Comparing networks of life: Cherry picking the path between orchards

by

Kaari Landry

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Doctor of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2025

Abstract

Phylogenetic trees have been the traditional model for evolutionary relationships between species, but it has become clear that networks are a better model. Networks add reticulation vertices and edges which can better capture complex evolutionary scenarios such as hybridization and horizontal gene transfer. Network distance, a measure of discrepancy between two networks, is an important problem in the development of the network model as it is often used to validate construction methods. In phylogenetic networks, a cherry describes a structure containing either sibling leaves or leaves whose parents are the endpoints of a reticulation edge. A reduction operation induced on this structure removes the cherry by the deletion of a leaf or a reticulation edge. Cherry reductions have been shown to have algorithmic applications to networks, including determining containment and isomorphism, pointing to sequences of cherry reductions representing some level of similarity between two networks.

With this in mind, we define a novel phylogenetic network distance, design an algorithm that solves for it, and show that it is usable for real network applications. We give a number of equal distance formulations that we call "cherry distance". We show that it is NP-hard to calculate, even between a tree and network. Because cherry distance is NP-hard, we likely cannot avoid an exponential runtime to calculate it exactly. We do, however, design an FPT algorithm that runs in time exponential only in the combined number of reticulations in the queried networks. This is made possible by the constraint that 'blobs' remain uncomplicated by having no more than one reticulation each. Finally, we show the efficient operation of software that calculates cherry distance. This new package, CherryRed, implements the algorithm previously designed, and includes a new optimization and a fast heuristic. We furthermore use the software to show some interesting behaviours of cherry distance by comparing it to other distances, in particular we show there is a nice correlation between cherry distance and the number of network leaves downstream from a structural disagreement. In all, the impact of this program of research is in furnishing the computational toolkit available to practitioners in comparative genomics.

Contents

List of Figures

Acknowledgments

I would like to first thank the community at University of Manitoba, so many people contribute to creating a positive environment where I found great support for my work, education, and wellbeing. I hope to exemplify the welcoming spirit I found at U of M.

I cannot extend enough thanks to the Bioinfomatics Lab at U of M. Thank you to the other students and peers for your listening, your contributions, and your friendship. Special thanks to Dr. Rogerio de Leon Pereira who spent so many days in the lab with me, and to Aaron Petkau and Kimia Shadkami for sharing the experience and being so supportive.

Thank you above all to Dr. Olivier Tremblay-Savard whose leadership, insight, and encouragement made my time during this program so enriching. Thank you, Olivier, for your invaluable guidance and support.

I would like to acknowledge the hard work and effort of my committee, Dr. Shahin Kamali and Dr. Avery Miller, and my research collaborators, especially Dr. Manuel Lafond, and to my undergraduate advisor Dr. Norbert Zeh, all whose generosity was always given with kindness and sincerity. This mentorship is truly what has made possible and enlivened my academic journey. Thank you.

I would like to acknowledge the generous financial support I have received, both in the form of ongoing support, and as one-time awards and travel support. Thank you (in no particular order) to Mark and Sharon Evans, the Faculty of Graduate Studies, the Faculty of Science, the CS department, the province of Manitoba, and to the Bioinformatics lab and Dr. Olivier Tremblay-Savard. Without such support, completing this program of study would not have been possible.

A big thank you to my parents who not only provide unwavering support and love, but who themselves set an example of living, a balance of ambition and quietude, I'll be lucky to emulate.

A final personal thanks to Klinic Community Health, a registered charitable organization. Klinic helped me access life-changing health care throughout the duration of my program.

To Sam, my love, my partner, my best friend, for everything we have accomplished together, you make me capable of so much.

Chapter 1

Introduction

1.1 Research objectives

My research topic is the development and study of efficient algorithms on phylogenetic networks. Specifically, my work has introduced a new phylogenetic network distance and designed and implemented an efficient algorithm to calculate it, with experimental support. The impact of this work is in building out the computational tools needed for the burgeoning network model of evolution. The prospective benefactors of this research program are researchers and practitioners in comparative genomics and phylogenomics. The advancement of these fields is gratifying in and of itself, and so is the opportunity to make connections in the application of mathematical theory, and in the exploration of algorithms and analysis.

1.2 Personal motivation

I have had a long interest in biology that grew from an initial passion for nature. This interest was too often enjoyed from afar as I first pursued Fine Arts before settling into Computer Science. Imagine my surprise and delight to learn, upon asking my undergraduate algorithms professor for a research project, that there was such a rich field that straddled the two domains. Bioinformatics has the problem solving and analysis of algorithms that I find so compelling, paired with an application I find deeply meaningful. At this first introduction, I was hooked.

