobs is that we never obtain pendant level-2 blobs of the form $(1,0,0,0)$ in our networks. Conditions for identifying and reducing such pendant blobs from level-2 networks are outlined in Lemmas 5.9 and 5.10 of [1]. In fact, such pendant blobs can be inferred from only the shortest distance matrix. This means that Theorem 28, which says that level-2 networks are reconstructible from their sl-distance matrix, holds in general when this restriction is not imposed. On the other hand, allowing for such blobs introduces a new level of complexity within alt-path structures. Call a level-2 blob with two cut-edges a *macaron*, and consider an alt-path structure G obtained from some tree T, and replace every cut-edge in G by a path of arbitrary many macarons. Call this graph G'. Let H denote a similar alt-path structure to G, and let us replace the same cut-edges by paths consisting of the same number of macarons (where by the same cut-edge, we mean the cut-edge that induces the same split). Call this resulting graph H'. It is easy to see that G' and H' realise the same shortest distance matrix. The converse is not immediately obvious. In other words, it is not clear whether excluding these 'macaron-added' alt-path structures from level-2 networks guarantee reconstructibility from their shortest distances. Nevertheless, we make the following conjecture.

Conjecture 7. *A level-2 network is reconstructible, within the class of level-2 networks, from its shortest distance matrix if and only if after suppressing macarons and degree-2*

vertices, it does not contain an alt-path structure.

A potential shortcoming of our findings lies in the fact that the networks we consider are unweighted. In phylogenetic analysis, weighted edges are often used to indicate the extent on how two species may differ from one another - to depict the passage of time, or to indicate the amount of genetic divergence between two species. The major issue that arises from weighted edges is that the foundational structures such as cherries and chains can no longer be characterized by their distances. In the rooted weighted variant of the problem, this is overcome by simulating a 'relative root' by imposing ultrametric conditions and through the use of outgroups [10]. This makes it possible to locate cherries and the rooted analogue of chains (*reticulated cherries*), even when the network is weighted. While these techniques do not translate over to the unrooted setting, some additional conditions will almost certainly be required to obtain results for the weighted variant of the problem.

Weighted unrooted networks have been considered before. For level-1 networks (or for the more general cactus graphs), it was shown recently that while there may exist multiple level-1 networks that realize the same shortest distance matrix, there is a unique optimal edge-weighted network whose sum of edge weights is minimal [7]. It was also noted that this is not the case for edge weighted, level-2 networks by considering an example presented in [13]. It could thus be of interest to ask whether if we consider optimality in terms of the multisets of distances instead, then is there a unique optimal level-2 network?

REFERENCES

[1] L. van Iersel, V. Moulton, and Y. Murakami, *Reconstructibility of unrooted level-k phylogenetic networks from distances,* Advances in Applied Mathematics **120**, 102075 (2020).

[2] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, and Y. Murakami, *Level-2 networks from shortest and longest distances,* Discrete Applied Mathematics **306**, 138 (2022).

[3] M. Bordewich and N. Tokac, *An algorithm for reconstructing ultrametric tree-child networks from inter-taxa distances,* Discrete applied mathematics **213**, 47 (2016).

[4] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[5] M. Jones, P. Gambette, L. van Iersel, R. Janssen, S. Kelk, F. Pardi, and C. Scornavacca, *Cutting an alignment with ockham's razor,* arXiv preprint arXiv:1910.11041 (2019).

[6] D. Bryant, V. Moulton, and A. Spillner, *Consistency of the neighbor-net algorithm,* Algorithms for Molecular Biology **2**, 8 (2007).

[7] M. Hayamizu, K. T. Huber, V. Moulton, and Y. Murakami, *Recognizing and realizing cactus metrics,* Information Processing Letters , 105916 (2020).

[8] S. J. Willson, *Unique reconstruction of tree-like phylogenetic networks from distances between leaves,* Bulletin of mathematical biology **68**, 919 (2006).

[9] M. Bordewich, K. T. Huber, V. Moulton, and C. Semple, *Recovering normal networks from shortest inter-taxa distance information,* Journal of mathematical biology **77**, 571 (2018).

[10] M. Bordewich, C. Semple, and N. Tokac, *Constructing tree-child networks from distance matrices,* Algorithmica **80**, 2240 (2018).

[11] P. Buneman, *The recovery of trees from measures of dissimilarity,* Mathematics in the archaeological and historical sciences (1971).

[12] R. Mol, *Reconstruction of Phylogenetic Networks: An algorithm for reconstructing Level-2 Binary Networks based on their distances.*, Bachelor's thesis, Delft University of Technology (2020).

[13] I. Althöfer, *On optimal realizations of finite metric spaces by graphs,* Discrete & Computational Geometry **3**, 103 (1988).

8

CONCLUSION

In this thesis, we have investigated reconstructibility of phylogenetic networks and have introduced, and studied properties of orchard networks.

A concerning characteristic of many network inference methods is that optimal solutions are often non-unique [1, 2]. As stated in Chapter 1, this is what makes encoding results interesting and of value to the literature; they ensure a unique solution within the confinements of the network class. Moreover, they often give ideas for polynomial-time algorithms for inferring networks. These algorithms have the appealing property that they are guaranteed to return the correct solution if sufficient data is available.

The original aim of investigating the class of orchard networks was to solve the NETWORK CONTAINMENT *problem. We were able to use cherry-picking sequences, which characterize orchard networks, to solve* NETWORK CONTAINMENT *for the class of tree-child networks, which are strictly contained in the class of orchard networks. Also, we were able to uncover a multitude of fascinating results about orchard networks. Furthermore, we showed that the class is biologically relevant in that they are the networks wherein only horizontal edges are added to some base tree.*

8.1. PRACTICAL RELEVANCE OF AND THEORETICAL ADVANCEMENT BY RECONSTRUCTIBILITY RESULTS

We wish to comment on the practical relevance of and the theoretical advancement by the results presented here, as well as any extra steps that must be taken before they can be used.

8.1.1. HEURISTICS

For results related to reconstructibility, the next major challenge is to account for non-perfect data.

SUBNETWORKS

In Chapter 3, we showed that directed level-k binary tree-child networks are reconstructible from their reticulate-edge-deleted subnetworks. The algorithm that we have provided assumes to have perfect data. To make this more robust, one could consider the following problem.

Problem 1
Given a set S of tree-child networks on the same set of taxa, find a level-k tree-child network that displays the maximum number of networks in S.

This line of building a heuristic is not uncommon for reconstructibility problems with subnetworks as building blocks. LEV1ATHAN is a heuristic for building level-1 networks from a set of triplets (trees on three vertices) [3]; TriLoNet builds level-1 networks from a set of level-1 trinets (networks on three vertices) [4]; TRIL2NET builds a level-2 network from a set of level-2 trinets [5]. All three algorithms output the unique solution should one exist; otherwise, the heuristic aims to generate a network that is consistent with as much of the networks in the input set as possible. Heuristics are used here because the problem of finding a network that is consistent with the maximal number of input networks is NP-hard (Theorem 2.17 of [6]).

Problem 1 is motivated by these factors. While network building heuristics from smaller subnetworks have been proposed, there is currently not much known on constructing networks by combining simpler networks of the same taxa. So-called consensus trees / networks can be constructed in polynomial-time [7, 8], though in some instances, they will lead to outputs consisting of highly unresolved vertices. These consensus methods aim to find a tree or a network that agrees with all or most input trees / networks. An FPT algorithm for TREE-CHILD NETWORK HYBRIDIZATION, the problem of finding a tree-child network with minimal reticulation number that displays all input tree-child networks, has been showcased in [9]. There, cherry-picking sequences are employed to reduce all input networks, and the output sequence corresponds to the tree-child network that displays them. Yet, a practical heuristic still has not surfaced for building networks from a set of input networks on the same leafsets – therefore, any finding in this area would be highly beneficial to the further exploration of phylogenetic networks. Such a scenario can arise in practice, for instance, if certain genes evolve on gene networks rather than on gene trees.

DISTANCES

A similar question can be asked to construct heuristics for distance-based methods, based on the results presented in Chapter 7. Let A, B be same-sized matrices. Then the L_∞ distance between the matrices is the value

$$L_\infty(A,B) = max_{i,j}|A_{i,j} - B_{i,j}|.$$

In Chapter 7, we gave three encoding results for distance-based reconstructibility. We now pose a general problem that can then be adjusted to cater to the three different situations.

Problem 2
Given a distance matrix $\mathscr{D}$, find an undirected network N with induced distance matrix $\mathscr{D}(N)$ that minimizes $L_\infty(\mathscr{D},\mathscr{D}(N))$.

The problem was originally posed as the *numerical taxonomy problem*, for finding a weighted directed tree for which the L_k distance is minimized between the induced distance matrix and the given input matrix [10, 11]. For $k = 1$ and $k = 2$, the problem was shown to be NP-complete [12] whereas for $k = \infty$, the problem was shown to be polynomial-time solvable, for ultrametric trees [13] as well as for additive metrics [11]. This means, in particular, that the more general problem that we consider for networks is also NP-complete if we were to use the L_1 or the L_2 norm; thus we follow suit and pose the same problem for networks using the L_∞ norm. We may define three subproblems for each of the three results of Chapter 7.

Subproblem 1
Given a distance matrix $\mathscr{D}$, find an undirected network N with induced shortest distance matrix $\mathscr{D}(N)$ that minimizes $L_\infty(\mathscr{D},\mathscr{D}(N))$.

Subproblem 2
Given a pair of distance matrices $(\mathscr{D}_m,\mathscr{D}_l)$, find an undirected level-2 network N with shortest distance matrix $\mathscr{D}_m(N)$ and longest distance matrix $\mathscr{D}_l(N)$ that minimizes $L_\infty(\mathscr{D}_m,\mathscr{D}_m(N)) + L_\infty(\mathscr{D}_l,\mathscr{D}_l(N))$.

Subproblem 3
Given a distance matrix $\mathscr{D}$, find an undirected level-k network N with shortest distance matrix $\mathscr{D}(N)$ that minimizes $L_\infty(\mathscr{D},\mathscr{D}(N))$.

Ideally we wish for the heuristic to be consistent, in the sense that, if a distance matrix is realizable by a network in a class that we have encoding results for, then we should get the unique network as output. Distance-based network construction methods such as NeighbourNet, as well as most distance-based tree construction methods satisfy the consistency property [14, 15].

For each algorithm, one can also try to calculate how much the distance matrices can deviate, but still provide a unique solution. This is the so-called 'safety radius' of a distance-based algorithm, and they have been investigated rather thoroughly for tree construction algorithms (for an overview, see [16]). Formally speaking, the safety radius of an algorithm is the maximum value r such that, if $L_\infty(\mathscr{D},\mathscr{D}_m(N)) < r\mu(N)$, where $\mu(N)$ is the length of the shortest edge in N, the algorithm outputs the network N for input $\mathscr{D}$. In this thesis we have considered unweighted networks, so that $\mu(N) = 1$.

8.1.2. DISTANCE METRICS

Another interesting observation is that encodings naturally induce a distance metric on the set of networks that are encoded by a particular building block. The distance between two networks within a class is done by taking the symmetric difference between their encodings. Since they uniquely encode the network class, no two networks have the same building blocks, thereby ensuring that the separation condition is satisfied (distinct networks have a positive distance between them). Until now, we have seen ances-

tral profiles [17, 18], triplets, trees, softwired clusters [19], trinets [20], and cherry-picking sequences [21] used to define distance metrics in this manner. For the reticulate-edge-deleted subnetwork encoding result of Chapter 3, we can naturally use the symmetric difference of the two sets of subnetworks as the distance metric. For the distance encoding results of Chapter 7, we may use any distance metric on matrices. In particular, we may use the L_∞ distance as the distance metric.

DIAMETER BOUNDS

When distance metrics are involved, it is quite natural to investigate diameter bounds within network classes [17, 18, 22, 23]. A diameter of a network class with respect to some distance metric is the maximum distance between two networks in the class. Finding the diameter in a closed form as done in [17] gives a way of normalizing the distance metric, so that every network within the class is distance at most 1 from every other network.

Problem 3
What is the diameter of the reticulate-edge-deleted subnetwork distance metric for the class of binary tree-child networks?

8.1.3. MODEL-BASED METHODS

Another area where encoding results have shown to be fruitful is in proving identifiability and distinguishability results under various model-based methods. Model-based methods construct phylogenetic networks directly from multiple sequence alignments, and include those such as pseudo-likelihood models [24], the network multispecies coalescent model (NMSC model) [25, 26], and Markov models [27]. In particular within Markov models and the NMSC model, combinatorial results on 4-leaf subnetworks have been used.

Within a Markov model, we assume a time-reversible transition of states on the edges, and some inheritance probability on the reticulation edges [27]. The transition matrices follow a certain model of DNA evolution, such as JC69, K2P, or K3P. The inheritance probability gives the proportion of sites that are inherited from each parent of a reticulation vertex. A root distribution is also specified. Given an n-leaf network with a transition matrix for each edge and an inheritance probability for each reticulation, one obtains a probability distribution on the k^n possible site patterns on the leaves of the network, for a k-state alignment (typically 4 for the DNA bases, or 2 for morphological data. See [28] or [29] for this calculation).

Within NMSC models, one is interested in computing the probability of observing a particular gene tree topology inside a given network. Reticulation edges are again given an inheritance probability; each edge is weighted, and the probability of lineages coalescing (going backwards in time) can be computed, while accounting for incomplete lineage sorting. Longer branch lengths give ample opportunity for species to coalesce along a certain edge. Under the NMSC model, the gene tree can be taken on any number of leaves; most often, quartets (trees on four leaves) are taken. There are two reasons why. The first reason is for scalability. The number of undirected trees explodes for increasing taxa size [30]. The second reason is for information. Quartets are the smallest undirected trees that are informative. For each size-four subset of the taxa, the probability of observing each of the three possible quartet gene trees within the network is

calculated. The tuple containing the three probabilities is called the *quartet concordance factor (qCF)*, and it represents the proportion of the genome for which the true network displays the gene trees [31]. It was shown in [24] that the qCFs can be computed from semi-directed networks (networks where only the reticulate edges have directions). Furthermore for level-1 networks, the qCFs can be computed from the network induced by four leaves [26]. The qCFs can be estimated from the input sequence alignment. The network that induces qCFs that are closest to the observed qCFs is inferred.

In both Markov and NMSC models, combinatorial results, in tandem with results from algebraic geometry, have been used in proving generic identifiability for certain level-1 networks [26, 27, 29]. Identifiability essentially means that the network topology and possibly the numerical parameters of the network – such as branch lengths, inheritance probabilities, and transition rates – can be inferred uniquely from the observed sequence alignment. In both Markov models and the NMSC model, the identifiability results can therefore be seen as encoding results where the building blocks are the probability distributions on the k^n site patterns and the qCFs, respectively.

Within the Markov setting, it was shown that distinguishability results can be inferred from distinguishability results on the four-leaf subnetworks [29]. Algebraic results on the four-leaf subnetworks were used to show the distinguishability between larger networks. For the NMSC model, combinatorial results on four-leaf subnetworks allowed for the identification of cycle sizes as well as the cycle partition of the leaf-set, which were key ingredients in showing distinguishability of distinct networks within the class [26]. Although these proofs do not directly use any previously existing encoding results, they do show that encoding results, like the ones in Chapter 3, and the used proof techniques, have the potential to be useful for establishing identifiability results under various models. "

8.2. FUTURE DIRECTIONS FOR ENCODING RESULTS

In Chapter 3, we considered the MLLSs as building block; in Chapter 6, we considered μ-representations as building block; in Chapter 7, we considered shortest distances, and shortest and longest distances as building blocks. Needless to say, there are more building blocks that can be considered when trying to encode a certain network class.

8.2.1. OTHER BUILDING BLOCKS

When considering building blocks, one often looks for the minimum amount of information that is needed to entirely capture a network class. This motivates researchers to first consider datatypes of lower complexity, such as subnetworks on fewer vertices or fewer edges.

SUBNETWORKS

Subnetworks such as trees, triplets, quartets, trinets, quarnets are often targeted as building blocks as methods for constructing them from sequences are rather well-established compared to their more complex counterparties. The number of networks on fewer taxa is limited, which gives a much more lenient network space to work with. While such building blocks are used in direct network inference methods [3–5], they are also used

to identify introgression, such as in the ABBA-BABA test [32], and also for hybridization, such as in HyDe [33]. They use directed networks on four species – which consist of three ingroups and one outgroup – to detect reticulate events, or more precisely, to see whether trees or networks are more suitable in representing their histories.

Another interesting open question is whether networks from certain classes (other than level-2) can be reconstructed from their level-1 subnetworks. Trees, which are level-0 networks have been an integral first step in pushing the network construction literature (e.g., through the HYBRIDIZATION NETWORK problem). Furthermore, non-binary regular networks [34] and binary level-1 networks that are 4-outward [19][1] are known to be encoded by their induced trees. Therefore, it makes sense to work with slightly more information, and see whether more general network classes are encoded by their induced level-1 networks.

Problem 4
Which network classes are encoded by their induced p-leaf level-k subnetworks?

WEIGHTED EDGES

One of the largest factors that we did not consider in this thesis are branch lengths. Typically, edges are weighted to depict the amount of genetic divergence between two events, or to depict the passage of time (especially if the molecular clock hypothesis is assumed). In the reconstructibility context, edge lengths could provide the additional information that is needed to distinguish between two networks. However, it has been shown that even considering inputs whose networks have weighted edges is sometimes not enough [2]. In the same paper, they also showed that a way to resolve this was to consider canonical forms of networks, by contracting some edges.

DISTANCE MATRICES

As for distance matrices, we have considered the shortest and the longest distance matrices induced by networks in this thesis. Other distances that have been considered in the directed network literature include multisets of distances [35], sets of distances [36], tree-average distances (where the incoming edges of every reticulation have inheritance probabilities, so that different inter-taxa paths can be averaged) [37]. The tree-average distances can be seen as the expected value of the inter-taxa distances seen in the displayed trees. This information can be obtained in practice by considering several gene trees for the taxa in consideration. The set of distances can be motivated by considering the NELP (no equally long paths) property as suggested in [2]. There it was shown that directed weighted canonical networks satisfying the NELP property are uniquely determined by the trees they display[2]. The inter-leaf distance of two leaves in a network equipped with the NELP property must all be distinct; to obtain sets of distances in practice, one can for example take inter-taxa distances on different blocks of the sequence alignment. The most commonly used in literature is still the shortest distances;

[1] A directed network is 4-outwards if the underlying undirected network does not have a cycle of length less than 5.