> *"Nature is not a place to visit, it is home."*
>
> – Gary Snyder

> *"If you're not having fun, you're not learning. There's a pleasure in finding things out."*
>
> – Richard Feynman

1.3 Format of document

This document is formatted as a "sandwich thesis": First presented is three more preliminary chapters, followed by three chapters that (more or less) reproduce three main publications made during this program, ending with a concluding chapter.

Chapter 2, serves as a primer on the subject matter. We start with a few simple definitions, informally presented, to facilitate the background and literature review that follows (Section 2.1). We follow the rest of the background chapter with a short summary of phylogenetics modelling (Section 2.2), computational techniques (Section 2.3), and end with an overview of network construction and distance (Section 2.4).

The next Chapter 3 is the literature review, a deeper exploration of the state of the research on two topics central to the research work in this program. First reviewed is the distance problem on (phylogenetic) trees and networks (Section 3.1), following by a comprehensive discussion of cherry picking, how it was developed and later applications (Section 3.2). Chapter 4 will formalize initial definitions.

The next three chapters represent the three projects conducted during my research program. Chapter 5 represents the introduction of our new distance, Chapter 6 the presentation of an algorithm designed to efficiently calculate it, and Chapter 7 that makes and tests the resulting software. Each chapter is presented in the same way, in which the introductory, preliminaries, and conclusion section of the publication is removed and only the main content is reproduced here. In lieu of the removed sections are instead a selected introduction and conclusion. This is done to avoid duplication of content, improve legibility, and provide a comfortable transition between each work. There is a section in each of these chapters that presents meta information pertaining to the publication of the work. In particular, it will contain information about the places and circumstances of publication and outline modifications that may have been made to the reproduced material. Then, as described, a selected introduction section follows in which any new definitions or required context are provided along with an outline of the main content. After the main published content is presented in two or three sections, a selected discussion ends each chapter. This final section serves mainly to highlight some content of the previous work that serves as a liaison between chapters.

At last, a concluding Chapter 8 completes the document. This chapter first summarizes the contents of the previous three projects in Section 8.1. Next, a section on limitations defines what imminent work we would have liked to include, time permitting (Section 8.2). The final Section 8.3 gives a deeper analysis of future directions of research.

Chapter 2

Background

2.1 Simple tree and network definitions

This section will provide some very basic definitions of trees and networks in the context of this work. Consider this section to be a generic starting point to facilitate background discussion and the literature review. For now, we rely on intuitive definitions while more formal definitions will be provided in a later chapter. See Figure 2.1 for an illustration of select definitions from this section.

The phylogenetic models presented here are called X-trees and X-networks (hereafter we just say "network") and take the form of a directed acyclic graph along with a set of taxa (or genes) X which have a bijective mapping with the leaves of the network, i.e. a leaf-labelled DAG. I equally refer to X as the *leaf labels*, and use $|X(\mathcal{N})|$ to denote the number of leaves in a network $\mathcal{N}$. Directed paths form the basis of an ancestor-descendant relationship between vertices. Throughout this manuscript, the terminology *cluster on vertex* v is used to refer to the set of leaf labels descending the given vertex v i.e. the cluster on v is the set of leaves that have v as an ancestor. The cluster of a network is X. These networks have one root vertex, representing the shared common ancestor of all sampled taxa. The direction of edges indicates the direction of time in the evolutionary model; edges are directed away from the shared common ancestor, the single root of the network, and toward the leaf taxa.

We say two networks $\mathcal{N}$ and $\mathcal{N}'$ are *isomorphic* if there exists an *isomorphism* between them. We define an isomorphism to be a bijective function between vertices of the two networks in such a way that arcs are preserved and so are leaf labels. We write $\mathcal{N} \simeq \mathcal{N}'$ if $\mathcal{N}$ and $\mathcal{N}'$ are isomorphic networks. In Chapter 6, the multi-labeling of leaves is introduced, and the definition of isomorphism is appropriately refined. Let T be a tree and $\mathcal{N}$ be a network. If T can be produced from $\mathcal{N}$ by deleting appropriate edges and vertices, then we say $\mathcal{N}$ *displays* T or that T is *embedded* in $\mathcal{N}$.