[2] A canonical network is a network that contains no vertices with indegree greater than 0 and outdegree 1. For more details, see [2].

though we have suggested a possible biological motivation for considering the multisets of distances in Chapter 7, the link is still rather tenuous. That said, under the NELP property, multisets can be regarded as the set of distances where the multiplicities are all one. Note that this is not exactly the same as having a set of distances because in a set, the multiplicities are unknown. For open problems regarding distance matrices, we refer the reader to Section 7.7 or to Section 8.1.1.

CLUSTERS AND SPLITS

In tree building literature, it is often customary to come across building blocks such as clusters and splits (see, for example, [30] for an overview). Both data types provide information on how certain subsets of the taxa are separated by edges in the tree or network; clusters are defined for directed graphs and splits can be seen as the undirected analogy. Biologically, they are both used to indicate clades or monophyletics groups, by grouping them into subsets of the full taxa. Clusters are known to encode directed trees [38, Theorem 9.2], softwired clusters (clusters of trees displayed by the network) encode level-1 networks that are 4-outwards [19], and splits encode undirected trees [39].

While splits do not encode undirected networks in general (see Figure 7.1), we saw in Chapter 7, in particular in Theorem 27, that they can be recovered for an undirected level-2 network from the shortest distances. This opens up the discussion for clusters or splits of network classes that can be obtained from certain building blocks. What information about the network itself, which perhaps may not be sufficient to encode the network, can be recognized from the building blocks being considered? This measures some form of strength of building blocks, with regards to how much information they capture over different network classes.

FORBIDDEN SHAPES THAT ARE HIGHLY UNLIKELY

In this thesis, we have considered only sets of input data that are assumed to be displayed / realized by the output network. However, as studied in [40], there may arise situations where we wish to exclude some subnetworks / information from the output network. Such an exclusion can be motivated by the knowledge of evolutionary scenarios that are known to be highly unlikely. Therefore, it would be interesting in future if the results of this thesis can be extended to consider certain forbidden input sets of data.

8.2.2. RELEVANCE TO THE RECONSTRUCTION CONJECTURE

Recognizing certain aspects of phylogenetic networks, as mentioned in the previous paragraph, comes up in the reconstruction problem in graph theory. In fact, encoding problems in phylogenetics can be seen as a variant of the reconstruction problem. The problem, set by Kelly and Ulam in 1941, asks whether undirected graphs are determined (encoded) uniquely by their subgraphs [41, 42]. The original problem considered the multiset of subgraphs that could be obtained from deleting vertices, called the *deck* of the graph[3]. A variant of the problem also considers the multiset of subnetworks that can be obtained by deleting edges, called the *edge-deck* of the graph [43] (see [42, Section 2] for a review of edge-reconstruction in graphs). This is somewhat, but not totally alike

[3] Each graph in a deck is called a card – the association here is that graphs can be uniquely determined by their deck of cards.

what we have considered in Chapter 3, with the differences including directed edges, labelled leaves, deleting only reticulation edges, and deleting a reticulation edge from each level-k blob.

Other subnetwork building blocks can also be categorized as a variant of this reconstruction problem. In particular, van Iersel and Moulton have considered the reconstructibility problem with leaf-deleted subnetworks as building blocks, i.e., those obtained by deleting a single leaf from a network, which have shown to encode undirected trees, undirected binary networks with at least one nontrivial cut-edge, and undirected binary level-4 networks [44]. The set of all such subnetworks were called the X-deck of a network, and they make up a subset of the full deck of a leaf-labelled network. It was shown in the same paper that any undirected network that is reconstructible from its X-deck is also reconstructible from its edge-deck; they further also showed that any undirected binary network with a size 2 chain is reconstructible from its edge-deck. Traditionally, edge-decks allow for any edge to be deleted; it may be interesting to also consider the reconstructibility problem for *blob-edge-decks*, which are multisets of networks obtained by deleting a single edge that belongs to some blob in the network, and suppressing its endpoints. We do not delete any cut edges, and so this ensures that every graph in the blob-edge-deck is a network. One can just as easily define tree-decks, triplet-decks, quartet-decks, or more generally Y-decks for some building block Y. Perhaps this connection to the unsolved problem in graph theory makes the encoding problem mathematically relevant and interesting.

8.2.3. CONNECTING DIFFERENT ENCODINGS OF THE SAME NETWORK CLASS

A network class is not limited to a single encoding. As an example, undirected trees are encoded by their induced distance matrix, splits, and induced quartets [38]. In the same text, the authors explore ways to convert quartet systems to and from split systems of undirected trees, as well as from quartet systems to induced distance matrices (see Chapter 8 of [38]). It makes intuitive sense that one should be able to move between different encodings of the same network class. After all, we could simply infer the network first, and then obtain the clusters or splits thereafter. Nevertheless, their method is direct and makes no use of the trees as an intermediate step, illustrating the point that some encodings are equivalent under certain network classes. This gives us the following question.

8

Problem 5
Find possible encodings for a given network class. Can every encoding be inferred from the other encodings?

8.2.4. ENCODINGS AND ORCHARDS

The recursive nature of orchard networks, in terms of repeatedly reducing substructures, make it an attractive network class to tackle. Effectively, this often translates to asking what datatypes provide information for identifying cherries and reticulated cherries within networks; thereafter, the network is constructed in a recursive manner, by reducing the parts of the building block that corresponds to the cherry or the reticulated

cherry. This partially explains why encoding results for tree-child networks and more recently, for the more general class of orchard networks have been explored in great detail [17–19, 35, 45–47].

For non-orchard networks, it is not always the case that they contain a cherry or a reticulated cherry. Narrowing the scope slightly, stack-free networks necessarily contain either a cherry, a reticulated cherry, or a double-reticulated cherry. Therefore, being able to reduce the third structure will be key in developing encoding results for networks beyond orchards. Therefore, the difficulty of extending these recursive results lies in reducing *double-reticulated cherries*. Within binary networks, double-reticulated cherries refer to the substructure formed by three tree vertices t_1, t_2, t_3, two reticulation vertices r_1, r_2, and two leaves l_1, l_2 such that the network contains edges $t_1r_1, t_2r_1, t_2r_2, t_3r_2$, r_1l_1 and r_2l_2 (see Figure 3.17). These were briefly mentioned in the Discussion section of Chapter 3. The difficulty appears when attempting to reattach the deleted reticulation edge, because there can be multiple places where the edge can be reattached. A wrong reattachment could potentially yield the wrong network. Perhaps another trap awaits the researcher in ensuring that networks after such a reduction remains within the considered network class.

Problem 6
Can we find encoding results for network classes that are more general than orchard networks?

But for now, we remain in the class of orchard networks and discuss what the future holds for them.

8.3. ORCHARD NETWORKS

As orchards were discovered independently by [48] and [46] just recently, there is still much that is unknown about the class. Apart from the results of Chapters 4 and 5, there are currently a few results in circulation. It is known that binary stack-free orchard networks are encoded by their ancestral profiles (or μ-representations; see Chapter 6 for more) [46, 47]. It is also known that binary orchard networks are encoded by their trinets [49]. In terms of distance matrices, directed unweighted binary orchard networks are encoded by their multisets of distances (Theorem 3.4 of [35]). Recently, a distance metric on the class of directed binary orchards was defined on basis of obtaining a certain agreement subnetwork from the two networks [21]. It was shown that the problem is NP-hard in general, though for two trees, the problem is solvable in quadratic time. Last but not least, cherry picking sequences (which are closely related to orchard networks) have been used to produce promising results in tackling the HYBRIDIZATION NUMBER problem for temporal tree-child networks and tree-child networks [50–52].

8.3.1. FORBIDDEN STRUCTURES

A key point of orchards is that they strictly contain the ever so prominent class of tree-child networks, while being strictly contained in the class of tree-based networks. Much is known about the latter two network classes; for example, a characterization of tree-child networks and tree-based networks based on some 'forbidden structures' is known

[53, 54]. For orchards, this is yet to be found. As commented in Section 4.8, crowns are forbidden within orchard networks. This notion lead to the acyclic cherry cover characterization in Chapter 5 – yet, a concrete forbidden structure characterization remains elusive.

Problem 7
What are the forbidden structures of orchard networks?

8.3.2. ENUMERATION PROBLEM

Another interesting question is on the enumeration of the orchard network class. The enumeration problem for the class of binary tree-child networks has received much attention [55–58]; The number of tree-child networks on n vertices reaches $2^{\frac{5}{4}n\log(n)+O(n)}$ asymptotically [55]. This begs the question – just how much larger is the class of orchard networks? Every network class is asymptotically small compared to the full space of phylogenetic networks; just how small, or how the network classes compare to each other is something that can be answered by enumeration results. Apart from this being an enticing problem in mathematics, there is another reason to consider enumeration problems. Within a Bayesian framework, in particular through a Markov Chain Monte Carlo Metropolis-Hastings algorithm, for a given prior distribution, a specified phylogenetic network space is traversed via rearrangement moves to obtain a network with a higher likelihood for the given input [30, 59]. Enumeration results therefore can possibly offer insight into how long the traversal should take to reach convergence (i.e., when a good approximation of the posterior distribution of the network has been obtained).

Problem 8
Find the number of distinct orchard networks on p vertices. Alternatively, find the number of distinct orchard networks on n leaves and k reticulations.

8

8.3.3. UNDIRECTED ORCHARD NETWORKS

Directed orchard networks are defined on the basis of recursively reducing cherries and reticulated cherries until the network has a single leaf. In this section, we comment briefly on how we can extend this notion to undirected networks. Biologically, undirected orchard networks can be seen as those with a base tree, which can be rooted, such that all reticulation edges are horizontal.

To give a definition that works directly on the undirected network, we first define undirected analogues of cherries and reticulated cherries. Two leaves form a *cherry* if they share a common neighbour, or if they are adjacent. To reduce cherries from networks, we delete one of the leaves and suppress the resulting degree-2 vertex, or in the case that the network contains a single edge between two leaves, simply remove one of the leaves. Two leaves x and y form a *reticulated cherry* if x and y have neighbours p and q, respectively, where pq is an edge that is not a cut-edge in N. In particular, this edge pq can be seen as the reticulation edge analogue for undirected networks. To reduce reticulated cherries from undirected networks, we delete this edge pq and suppress the resulting degree-2 vertices. Undirected orchard networks are defined as those that can be reduced to a single leaf by some sequence of cherry and reticulated cherry reductions.

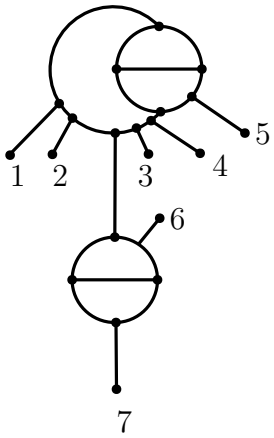


Figure 8.1: An undirected orchard network with the cherry-picking sequence $(3,4)(4,5)(4,5)(4,5)(5,1)(1,2)(2,3)(3,6)(3,6)(3,6)(6,7)$. At the start, there are two reticulated cherries that can be picked: $(1,2)$ or $(3,4)$. Upon picking $(1,2)$, the pair $(3,4)$ is no longer a reticulated cherry. The resulting network does not contain any reducible pairs, and thus cannot be picking in this order.

KEY DIFFERENCE FROM THE DIRECTED VARIANT

Though the underlying thesis of orchard networks - that networks may be reduced via a sequence of leaf-pair structures - are the same for directed and undirected networks, there are major differences between them. The way reticulated cherries can be reduced is completely different. Reticulated cherries in a directed network can only be picked in one way, with the leaf below the reticulation as the first coordinate. However, such a rule does not apply for reticulated cherries in undirected networks, where reticulated cherries may be picked via either (x,y) or (y,x).

We showed for directed orchard networks that they may be reduced in any order (Chapter 4, Theorem 9). Unfortunately, this is not true for undirected networks, even for binary networks (see Figure 8.1). This presents a rather concerning issue – this could mean that the problem of deciding whether an undirected network is orchard could be NP-hard.

The reason why this occurs is because unlike directed binary orchards, not every reducible pair (cherry or reticulated cherry) remains a reducible pair in the reduced network. That is, given a reducible pair (x,y) in a directed orchard network N, all reducible pairs remain a reducible pair in the network $N(x,y)$. This is not the case if N is an undirected orchard network. As an example, the reticulated cherry $(3,4)$ in the network N of Figure 8.1 is not a reticulated cherry in $N(1,2)$.

Problem 9
Can it be decided in polynomial-time whether an undirected binary network is orchard?

THE ROOTABILITY PROBLEM

Huber et al. studied the problem of rooting an undirected network [60]; there, it was posed as an open problem whether it is possible to find a rooting of an undirected net-

work such that the directed network is orchard in polynomial-time. Here, a rooting refers to subdividing an edge of the undirected network with the root vertex, selecting the reticulation vertices, and directing the edges of the network to obtain a directed network. It was shown that the question of deciding whether an undirected network has a tree-based rooting is NP-complete (Theorem 6 of [60]). A direct consequence of solving Problem 9 is that a polynomial-time algorithm for deciding whether an undirected binary network is orchard immediately gives a polynomial-time algorithm for the orchard rootability problem. The two problems are equivalent, as given an undirected orchard network, we can find an orchard rooting for it by building a directed network from the cherry-picking sequence (using the (1a, 2a) construction of Section 4.2.2). Indeed, the reduction of each reticulated cherry specifies the reticulation vertex for that pair depending on how the first and second coordinate is chosen. On the other hand, given an orchard rooting of an undirected network, one can find a cherry-picking sequence that reduces the directed version of the network in linear time by using the results of Chapter 4. It is easy to see that this sequence must also reduce the undirected network as well.

Another enticing aspect of this connection to the orchard rootability problem is that if Problem 9 can be solved, then it gives a line of attack for tackling the *tree-child rootability problem*, for which the computational complexity is also not known [60]. We may use the fact that the class of directed tree-child networks are properly contained within the class of directed orchard networks, and that we may reduce directed tree-child networks using tree-child sequences.

THE UNDIRECTED NETWORK CONTAINMENT PROBLEM

The undirected variant of the TREE CONTAINMENT problem has been shown (like the directed variant) to be NP-hard [61]. If parallels can be drawn from directed orchards to undirected orchards, perhaps results on undirected containment can also be shown in terms of cherry-picking sequences. Since we consider only binary networks in this case, we say that an undirected network N contains another undirected network N' on the same taxa set if N' can be obtained from N by deleting edges and suppressing degree-2 vertices. We conjecture the following.

Conjecture 8. *Let N, N' be undirected orchard networks on the same taxa set. If a cherry-picking sequence that reduces N also reduces N', then N' is contained in N.*

CONNECTION TO DISTANCES

In Chapter 7, we studied the reconstructibility problem of level-k networks with respect to their distance matrices. Is it possible that orchard networks are characterized by some induced distance matrix? Assuming we have the multisets of distances of an orchard network, we may use the results of Chapter 7 to see that cherries and chains may be identified; cherries can naturally be reduced, the question is a bit more involved for reticulated cherries. Reticulated cherries can be identified by looking for leaves that are distance-3 away with multiple paths between them. However, because we do not know which reticulated cherries can be reduced first – see Problem 8 – we cannot simply pick reticulated cherries in any order, which may render the typical inductive proof ineffective. We

also do not know how to adjust the distances of leaves that do not involve x or y when reducing a reticulated cherry (x, y).

Problem 10
Are undirected orchard networks encoded by their multisets of distances?

8.4. FINAL REMARKS

ALTHOUGH there is still a lot to be done, we have laid some of the theoretical foundations for methods that are guaranteed to generate the correct phylogenetic network if sufficient data is available. We focussed on data consisting of subnetworks with fewer reticulations, distances, or the number of distinct paths to leaves, and we have mentioned many other interesting building blocks.

We have also investigated first theoretical results on orchard networks. Its HGT-consistent time labelling characterization makes it a suitable network class to consider biologically, depending on the dataset and expected reticulate processes, all the while having computationally interesting characteristics. There is no doubt that these will be investigated further in future works.

REFERENCES

[1] K. T. Huber, L. van Iersel, V. Moulton, and T. Wu, *How much information is needed to infer reticulate evolutionary histories?* Systematic biology **64**, 102 (2014).

[2] F. Pardi and C. Scornavacca, *Reconstructible phylogenetic networks: do not distinguish the indistinguishable,* PLoS computational biology **11**, e1004135 (2015).

[3] K. T. Huber, L. Van Iersel, S. Kelk, and R. Suchecki, *A practical algorithm for reconstructing level-1 phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 635 (2010).

[4] J. Oldman, T. Wu, L. van Iersel, and V. Moulton, *Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories,* Molecular biology and evolution **33**, 2151 (2016).

[5] S. Kole, *Constructing level-2 phylogenetic networks from trinets*, Master's thesis, Technische Universiteit Delft, the Netherlands (2020).

[6] D. Bryant, *Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis*, Ph.D. thesis, University of Canterbury (1997).

[7] J. Jansson, C. Shen, and W.-K. Sung, *Improved algorithms for constructing consensus trees,* Journal of the ACM (JACM) **63**, 1 (2016).

[8] N. Tahiri, M. Willems, and V. Makarenkov, *A new fast method for inferring multiple consensus trees using k-medoids,* BMC evolutionary biology **18**, 1 (2018).

[9] R. Janssen, M. Jones, and Y. Murakami, *Combining networks using cherry picking sequences,* in *International Conference on Algorithms for Computational Biology* (Springer, 2020) pp. 77–92.

[10] P. H. Sneath and R. R. Sokal, *Numerical taxonomy,* Nature **193**, 855 (1962).

[11] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup, *On the approximability of numerical taxonomy (fitting distances by tree metrics),* SIAM Journal on Computing **28**, 1073 (1998).

[12] W. H. Day, *Computational complexity of inferring phylogenies from dissimilarity matrices,* Bulletin of mathematical biology **49**, 461 (1987).

[13] M. Farach, S. Kannan, and T. Warnow, *A robust model for finding optimal evolutionary trees,* Algorithmica **13**, 155 (1995).

[14] D. Bryant, V. Moulton, and A. Spillner, *Consistency of the neighbor-net algorithm,* Algorithms for Molecular Biology **2**, 8 (2007).

[15] F. Pardi and O. Gascuel, *Distance-based methods in phylogenetics,* in *Encyclopedia of Evolutionary Biology* (Elsevier, 2016) pp. 458–465, 1st ed.

[16] O. Gascuel, F. Pardi, and J. Truszkowski, *Distance-based phylogeny reconstruction: Safety and edge radius,* Encyclopedia of Algorithms , 567 (2016).

[17] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, *A distance metric for a class of tree-sibling phylogenetic networks,* Bioinformatics **24**, 1481 (2008).