These networks are forced to be acyclic for obvious reasons, the same species

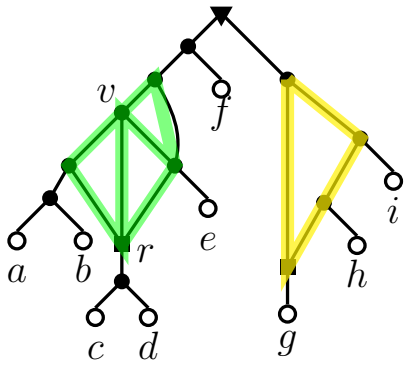

Figure 2.1: A non-binary network $\mathcal{N}$ on cluster $X = \{a, b, c, d, e, f, g, h, i\}$ with two reticulations and such that $r(\mathcal{N}) = 3$. The network root is indicated as a reversed triangle, the closed circles are tree vertices, the square vertices are reticulations, and the open circles are leaves. The level of $\mathcal{N}$ is 2, the yellow highlighted biconnected component on the right has level 1 and the green highlighted biconnected component on the left has level 2. Every non-highlighted vertex is in a biconnected component on its own. Vertex v is multifurcating, reticulation vertex r is multireticulating. The simple cherries in $\mathcal{N}$ are (a, b), (b, a), (c, d), and (d, c), the reticulated cherry in $\mathcal{N}$ is (g, h).

cannot be both the descendant of and the ancestor to another species. We also prohibit parallel edges, as well as vertices with exactly one out-edge and one in-edge (unary vertex). In some models, the edges are weighted (or have a length) in such a way to represent, for example, the passage of time or the rate of mutation along the edge. For this work, we focus on only the unweighted case. In some models the network is unrooted and semi-directed or undirected. Rooting is often done by ensuring the sample group in X includes one member known to be more distantly related, called an *outgroup*. Because it is less related, the outgroup should be placed on an edge separate from the rest of the network, making such an edge suitable for the placement of a root. We assume networks are directed here.

How these networks differ from trees are the presence of *reticulation vertices*, also called *reticulated vertices* or simply *reticulations*. Unlike in a tree, these special vertices may have many incoming edges. While it may seem like a simple modification, reticulated vertices introduce an unequal amount of complexity to the graph when compared to a tree. Usually, when referring to the complexity of the network, what is meant is the reticulate complexity: the overall number of "extra" in-edges across all reticulation vertices $\mathcal{N}$. This number is called the *reticulation number* of a network and is denoted $r(\mathcal{N})$ for network $\mathcal{N}$. We call all incoming edges to a reticulation, not just the "extra", as *reticulation edges*, however it is worth noting that not all models adopt this custom.

Next, the main methodology involves the graph structures *cherry* and *reticulated cherry*. A cherry contains two distinct leaf vertices and their shared parent. A reticulated cherry contains two distinct leaf vertices, and a reticulation edge whose

endpoints are each the leaves' (distinct) parents. There is an associated graph operation, a reduction that destroys the cherry if it exists. This reduction, cherry picking, is defined at first on trees, and then adapted to networks. Cherry picking is often performed in batches, which we call a *cherry-picking sequence*. The reverse operation, one that builds cherries, is eventually more developed in research, so we move to more general language, preferring to say *cherry sequence* (CS). If a network can be reduced to a single leaf by a CS, it is known as an *orchard*. Equivalently, an orchard is a network that can be constructed by reverse cherry picking.

A network, while directed, has an *underlying graph* on all vertices and edges of the network taken as undirected. A *biconnected component* (also called a *2-connected component*) of a network is a maximal subgraph that cannot be disconnected by removing an edge whose endpoints are both in the subgraph. The *level* of a network $\mathcal{N}$ is defined by the maximum reticulation number across all such biconnected components. In such a way, if a network $\mathcal{N}$ has a biconnected component contributing k to $r(\mathcal{N})$, and none contributing more, then we say $\mathcal{N}$ is a level-k network. A singular reticulation that has exactly two in-edges forms (at least) one cycle on the underlying graph consisting of itself, its two parents, the least common ancestor of its two parents, and all the vertices on the two paths between the reticulation and this least common ancestor. Additionally assuming there is exactly one reticulation and exactly two reticulation edges in this component (as is the case in level-1 networks), all vertices on this cycle exactly forms the biconnected component containing this reticulation. See the yellow highlighted biconnected component on the right of network $\mathcal{N}$ depicted in Figure 2.1.

The work in this thesis is restricted to binary networks, networks in which internal vertices are degree three exactly, that is to say they have either two children, or exactly two parents in the case of reticulations. When this constraint is relaxed, we can have the in-degree of reticulations unbounded which we call a *multireticulating* vertex and a *semi-binary* network. Then we can additionally have the out-degree of a vertex unbounded which we call a *multifurcating* vertex and a *non-binary* network. We do not allow a vertex to be both multireticulating and multifurcating.

2.2 Phylogenetics modelling and networks of life

2.2.1 History of phylogenetics

Phylogenetics is a field of biology that studies evolutionary relationships. Very simply defined, phylogenetics considers a set of living species and asks their degree of relatedness by finding a hierarchy of inferred common ancestors. Evolutionary relationships are historically thought of as being between species, but can also be between genes, between populations, or even (in a non-phylogenetic setting) between languages [4, 1], and material artifacts [5]. The central model in these three examples is a type of graph

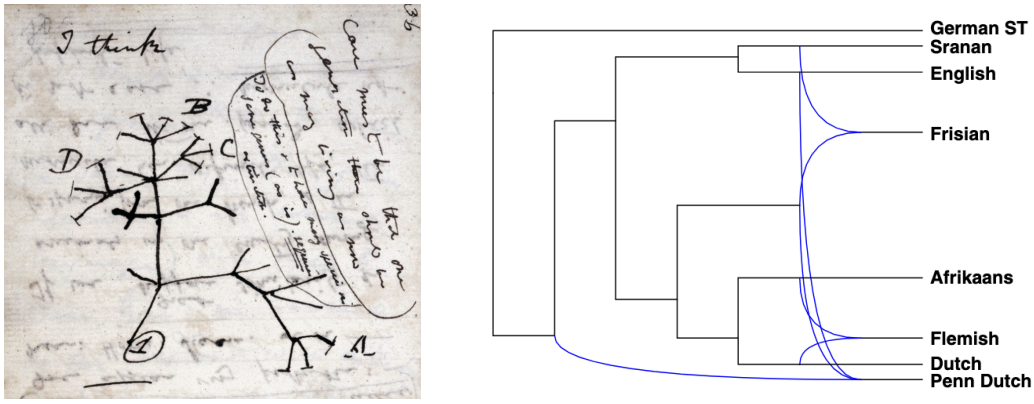

Figure 2.2: Left is the first sketch of a phylogenetic tree that Darwin drew in his notebook, 1837. The words "I think" are heading the image and emphasize the emergence of this model as novel. On the right is a modern application of a phylogenetic network. From [1], this image shows a network constructed on eight West-Germanic languages.

called the phylogenetic network. The study of networks in this context has its (pun intended) roots in the tree graph. Grouping and sorting species in the earliest forms were represented as a "tree of life" by Lamarck, Darwin, and Haeckel [6, 7, 8]. See Figure 2.2 for an example of a phylogenetic tree of life by Darwin and a network of languages. This early model of life is imagined to radiate out as unfurling branches from an initial stem. The species we know today are the very terminal nodes of an interwoven lineage sprouting from a common ancestor.

From these early ideas, a few key concepts and processes are established which form the basis of how we understand evolution today. See [9] for a reliable introductory text on the topics that follow. When we think of evolution, we might think of mutation, as DNA undergoes constant random mutations. Mutations on their own though do not make a new species, instead there are evolutionary forces that crystallize mutations in place when they are useful, and destroys them when they are not. The natural, random variation that mutation causes to DNA is *genetic drift*, which can create or destroy functionality in the genome. Functions then experience positive or negative evolutionary pressure in the form of *natural selection*, which includes selection pressures based off mate selection and increased fecundity, or survivability in the landscape based on geographic features or food availability, just to name a few. This accumulated variation followed by selection is the cycle of evolution and is what has lead to the diversity of life we witness on earth.

The phylogenetic tree model has leaves representing the known sample being queried. Often a set of extant taxa are sampled as the set represented at the leaves, in which case often the internal vertices represent the inferred ancestors of this set. The edges imply genetic inheritance. The tree has been the mainstay model due to the underlying assumption that new species arise through divergent speciation events, where an ancestor species differentiates to beget new species groups. This

intuitive assumption was historically coupled with an examination of morphology to distinguish which groups may have differentiated from a (relatively) recent common ancestor.

Modern biotechnology has fully exposed how naive this assumption is, and a much more complex picture of evolution and speciation is being painted. Passing of genetic material does not appear to be exclusively tree-like, with donations appearing to and from nearly anywhere in the so-called "tree" of life [10]. Phylogenetic networks introduce the concept of reticulations [11], which allow internal vertices to have multiple parents, and thus represent the transfer of genetic information from other species than the direct ancestor. This reticulate evolution stands in contrast to vertical evolution and splitting where genes come only from direct ancestors and speciation is implied when one species "splits" into multiple taxa. Advances with genetic sequencing technology has uncovered how deep reticulation flows, in some form or another, in all forms of life, making the tree model of evolution lacking [12]. Phylogenetic networks more accurately retell the real genetic story of evolution.

2.2.2 Models of evolution on networks

There are generally considered to be three domains of life: bacteria, archaea, eukaryotes. The bacteria and archaea together are called prokaryotes while eukaryotes constitute what we generally know of as non-bacterial life: plants, animals, fungi, and protists. Reticulate evolution is seen in all these forms of life, even in viruses is evolution well-known to be network-like [13].