[18] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[19] P. Gambette and K. T. Huber, *On encodings of phylogenetic networks of bounded level,* Journal of mathematical biology **65**, 157 (2012).

[20] V. Moulton, J. Oldman, and T. Wu, *A cubic-time algorithm for computing the trinet distance between level-1 networks,* Information Processing Letters **123**, 36 (2017).

[21] K. Landry, A. Teodocio, M. Lafond, and O. Tremblay-Savard, *Novel phylogenetic network distances based on cherry picking,* in *International Conference on Algorithms for Computational Biology* (Springer, 2021) pp. 57–81.

[22] J. Klawitter, *Spaces of phylogenetic networks,* Ph.D. thesis, University of Auckland (2020).

[23] R. Janssen, *Rearranging Phylogenetic Networks,* Ph.D. thesis, Delft University of Technology (2021).

[24] C. Solís-Lemus and C. Ané, *Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting,* PLoS genetics **12**, e1005896 (2016).

[25] C. Meng and L. S. Kubatko, *Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: a model,* Theoretical population biology **75**, 35 (2009).

[26] H. Baños, *Identifying species network features from gene tree quartets under the coalescent model,* Bulletin of mathematical biology **81**, 494 (2019).

[27] E. Gross, L. van Iersel, R. Janssen, M. Jones, C. Long, and Y. Murakami, *Distinguishing level-1 phylogenetic networks on the basis of data generated by markov processes,* Journal of Mathematical Biology **83**, 1 (2021).

[28] L. Nakhleh, *Evolutionary phylogenetic networks: models and issues,* in *Problem solving handbook in computational biology and bioinformatics* (Springer, 2010) pp. 125–158.

[29] E. Gross and C. Long, *Distinguishing phylogenetic networks,* SIAM Journal on Applied Algebra and Geometry **2**, 72 (2018).

[30] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications* (Cambridge University Press, 2010).

[31] D. A. Baum, *Concordance trees, concordance factors, and the exploration of reticulate genealogy,* Taxon **56**, 417 (2007).

[32] E. Y. Durand, N. Patterson, D. Reich, and M. Slatkin, *Testing for ancient admixture between closely related populations,* Molecular biology and evolution **28**, 2239 (2011).

[33] P. D. Blischak, J. Chifman, A. D. Wolfe, and L. S. Kubatko, *Hyde: a python package for genome-scale hybridization detection,* Systematic Biology **67**, 821 (2018).

[34] S. J. Willson, *Regular networks can be uniquely constructed from their trees,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 785 (2011).

[35] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[36] M. Bordewich and N. Tokac, *An algorithm for reconstructing ultrametric tree-child networks from inter-taxa distances,* Discrete applied mathematics **213**, 47 (2016).

[37] S. J. Willson, *Tree-average distances on certain phylogenetic networks have their weights uniquely determined,* Algorithms for Molecular Biology **7**, 13 (2012).

[38] A. Dress, K. T. Huber, J. Koolen, V. Moulton, and A. Spillner, *Basic phylogenetic combinatorics* (Cambridge University Press, 2012).

[39] P. Buneman, *The recovery of trees from measures of dissimilarity,* Mathematics in the archaeological and historical sciences (1971).

[40] Y.-J. He, T. N. Huynh, J. Jansson, and W.-K. Sung, *Inferring phylogenetic relationships avoiding forbidden rooted triplets,* Journal of Bioinformatics and Computational Biology **4**, 59 (2006).

[41] P. V. O'Neil, *Ulam's conjecture and graph reconstructions,* The American Mathematical Monthly **77**, 35 (1970).

[42] J. A. Bondy and R. L. Hemminger, *Graph reconstruction–a survey,* Journal of Graph Theory **1**, 227 (1977).

[43] F. Harary, *On the reconstruction of a graph from a collection of subgraphs,* in *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)* (Publ. House Czechoslovak Acad. Sci. Prague, 1964) pp. 47–52.

[44] L. Van Iersel and V. Moulton, *Leaf-reconstructibility of phylogenetic networks,* SIAM Journal on Discrete Mathematics **32**, 2047 (2018).

[45] L. van Iersel and V. Moulton, *Trinets encode tree-child and level-2 phylogenetic networks,* Journal of Mathematical Biology **68**, 1707 (2014).

[46] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[47] A. Bai, P. L. Erdős, C. Semple, and M. Steel, *Defining phylogenetic networks using ancestral profiles,* Mathematical Biosciences **332**, 108537 (2021).

[48] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[49] C. Semple and G. Toft, *Trinets encode orchard phylogenetic networks,* Journal of Mathematical Biology **83**, 1 (2021).

[50] P. J. Humphries, S. Linz, and C. Semple, *Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies,* Bulletin of mathematical biology **75**, 1879 (2013).

[51] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees,* arXiv preprint arXiv:1907.08474 (2019).

[52] S. Borst, L. van Iersel, M. Jones, and S. Kelk, *New fpt algorithms for finding the temporal hybridization number for sets of phylogenetic trees,* arXiv preprint arXiv:2007.13615 (2020).

[53] L. Zhang, *On tree-based phylogenetic networks,* Journal of Computational Biology **23**, 553 (2016).

[54] M. Hayamizu, *A structure theorem for tree-based phylogenetic networks,* arXiv preprint arXiv:1811.05849 (2018).

[55] C. McDiarmid, C. Semple, and D. Welsh, *Counting phylogenetic networks,* Annals of Combinatorics **19**, 205 (2015).

[56] M. Fuchs, B. Gittenberger, and M. Mansouri, *Counting phylogenetic networks with few reticulation vertices: tree-child and normal networks,* AUSTRALASIAN JOURNAL OF COMBINATORICS **73**, 385 (2019).

[57] G. Cardona, J. C. Pons, and C. Scornavacca, *Generation of binary tree-child phylogenetic networks,* PLoS computational biology **15**, e1007347 (2019).

[58] G. Cardona and L. Zhang, *Counting and enumerating tree-child networks and their subclasses,* Journal of Computer and System Sciences **114**, 84 (2020).

[59] D. Wen, Y. Yu, and L. Nakhleh, *Bayesian inference of reticulate phylogenies under the multispecies network coalescent,* PLoS genetics **12**, e1006006 (2016).

[60] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, and C. Semple, *Rooting for phylogenetic networks,* arXiv preprint arXiv:1906.07430 (2019).

[61] L. Van Iersel, S. Kelk, G. Stamoulis, L. Stougie, and O. Boes, *On unrooted and root-uncertain variants of several well-known phylogenetic network problems,* Algorithmica **80**, 2993 (2018).

ACKNOWLEDGEMENTS

One often describes the PhD as a lonely struggle in pushing the knowledge boundary of a specific field. After having completed the dissertation that you're reading right now, I would have to disagree. Was I able to push the knowledge boundary of a specific field? I sure hope so; was it a struggle? Without a doubt. But was it a lonely struggle? Definitely not. I was lucky enough to be surrounded and supported by a great number of amazing people, without whom myself, nor my dissertation, would be here today. So this is a list — and certainly not an exhaustive one — of all those that have made the last four years such an enjoyable, memorable, and a truly wonderful experience.

First and foremost, I would like to thank my promotors Leo and Karen. I don't think I could have asked for better role models. Leo – you are a fantastic supervisor with the utmost patience and great empathy. You were always approachable and inspiring in your knowledge of phylogenetics, as well as in your work ethic. You always seemed to know what the important things were. Karen – thank you for welcoming me to the group with open arms. Seeing how much you treasured the Optimization group made me realize just how lucky I was to be here.

To all my collaborators – Elizabeth, Momoko, Kathi, Leo, Remie, Mark, Colby, Vincent, Celine, Charles, and Norbert – thank you. You all have inspired me with your brilliance, curiosity, and creativity. Elizabeth and Colby – the skype call ranging over four different time-zones was rather exciting, to say the least. Momoko – for your great hospitality in my visit to ISM. Kathi and Vince – for creating a humorous and pleasant work environment during your visits to Delft. Celine – for your great hospitality in my visit to Montpellier. Charles – for introducing me to the concept of a 'lunch time proof'. Norbert – for your wonderful insight into ranging topics during weekly skype calls. Thank you. To the Delft phylogenetics team – Leo, Mark, and Remie – thank you for welcoming me into discussions from early on in the PhD journey. It was a lot of fun bouncing ideas off of you three and rather invigorating, to complete one project after the other in quick succession.

To everyone in the Optimization group – thank you for the captivating seminars, for the interesting conversations at lunch, and for being my family away from family. You were all very approachable and extremely friendly (except when we went paintballing). To the PhD's in the Optimization group – Teun, Remie, Fei, Jacopo, Tom, Josse, Qiaochu, Guillermo, Lara, Willem, Esther, Naqi, Merel, and Nando – thank you all for keeping things light. The warm and playful atmosphere that you have created made it feel like I was working with friends rather than colleagues. A big thank you to the support staff of the TU Delft, the unsung heroes behind the whole operation. Thank you in particular to Miriam – you are a delight to work with.