In prokaryotes, the dominant strategy of reticulate evolution is *horizontal* (or *lateral*) *gene transfer*, *HGT* [14, 15]. In HGT, closely related or distant individuals can transfer and exchange genetic material in transfer events ranging from cell-to-cell contact to transference via a virus. Of particular concern, antibiotic resistance is known to be transmissible in this way which has an obvious implication for human health [10, 16]. By all accounts, HGT is critical to the history of evolution of single-celled organisms.

For multicellular organisms (eukaryotes), transfer of organelle genetic material such as mitochondria and plastids is known, and this form of HGT is important to eukaryotic evolutionary history [17, 18]. There are mechanisms being uncovered, for example, in the horizontal transfer of plastid genomes through the plant-to-plant transfer of the entire organelle [19]. In even grander posturing, HGT has been cited as the reason for the invariance of triplet-encoded synthesis of amino acids across all life in an argument that notes how well-preserved this universal language is [20].

Then there are hybridization events, such as the ones created by cross-pollination or crossbreeding in plants [21, 22]. Plants have long been known to "cross-pollinate" on a local-level between varieties, however, long term hybridization like this can contribute to speciation [23]. This fact has been observed and exploited by humans in plant cultivation and was studied in the western scientific tradition as early as

1761 [24]. In the modern context, hybrid varieties are bred and genetically engineered for commercially desirable qualities like disease- and climate-resilience [25]. The beneficial aspects of hybridization of plants is kept in balance; it also allows for transfer of herbicide resistance and general robustness of weeds [10, 26]. Hybridization can occur in animals. Rarely, speciation events can be attributed to a kind of hybridization and backcrossing (when hybrids cross with one of its originating groups) known as introgression [27].

In this light, let's consider the biological interpretation of the network model. These are not the only interpretations of course, but will broadly cover major themes of the model. First, recall that we have restricted ourselves to binary networks. There is a mathematical convenience here, but there is also a biological relevance: it is quite unlikely that a speciation event results in the divergence of multiple (more than two) distinct species, in the same vein we assume there are only two parental species in reticulated evolutionary events. Though exceptions always exist in biology, speciation appears to be naturally binary.

We have established that edges of the network are directed "forward" in time, however there is an interpretation of the network as a tree, in such a way that the tree edges are arcing "forward", but that other non-tree edges are travelling "horizontally". This instantaneous edge (or the reticulation itself) represents the reticulated evolution event. Recall that not all models consider all incoming edges to a reticulation vertex to be reticulation edges. It can be interpreted that one of these edges is a tree edge, tracking the forward natural evolution of a species. Then, there is a horizontal edge coming from some other lineage meeting at a reticulation vertex. This is a model of HGT in which there is an obvious proper tree lineage with an external genetic donation. To model hybridization of plants (and to a lesser extent animals), all edges that are incoming a reticulation vertex are horizontal, reticulated edges. This model enforces that the hybridizing species exist simultaneously for such an event to occur. Recall that some models have edge weights, in the unrooted model for example, sometimes reticulation edges are weighted to denote the strength of genetic signal.

What's more, the network model is a benefit when it comes to handling real world data. Networks, being able to express data in a more general way, provide much needed flexibility when confronting such issues as errors in the phylogenetic construction pipeline, missing and poor data handling [28], deficient evolutionary model selection [29], and the use of different datasets or different construction methods/models that result in incompatible evolutionary relationships [30]. In fact, aiming to infer a tree may result in averaging, which strays from the input data and may commit to a result that has ignored the real signal from the data [31, 28]. Even more generality, in the form of non-binary networks, allows the expression of uncertainty in the order of speciation events, for example. Truly tree-like evolution may only be the case for a small portion of the genome [10], but even when inference of a tree is the preferred final result, it is prudent to only commit to the tree-shape as late as is possible and stay network-like during preliminary data explorations in the abstract early stages of

data processing [29].

2.2.3 Biologically-based network models with applications to human health

Another interesting consideration for the use of networks to model animal evolution is the broad application to systems. A few highlighted here have a particular application to human health. Let's start with the emerging concept of the hologenome and metagenome. Both study genetic selection and evolution at the level of (symbiotic and otherwise) communities. The *metagenome* describes the entire genomic material constituting an environment. The *hologenome* considers evolution at the level of the host and its symbionts and parasites [16, 10]. Multicellular organisms all have a community of microbiota which contributes to functions like nutrition and metabolism [32], a community transmitted through generations and receiving donations from a variety of sources.

In humans, gut microbiota are essential for good metabolic health, its disbalance has health implications for gastrointestinal issues like inflammatory bowel disease and irritable bowel syndrome. The human microbiome consists of microbiota on nearly our every surface, with distinct populations on the mouth, skin, and vagina [33]. More than just GI, the gut microbiome is associated with disease and disorder across many body systems such as metabolic (type 2 diabetes, obesity) and autoimmune (type 1 diabetes, rheumatoid arthiritis). Overall the microbiome is also implicated in neurodegenerative diseases and oral health and disease including cancer [34, 35, 36, 37].