To the tennis gang – Abhas, Rishabh, Aniket, Daniel, and Alan – thanks for keeping me in shape and humbling myself on the courts. To the Bored Games people – Jason,

Jay, Cameron, and Cathy – thanks for all the late-night boardgame sessions and discussions. You all have made the coronavirus lockdown bearable and gave me something to look forward to every week. To Christian, Elly, and Dong van Delft – thanks for the great company, the delicious meals, and the (sometimes intense) IRL boardgame nights. To the Jongeren van Balthasar (you know who you are), thanks for all the laughs. Without you two, my dissertation would have been finished several months ago. To the roomies – Tom and Eline – it was (and still is) a blast living with you both. Thanks for all the DMCs, for keeping me sane, and especially for the 'sage' advice that was given from time to time. To Lotte – thank you for being a breath of fresh air in the past few months, and for introducing me to new music, new cuisine, and new experiences.

And last, but definitely not least, a big thank you to my family – my Dad Katsuhiro, my Mom Junko, Yoshi, Ji-Yeong, Haru, and Senna (who I hope to see very soon!). You all have been incredibly supportive throughout this PhD journey. Ten ants.

Curriculum Vitæ

Yukihiro Murakami

20-07-1994 Born in Hiroshima, Japan.

Education

2010–2013 Secondary School
Munich International School, Germany

2013–2016 B.A. Mathematics
University of Oxford, United Kingdom

2016–2017 M.Math Mathematics
University of Oxford, United Kingdom

2017–2021 Ph.D. Applied Mathematics
Delft University of Technology, the Netherlands
Thesis: On Phylogenetic Encodings and Orchard Networks
Promotor: Prof. dr. ir. K. Aardal
Promotor: Dr. ir. L.J.J. van Iersel

List of Publications

Journal Papers

1. **Yukihiro Murakami**, Leo van Iersel, Remie Janssen, Mark Jones, and Vincent Moulton *Reconstructing Tree-Child Networks from Reticulate-Edge-Deleted Subnetworks*, Bulletin of Mathematical Biology **81** 3823-3863 (2019).

2. Leo van Iersel, Remie Janssen, Mark Jones, **Yukihiro Murakami**, and Norbert Zeh *Polynomial-Time Algorithms for Phylogenetic Inference Problems Involving Duplication and Reticulation*, IEEE/ACM Transactions on Computational Biology and Bioinformatics **17(1)**, 14-26 (2019).

3. Momoko Hayamizu, Katharina T Huber, Vincent Moulton, and **Yukihiro Murakami** *Recognizing and realizing cactus metrics*, Information Processing Letters **157**, 105916 (2020).

4. Leo van Iersel, Vincent Moulton, and **Yukihiro Murakami** *Reconstructibility of unrooted level-k phylogenetic networks from distances*, Advances in Applied Mathematics **120**, 102075 (2020).

5. Remie Janssen and **Yukihiro Murakami** *On cherry-picking and network containment*, Theoretical Computer Science **856**, 121-150 (2021).

6. Leo van Iersel, Remie Janssen, Mark Jones, **Yukihiro Murakami**, and Norbert Zeh *A unifying characterization of tree-based networks and orchard networks using cherry covers*, Advances in Applied Mathematics **129**, 102222 (2021).

7. Elizabeth Gross, Leo van Iersel, Remie Janssen, Mark Jones, Colby Long, and **Yukihiro Murakami** *Distinguishing level-1 phylogenetic networks on the basis of data generated by Markov processes*, Journal of Mathematical Biology **83**, 32 (2021).

8. Katharina T Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, and **Yukihiro Murakami** *Reconstructibility of level-2 unrooted phylogenetic networks from shortest distances*, Discrete Applied Mathematics **306**, 138-165 (2022).

Conference Papers

1. Leo van Iersel, Remie Janssen, Mark Jones, **Yukihiro Murakami**, and Norbert Zeh *Polynomial-time algorithms for phylogenetic inference problems*, International Conference on Algorithms for Computational Biology **10849**, 37-49 (2018).

2. Remie Janssen, Mark Jones, and **Yukihiro Murakami** *Combining Networks Using Cherry Picking Sequences*, International Conference on Algorithms for Computational Biology **12099**, 77-92 (2020).

3. Remie Janssen and **Yukihiro Murakami** *Linear Time Algorithm for Tree-Child Network Containment*, International Conference on Algorithms for Computational Biology **12099**, 93-107 (2020).

Preprints

1. Katharina T Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, **Yukihiro Murakami**, and Charles Semple *Rooting for phylogenetic networks*, arXiv preprint **1906.07430** (2019).

2. Leo van Iersel, Remie Janssen, Mark Jones, **Yukihiro Murakami**, and Norbert Zeh *A Practical Fixed-Parameter Algorithm for Constructing Tree-Child Networks from Multiple Binary Trees*, arXiv preprint **1907.08474** (2019).

3. Leo van Iersel, Remie Janssen, Mark Jones, **Yukihiro Murakami** *Orchard Networks are Trees with Additional Horizontal Arcs*, arXiv preprint **2110.11065** (2021).

REFERENCES

[1] P. Forster, L. Forster, C. Renfrew, and M. Forster, *Phylogenetic network analysis of sars-cov-2 genomes,* Proceedings of the National Academy of Sciences **117**, 9241 (2020).

[2] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications* (Cambridge University Press, 2010).

[3] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L. Stougie, and J. Whitfield, *Networks: expanding evolutionary thinking,* Trends in Genetics **29**, 439 (2013).

[4] R. Abbott, D. Albach, S. Ansell, J. W. Arntzen, S. J. Baird, N. Bierne, J. Boughman, A. Brelsford, C. A. Buerkle, R. Buggs, *et al.*, *Hybridization and speciation,* Journal of evolutionary biology **26**, 229 (2013).

[5] B. E. Goulet, F. Roda, and R. Hopkins, *Hybridization in plants: old ideas, new techniques,* Plant physiology **173**, 65 (2017).

[6] D. A. Wickell and F.-W. Li, *On the evolutionary significance of horizontal gene transfers in plants,* New Phytologist **225**, 113 (2020).

[7] P. J. Keeling and J. D. Palmer, *Horizontal gene transfer in eukaryotic evolution,* Nature Reviews Genetics **9**, 605 (2008).

[8] A. Wagner, R. J. Whitaker, D. J. Krause, J.-H. Heilers, M. Van Wolferen, C. Van Der Does, and S.-V. Albers, *Mechanisms of gene flow in archaea,* Nature Reviews Microbiology **15**, 492 (2017).

[9] T. Marcussen, S. R. Sandve, L. Heier, M. Spannagl, M. Pfeifer, K. S. Jakobsen, B. B. Wulff, B. Steuernagel, K. F. Mayer, O.-A. Olsen, *et al.*, *Ancient hybridizations among the ancestral genomes of bread wheat,* science **345** (2014).

[10] J. L. Horreo, T. Suarez, and P. S. Fitze, *Reversals in complex traits uncovered as reticulation events: Lessons from the evolution of parity-mode, chromosome morphology, and maternal resource transfer,* Journal of Experimental Zoology Part B: Molecular and Developmental Evolution **334**, 5 (2020).

[11] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *Polynomial-time algorithms for phylogenetic inference problems,* in *International Conference on Algorithms for Computational Biology* (Springer, 2018) pp. 37–49.

[12] J. Ottenburghs, *Multispecies hybridization in birds,* Avian Research **10**, 1 (2019).

[13] D. A. Morrison, *An introduction to phylogenetic networks* (RJR productions, 2011).

[14] C. Solís-Lemus and C. Ané, *Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting,* PLoS genetics **12**, e1005896 (2016).

[15] H. Baños, *Identifying species network features from gene tree quartets under the coalescent model,* Bulletin of mathematical biology **81**, 494 (2019).

[16] E. Gross and C. Long, *Distinguishing phylogenetic networks,* SIAM Journal on Applied Algebra and Geometry **2**, 72 (2018).

[17] G.-L. L. Buffon Comte de, *Histoire Naturelle, générale et particulière, avec la description du Cabinet du Roi*, Vol. 5 (1755).

[18] D. Morrison, *The first phylogenetic network (1755),* (February 26, 2012), http://phylonetworks.blogspot.com/2012/02/first-phylogenetic-network-1755.html.

[19] C. Darwin, *On the Origin of Species by Means of Natural Selection* (Murray, London, 1859).

[20] V. Makarenkov, D. Kevorkov, and P. Legendre, *Phylogenetic network construction approaches,* Applied mycology and biotechnology **6**, 61 (2006).

[21] L. R. Foulds and R. L. Graham, *The steiner problem in phylogeny is np-complete,* Advances in Applied mathematics **3**, 43 (1982).

[22] T. H. Jukes, C. R. Cantor, *et al.*, *Evolution of protein molecules,* Mammalian protein metabolism **3**, 21 (1969).

[23] M. Kimura, *A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences,* Journal of molecular evolution **16**, 111 (1980).

[24] M. Kimura, *Estimation of evolutionary distances between homologous nucleotide sequences,* Proceedings of the National Academy of Sciences **78**, 454 (1981).

[25] G. Jin, L. Nakhleh, S. Snir, and T. Tuller, *Maximum likelihood of phylogenetic networks,* Bioinformatics **22**, 2604 (2006).

[26] M. Bordewich and C. Semple, *On the computational complexity of the rooted subtree prune and regraft distance,* Annals of combinatorics **8**, 409 (2005).

[27] J. Jansson, N. B. Nguyen, and W.-K. Sung, *Algorithms for combining rooted triplets into a galled phylogenetic network,* SIAM Journal on Computing **35**, 1098 (2006).

[28] D. Bryant and V. Moulton, *Neighbor-net: an agglomerative method for the construction of phylogenetic networks,* Molecular biology and evolution **21**, 255 (2004).