For another application of this network model, see *phylodynamics*, the application of the phylogenetic model to the epidemiological patterns and dynamics of pathogens. Phylodynamics is mainly concerned with the interactions between genetics and the pathogen dynamics in populations, for example in the investigation of the effects of viral genetic change on the immune response of the host [38]. Or see *fate mapping* which tracks the differentiation and migration of cells through their development, where a phylogenetic approach has been shown to have a practical application and has been envisioned to be applied to the study of the development of cancer cell populations [39].

2.3 Computation and networks

2.3.1 Practical contributions of computer science

Computer science is woven into nearly every discipline, creating many specialties across disciplines. Biology is replete with data, making way for the broad field of

bioinformatics, the collection and study of biological data with computational methods. Bioinformatics is a broad domain encompassing diverse topics. One naturally thinks of string-based computational problems when we think of DNA sequence analysis and analysis of gene expression. Another prominent problem in bioinformatics is structure prediction of protein which seeks to infer, based on the amino acid sequence, the three-dimensional structure of a protein. This document is concerned with models of evolution in phylogenetics. Modern techniques of genetic sampling in phylogenetics requires the intervention of computer science techniques for data handling, modelling, and analysis of sequence data for phylogenetic inference. This includes work on the network model, as network problems are significantly more complex than the same problems on trees, so accurate and efficient network algorithms are needed. This thesis is mainly a work of algorithm engineering on networks. This section will briefly survey what main algorithmic techniques are used in the research contributions that follow. See [40] for a reliable introductory text on these topics.

Central to the study of algorithms is the issue of NP-hardness. Despite a possible solution to an NP-hard problem being easy to verify, to exhaust a search for the optimal solution among all possible ones is super polynomial, and currently, it is suspected that every NP-hard problem can only be solved algorithmically in super-polynomial time. A problem is proved to be NP-hard by way of a *reduction* from an existing NP-hard problem to it, in which one shows that if the target problem can be solved by an efficient algorithm, then so would the known NP-hard one. A reduction often employs structures composed of the target problem that model components of the NP-hard problem, called *gadgets*. Gadgets are then shown to be composable in such a way that any instance of the NP-hard problem is simulated on the target problem.

When it comes to algorithm design, we employ an important algorithmic paradigm known as *fixed-parameter tractibility* where we desire a selected and controlled-for exponential factor. We accept that the exponential runtime on NP-hard problems is non-negotiable and instead design to mitigate the size of the non-constant exponential factor, which in networks is often the reticulation number of the network that serves as input or is produced as a solution. We also employ the algorithmic strategy of *dynamic programming*, a paradigm that allows a solution to be built up from pre-calculated and stored subsolutions, provided we show that the sum of such subsolutions does indeed lead to the solution on the complete input. As an aside, the strategy of dynamic programming itself can be thought of in two ways. Mainly, it refers to a strategy of filling a table by tabulating subsolutions. When it considers the smallest subsolutions first, it is considered the "bottom-up" form of dynamic programming. There is also the option of performing a "top-down" calculation, wherein the table is used as a form of memoization. Any divide-and-conquer approach might benefit from adding such a table, as will making a main recursive call on the largest subproblem.

In implementation, speed of the software is of the utmost importance. When looking to win speed, we can opt for an *approximation* in the sense that we choose

to sacrifice finding an exact answer. When the solution produced is within a factor c of the optimal, we call this a *c-approximation*. Ideally c is a constant, i.e. it does not depend on the size of the input instance. Different from this formal guarantee of the quality of the produced solution, we can also use a *heuristic*. A heuristic in the broad sense is a shortcut, and its success is in its speed as long as the quality of the solution is "good enough" in some sense. This can be achieved, for example, by ranking possible choices made at the branching step of an algorithm. This is often done by defining a simple local optimization such that the algorithm is making an easy (fast) and greedy local choice.

2.3.2 Classification of networks

Reticulation is the key difference between the phylogenetic tree and the phylogenetic network. In terms of computability, the size of the space of networks dramatically increases with every reticulation i.e. there are vastly fewer possible trees on a given X set than there are networks with even one reticulation on the same X, and each additional reticulation makes even more combinations. Furthermore, trees have the property where if you take any two internal nodes, if their clusters overlap, then one cluster is fully contained in the other. This attribute is variously called *nested* or *laminar*. Reticulations break this property, and combined with the larger space, computations like building, searching, and evaluating are inherently more difficult on networks. In this way, reticulations are the source of computational complexity of networks.

If we want to mitigate such complexity, we must work with a manageable subset of the networks in the total possible input or solution space. We do so by applying constraints to the structure of the network, which form the basis for the classification of the space of networks. Of course the phylogenetic network structure is one such restriction based on a biological model, but we also work with other restrictions we employ for efficiency purposes as we wish to make runtime and correctness guarantees. We can then examine how mathematical properties that emerge in a class can be harnessed. Each class of networks provides a certain predictability that can be exploited for a number of benefits such as being able to prove stable properties while being processed or being able to characterize a network in another form like as a set of vectors or as a vertex cover. These properties can be used in an algorithm's design or used to show correctness of such an algorithm.

The constraints that form classes may be on the vertices, the edges themselves or on some other property the network exhibits. Being a major source of complexity, it is also natural to place constraints on the placement of reticulation vertices. Only a few network classes are summarized here, those with the most relevance to my research and with a biological interpretation. Later, in Section 3.2.3, one class of networks (orchards) will be discussed on its own because of its centrality in my research. For an excellent review covering classes of phylogenetic networks and their biological and

mathematical significance, see [2].

One very intuitive class, that we have alluded to informally, is the *temporal* or *time-consistent* network, introduced in 2004 [41]. A network is defined alongside an assignment of "time" for each vertex on the network, following a partial order. All non-reticulate edges are required to travel forward in time. Meanwhile, reticulation edge's endpoints have an equal assignment of "time", (these edges travel "horizontally" rather than forward). The biological incentive has been made clear, for a hybridization or HGT event to occur, both parental lineages must exist simultaneously.

We can place constraints on ancestry between the reticulations to induce a class of networks. For example, *tree-sibling network* is the class of networks such that every reticulation has at least one nonreticulate sibling [42]. One very fruitful class of networks is *tree-child networks* (TCN), introduced by Cardona et al. in 2009 [43]. A TCN stipulates that every internal vertex in the network have at least one child that is a tree vertex. This is equivalent to saying that every internal vertex has a path to a leaf that goes through only tree vertices, called a *tree path*. A natural extension of this is the observation that there is a tree path from the root to every leaf. In fact, Cardona et al., when introducing the class, notice that every TCN, up to isomorphism, is identifiable by the multiset of each vertex's number of paths to a leaf in the network. They call this multiset the *path multiplicity vector*, which later works refer to as an *ancestral profile* (see 3.1.1 for more).

There are some important biological underpinnings that make TCNs such a compelling class. A network reticulation can be thought of as representing either an evolution event, or the inferred ancestor directly. In the latter case, a reticulation with reticulate children corresponds to a hybrid individual that again hybridizes before undergoing a speciation event, this may be an unlikely scenario in reality [44, 45], though it may be possible in some artificial circumstances (human-led evolution of plants). Consider the former case, that reticulation vertices represent evolutionary events. Note that reticulations are constrained to having a single child. Then tree-childness sensibly enforces that reticulation events leading to other reticulation events require an intermediate inferred species to exist. It is also enforced that every inferred ancestor has some extant (leaf) progeny by only vertical genetic transmission and mutation (tree vertices). While this does not model every possible scenario of evolution, those that are not modelled can be approximated [46]. Tree-childness imposes this constraint on all internal vertices, however when only reticulations are subject to have a tree child, then the network is called *stack free*. A stack-free network has no adjacent reticulations. A network can be made stack free by iteratively identifying adjacent reticulations, compressing the stack to a single multireticulating vertex.

Every network in the class of *tree-based networks* displays at least one tree on exactly all the vertices. In other words, there is a tree that contains one incoming edge for each reticulation node, and is on the same X as the network [47], called a *base tree*. For another intuition, this is class of networks that can be constructed by

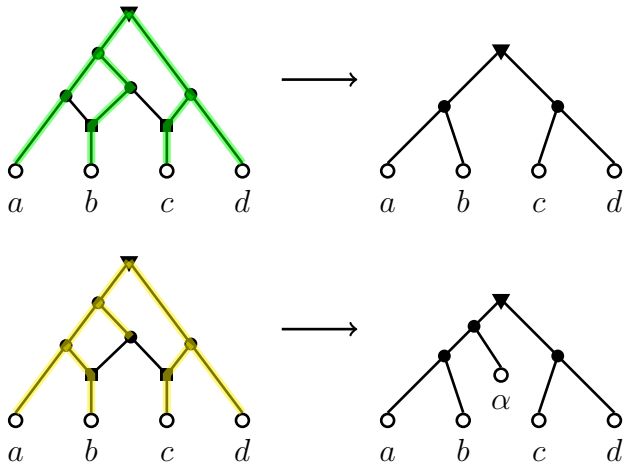

Figure 2.3: A network (upper left) and and a base tree (upper right) on the edges highlighted in green, with a reticulation edge selected for each reticulation. The base tree is on the same leaf set. Lower left is the same network, and a different set of edges are highlighted in yellow that also includes one edge for each reticulation. The embedded tree (lower right) is not a base tree, it is not on the same X set. The network (upper and lower left) is an orchard, for example $\langle (b,a)(c,b)(b,a)(c,d)(a,d) \rangle$ completely reduces it to a network on the leaf d, but it is not a TCN, the two reticulations share a parent with no other children. It is level-2.

adding reticulate edges to a tree. It is also worth noting that for any set of trees on the same taxa, there is a tree-based network that displays them [48]. See Figure 2.3 for an example of a tree-based network and a displayed tree that is not a base tree. The TCN class is related to tree-based networks in that it is exactly the class of networks wherein *every* embedded tree is a base tree [49].