[29] M. Bordewich and N. Tokac, *An algorithm for reconstructing ultrametric tree-child networks from inter-taxa distances,* Discrete applied mathematics **213**, 47 (2016).

[30] S. Guindon and O. Gascuel, *A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood,* Systematic biology **52**, 696 (2003).

[31] D. Wen, Y. Yu, and L. Nakhleh, *Bayesian inference of reticulate phylogenies under the multispecies network coalescent,* PLoS genetics **12**, e1006006 (2016).

[32] C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung, *Computing the maximum agreement of phylogenetic networks,* Theoretical Computer Science **335**, 93 (2005).

[33] L. Nipius, *Rooted binary level-3 phylogenetic networks are encoded by quarnets*, Bachelor's thesis, Delft University of Technology (2020).

[34] M. Hayamizu, *On the existence of infinitely many universal tree-based networks,* Journal of Theoretical Biology **396**, 204 (2016).

[35] S. Linz and C. Semple, *Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies,* Advances in Applied Mathematics **105**, 102 (2019).

[36] M. Bordewich and C. Semple, *Computing the minimum number of hybridization events for a consistent evolutionary history,* Discrete Applied Mathematics **155**, 914 (2007).

[37] M. Baroni, S. Grünewald, V. Moulton, and C. Semple, *Bounding the number of hybridisation events for a consistent evolutionary history,* Journal of mathematical biology **51**, 171 (2005).

[38] P. J. Humphries, S. Linz, and C. Semple, *Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies,* Bulletin of mathematical biology **75**, 1879 (2013).

[39] T. N. Huynh, J. Jansson, N. B. Nguyen, and W.-K. Sung, *Constructing a smallest refining galled phylogenetic network,* in *Annual International Conference on Research in Computational Molecular Biology* (Springer, 2005) pp. 265–280.

[40] J. Jansson, C. Shen, and W.-K. Sung, *Improved algorithms for constructing consensus trees,* Journal of the ACM (JACM) **63**, 1 (2016).

[41] N. Tahiri, M. Willems, and V. Makarenkov, *A new fast method for inferring multiple consensus trees using k-medoids,* BMC evolutionary biology **18**, 1 (2018).

[42] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions,* SIAM Journal on Computing **10**, 405 (1981).

[43] J. Jansson and W.-K. Sung, *Inferring a level-1 phylogenetic network from a dense set of rooted triplets,* Theoretical Computer Science **363**, 60 (2006).

[44] K. T. Huber, L. Van Iersel, S. Kelk, and R. Suchecki, *A practical algorithm for reconstructing level-1 phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 635 (2010).

[45] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout, *Constructing level-2 phylogenetic networks from triplets,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 667 (2009).

[46] T.-H. To and M. Habib, *Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time,* in *Annual Symposium on Combinatorial Pattern Matching* (Springer, 2009) pp. 275–288.

[47] J. Oldman, T. Wu, L. van Iersel, and V. Moulton, *Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories,* Molecular biology and evolution **33**, 2151 (2016).

[48] S. Kole, *Constructing level-2 phylogenetic networks from trinets,* Master's thesis, Technische Universiteit Delft, the Netherlands (2020).

[49] F. Pardi and C. Scornavacca, *Reconstructible phylogenetic networks: do not distinguish the indistinguishable,* PLoS computational biology **11**, e1004135 (2015).

[50] S. J. Willson, *Regular networks can be uniquely constructed from their trees,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **8**, 785 (2011).

[51] P. Gambette, K. T. Huber, and S. Kelk, *On the challenge of reconstructing level-1 phylogenetic networks from triplets and clusters,* Journal of Mathematical Biology **74**, 1729 (2017).

[52] L. van Iersel and V. Moulton, *Trinets encode tree-child and level-2 phylogenetic networks,* Journal of Mathematical Biology **68**, 1707 (2014).

[53] C. Semple and G. Toft, *Trinets encode orchard phylogenetic networks,* Journal of Mathematical Biology **83**, 1 (2021).

[54] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing tree-child networks from reticulate-edge-deleted subnetworks,* Bulletin of mathematical biology **81**, 3823 (2019).

[55] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, *A distance metric for a class of tree-sibling phylogenetic networks,* Bioinformatics **24**, 1481 (2008).

[56] P. L. Erdős, C. Semple, and M. Steel, *A class of phylogenetic networks reconstructable from ancestral profiles,* Mathematical biosciences **313**, 33 (2019).

[57] G. Cardona, F. Rossello, and G. Valiente, *Comparison of tree-child phylogenetic networks,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **6**, 552 (2009).

[58] A. Bai, P. L. Erdős, C. Semple, and M. Steel, *Defining phylogenetic networks using ancestral profiles,* Mathematical Biosciences **332**, 108537 (2021).

[59] M. Bordewich and C. Semple, *Determining phylogenetic networks from inter-taxa distances,* Journal of Mathematical Biology **73**, 283 (2016).

[60] J. P. Cunningham, *Free trees and bidirectional trees as representations of psychological distance,* Journal of mathematical psychology **17**, 165 (1978).

[61] R. W. Schvaneveldt, F. T. Durso, and D. W. Dearholt, *Network structures in proximity data,* in *Psychology of learning and motivation*, Vol. 24 (Elsevier, 1989) pp. 249–284.

[62] S. L. Hakimi and S. S. Yau, *Distance matrix of a graph and its realizability,* Quarterly of applied mathematics **22**, 305 (1965).

[63] S. Forcey and D. Scalzo, *Phylogenetic networks as circuits with resistance distance,* Frontiers in Genetics **11** (2020).

[64] A. Dewdney, *Diagonal tree codes,* Information and Control **40**, 234 (1979).

[65] R. R. Sokal and C. D. Michener, *A statistical method for evaluating systematic relationships.* Univ. Kansas, Sci. Bull. **38**, 1409 (1958).

[66] N. Saitou and M. Nei, *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Molecular biology and evolution **4**, 406 (1987).

[67] F. Pardi and O. Gascuel, *Distance-based methods in phylogenetics,* in *Encyclopedia of Evolutionary Biology* (Elsevier, 2016) pp. 458–465, 1st ed.

[68] M. K. Kuhner and J. Felsenstein, *A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates.* Molecular biology and evolution **11**, 459 (1994).

[69] H.-L. Chan, J. Jansson, T.-W. Lam, and S.-M. Yiu, *Reconstructing an ultrametric galled phylogenetic network from a distance matrix,* Journal of bioinformatics and computational biology **4**, 807 (2006).

[70] M. Hayamizu, K. T. Huber, V. Moulton, and Y. Murakami, *Recognizing and realizing cactus metrics,* Information Processing Letters , 105916 (2020).

[71] L. van Iersel, V. Moulton, and Y. Murakami, *Reconstructibility of unrooted level-k phylogenetic networks from distances,* Advances in Applied Mathematics **120**, 102075 (2020).

[72] M. Bordewich, C. Semple, and N. Tokac, *Constructing tree-child networks from distance matrices,* Algorithmica **80**, 2240 (2018).

[73] M. Bordewich, K. T. Huber, V. Moulton, and C. Semple, *Recovering normal networks from shortest inter-taxa distance information,* Journal of mathematical biology **77**, 571 (2018).

[74] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, and Y. Murakami, *Level-2 networks from shortest and longest distances,* Discrete Applied Mathematics **306**, 138 (2022).

[75] R. Janssen and Y. Murakami, *On cherry-picking and network containment,* Theoretical Computer Science **856**, 121 (2021).

[76] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A unifying characterization of tree-based networks and orchard networks using cherry covers,* Advances in Applied Mathematics **129**, 102222 (2021).

[77] J. Döcker, L. van Iersel, S. Kelk, and S. Linz, *Deciding the existence of a cherry-picking sequence is hard on two trees,* Discrete Applied Mathematics **260**, 131 (2019).

[78] S. Borst, L. van Iersel, M. Jones, and S. Kelk, *New fpt algorithms for finding the temporal hybridization number for sets of phylogenetic trees,* arXiv preprint arXiv:2007.13615 (2020).

[79] L. Zhang, *On tree-based phylogenetic networks,* Journal of Computational Biology **23**, 553 (2016).

[80] L. Jetten and L. van Iersel, *Nonbinary tree-based phylogenetic networks,* IEEE/ACM transactions on computational biology and bioinformatics **15**, 205 (2018).

[81] A. Francis, C. Semple, and M. Steel, *New characterisations of tree-based networks and proximity measures,* Advances in Applied Mathematics **93**, 93 (2018).

[82] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms* (MIT press, 2009).

[83] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, and V. Moulton, *Reconstructing tree-child networks from reticulate-edge-deleted subnetworks,* Bulletin of mathematical biology **81**, 3823 (2019).

[84] L. Nakhleh, *Phylogenetic Networks*, Ph.D. thesis, The University of Texas at Austin (2004).

[85] A. R. Francis and M. Steel, *Which phylogenetic networks are merely trees with additional arcs?* Systematic biology **64**, 768 (2015).

[86] L. Jetten and L. van Iersel, *Nonbinary tree-based phylogenetic networks,* IEEE/ACM transactions on computational biology and bioinformatics **15**, 205 (2016).

[87] W. F. Martin, *Early evolution without a tree of life,* Biology direct **6**, 36 (2011).

[88] B. M. Moret, L. Nakhleh, T. Warnow, C. R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme, *Phylogenetic networks: modeling, reconstructibility, and accuracy,* IEEE/ACM Transactions on Computational Biology and Bioinformatics **1**, 13 (2004).

[89] M. Baroni, C. Semple, and M. Steel, *Hybrids in real time,* Systematic biology **55**, 46 (2006).

[90] K. T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, and C. Semple, *Rooting for phylogenetic networks,* arXiv preprint arXiv:1906.07430 (2019).

[91] K. T. Huber, L. van Iersel, V. Moulton, and T. Wu, *How much information is needed to infer reticulate evolutionary histories?* Systematic biology **64**, 102 (2014).

[92] L. van Iersel, V. Moulton, E. de Swart, and T. Wu, *Binets: fundamental building blocks for phylogenetic networks,* Bulletin of mathematical biology **79**, 1135 (2017).

[93] D. Gusfield and V. Bansal, *A fundamental decomposition theory for phylogenetic networks and incompatible characters,* in *Annual International Conference on Research in Computational Molecular Biology* (Springer, 2005) pp. 217–232.

[94] S. J. Willson, *Properties of normal phylogenetic networks,* Bulletin of mathematical biology **72**, 340 (2010).

[95] R. Janssen and Y. Murakami, *Linear time algorithm for tree-child network containment,* in *International Conference on Algorithms for Computational Biology* (Springer, 2020) pp. 93–107.

[96] S. Kelk, *Validating methods for constructing evolutionary phylogenetic networks,* (April 28, 2012), http://phylonetworks.blogspot.com/2012/04/validing-methods-for-constructing.html.

[97] I. A. Kanj, L. Nakhleh, C. Than, and G. Xia, *Seeing the trees and their branches in the network is hard,* Theoretical Computer Science **401**, 153 (2008).

[98] L. van Iersel, C. Semple, and M. Steel, *Locating a tree in a phylogenetic network,* Information Processing Letters **110**, 1037 (2010).

[99] P. Gambette, A. D. Gunawan, A. Labarre, S. Vialette, and L. Zhang, *Solving the tree containment problem for genetically stable networks in quadratic time,* in *International Workshop on Combinatorial Algorithms* (Springer, 2015) pp. 197–208.

[100] P. Gambette, A. D. Gunawan, A. Labarre, S. Vialette, and L. Zhang, *Solving the tree containment problem in linear time for nearly stable phylogenetic networks,* Discrete Applied Mathematics **246**, 62 (2018).

[101] A. D. M. Gunawan, *Solving the tree containment problem for reticulation-visible networks in linear time,* in *Algorithms for Computational Biology*, edited by J. Jansson, C. Martín-Vide, and M. A. Vega-Rodríguez (Springer International Publishing, Cham, 2018) pp. 24–36.

[102] R. Huijsman, *Tree-child Network Containment*, Bachelor's thesis, Delft University of Technology (2019).

[103] G. Cardona, J. C. Pons, and C. Scornavacca, *Generation of binary tree-child phylogenetic networks,* PLoS computational biology **15**, e1007347 (2019).

[104] R. Janssen, M. Jones, and Y. Murakami, *Combining networks using cherry picking sequences,* in *International Conference on Algorithms for Computational Biology* (Springer, 2020) pp. 77–92.

[105] C. McDiarmid, C. Semple, and D. Welsh, *Counting phylogenetic networks,* Annals of Combinatorics **19**, 205 (2015).

[106] M. Hayamizu, *A structure theorem for tree-based phylogenetic networks,* arXiv preprint arXiv:1811.05849 (2018).

[107] A. Dress, K. T. Huber, J. Koolen, V. Moulton, and A. Spillner, *Basic phylogenetic combinatorics* (Cambridge University Press, 2012).

[108] L. Nakhleh, J. Sun, T. Warnow, C. R. Linder, B. M. Moret, and A. Tholse, *Towards the development of computational tools for evaluating phylogenetic network reconstruction methods,* in *Biocomputing 2003* (World Scientific, 2003) pp. 315–326.

[109] J. Klawitter, *Spaces of phylogenetic networks*, Ph.D. thesis, University of Auckland (2020).

[110] R. Janssen, *Rearranging Phylogenetic Networks*, Ph.D. thesis, Delft University of Technology (2021).

[111] M. Jones, P. Gambette, L. van Iersel, R. Janssen, S. Kelk, F. Pardi, and C. Scornavacca, *Cutting an alignment with ockham's razor,* arXiv preprint arXiv:1910.11041 (2019).

[112] D. Bryant, V. Moulton, and A. Spillner, *Consistency of the neighbor-net algorithm,* Algorithms for Molecular Biology **2**, 8 (2007).

[113] S. J. Willson, *Unique reconstruction of tree-like phylogenetic networks from distances between leaves,* Bulletin of mathematical biology **68**, 919 (2006).

[114] P. Buneman, *The recovery of trees from measures of dissimilarity,* Mathematics in the archaeological and historical sciences (1971).

[115] R. Mol, *Reconstruction of Phylogenetic Networks: An algorithm for reconstructing Level-2 Binary Networks based on their distances.*, Bachelor's thesis, Delft University of Technology (2020).

[116] I. Althöfer, *On optimal realizations of finite metric spaces by graphs,* Discrete & Computational Geometry **3**, 103 (1988).

[117] D. Bryant, *Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis*, Ph.D. thesis, University of Canterbury (1997).

[118] P. H. Sneath and R. R. Sokal, *Numerical taxonomy,* Nature **193**, 855 (1962).

[119] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup, *On the approximability of numerical taxonomy (fitting distances by tree metrics),* SIAM Journal on Computing **28**, 1073 (1998).

[120] W. H. Day, *Computational complexity of inferring phylogenies from dissimilarity matrices,* Bulletin of mathematical biology **49**, 461 (1987).

[121] M. Farach, S. Kannan, and T. Warnow, *A robust model for finding optimal evolutionary trees,* Algorithmica **13**, 155 (1995).

[122] O. Gascuel, F. Pardi, and J. Truszkowski, *Distance-based phylogeny reconstruction: Safety and edge radius,* Encyclopedia of Algorithms , 567 (2016).

[123] P. Gambette and K. T. Huber, *On encodings of phylogenetic networks of bounded level,* Journal of mathematical biology **65**, 157 (2012).

[124] V. Moulton, J. Oldman, and T. Wu, *A cubic-time algorithm for computing the trinet distance between level-1 networks,* Information Processing Letters **123**, 36 (2017).

[125] K. Landry, A. Teodocio, M. Lafond, and O. Tremblay-Savard, *Novel phylogenetic network distances based on cherry picking,* in *International Conference on Algorithms for Computational Biology* (Springer, 2021) pp. 57–81.

[126] C. Meng and L. S. Kubatko, *Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: a model,* Theoretical population biology **75**, 35 (2009).

[127] E. Gross, L. van Iersel, R. Janssen, M. Jones, C. Long, and Y. Murakami, *Distinguishing level-1 phylogenetic networks on the basis of data generated by markov processes,* Journal of Mathematical Biology **83**, 1 (2021).

[128] L. Nakhleh, *Evolutionary phylogenetic networks: models and issues,* in *Problem solving handbook in computational biology and bioinformatics* (Springer, 2010) pp. 125–158.

[129] D. A. Baum, *Concordance trees, concordance factors, and the exploration of reticulate genealogy,* Taxon **56**, 417 (2007).

[130] E. Y. Durand, N. Patterson, D. Reich, and M. Slatkin, *Testing for ancient admixture between closely related populations,* Molecular biology and evolution **28**, 2239 (2011).

[131] P. D. Blischak, J. Chifman, A. D. Wolfe, and L. S. Kubatko, *Hyde: a python package for genome-scale hybridization detection,* Systematic Biology **67**, 821 (2018).

[132] S. J. Willson, *Tree-average distances on certain phylogenetic networks have their weights uniquely determined,* Algorithms for Molecular Biology **7**, 13 (2012).

[133] Y.-J. He, T. N. Huynh, J. Jansson, and W.-K. Sung, *Inferring phylogenetic relationships avoiding forbidden rooted triplets,* Journal of Bioinformatics and Computational Biology **4**, 59 (2006).

[134] P. V. O'Neil, *Ulam's conjecture and graph reconstructions,* The American Mathematical Monthly **77**, 35 (1970).

[135] J. A. Bondy and R. L. Hemminger, *Graph reconstruction–a survey,* Journal of Graph Theory **1**, 227 (1977).

[136] F. Harary, *On the reconstruction of a graph from a collection of subgraphs,* in *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)* (Publ. House Czechoslovak Acad. Sci. Prague, 1964) pp. 47–52.

[137] L. Van Iersel and V. Moulton, *Leaf-reconstructibility of phylogenetic networks,* SIAM Journal on Discrete Mathematics **32**, 2047 (2018).

[138] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, *A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees,* arXiv preprint arXiv:1907.08474 (2019).

[139] M. Fuchs, B. Gittenberger, and M. Mansouri, *Counting phylogenetic networks with few reticulation vertices: tree-child and normal networks,* AUSTRALASIAN JOURNAL OF COMBINATORICS **73**, 385 (2019).

[140] G. Cardona and L. Zhang, *Counting and enumerating tree-child networks and their subclasses,* Journal of Computer and System Sciences **114**, 84 (2020).

[141] L. Van Iersel, S. Kelk, G. Stamoulis, L. Stougie, and O. Boes, *On unrooted and root-uncertain variants of several well-known phylogenetic network problems,* Algorithmica **80**, 2993 (2018).