Tree-based networks only require that one base tree exists, so every TCN is tree-based but the reverse is not true. Every TCN is an orchard, but the reverse is not true (as in Figure 2.3). Not every tree-based network is an orchard, but every orchard has a base tree. In fact, tree-based networks and orchards are closely related, orchards being the class of networks that are trees with additional horizontal edges (tree-based networks don't require the additional edges be horizontal) [50].

The last classes of networks that will be introduced here are *galled trees* and *galled networks*. Recall how when defining the level of a network, that (when edge directions are ignored) a reticulation vertex forms a cycle, called a *gall* when it is disjoint from any other reticulation-formed cycle. When every reticulation of a network is in a gall, it is a galled tree [51]. In this way, the class of galled-trees is exactly the class of level-1 (and level-0) networks. Out of interest, there is also a generalization of this class called galled-networks, wherein reticulation cycles may overlap on non-reticulation edges [52].

See Figure 2.4, adapted from [2], which shows the inclusion relationships between the various network classes presented here.

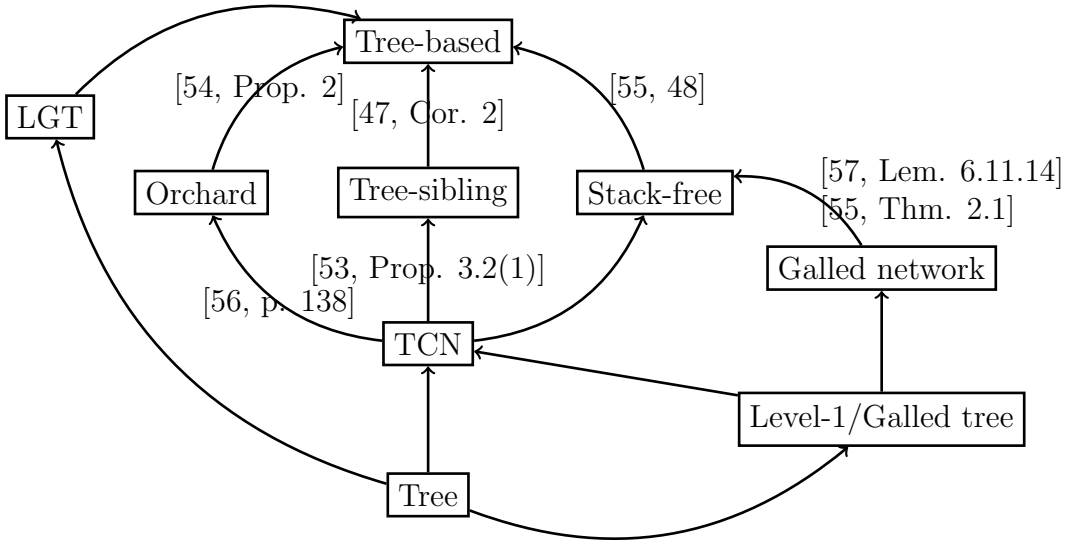

Figure 2.4: Inclusion relationship between various classes of rooted binary networks, adapted from [2, Fig. 12]. Edges are directed such that each (a, b) implies a is b. Edges are labelled with references that proves the inclusion, unlabelled edges are true by definition.

2.4 Computational problems on phylogenetic networks

2.4.1 Network construction

In the last 15 years or so, bioinformatics research has focused increasingly on solving problems related to phylogenetic networks, such as network construction, making it a well-studied problem [58, 59, 60, 61, 62, 63, 64].

As a starting point, let's look at the construction of a phylogenetic tree. We assume leaves (X) represent the real, extant set of taxa (for sake of intuition, we leave aside phylogenetic trees on genes) whose DNA sequences are sampled and analyzed. In essence, information exists at the leaves and trickles up as the inferred tree structure. Internal vertices represent ancestors and edges represent relationships inferred from the samples at the leaves, created with both a model of evolution and a method of inference. In general, the model of evolution provides differing rates at which various types of evolutionary mutation occurs. For example, swapping certain characters in the DNA sequence may be more or less common because of the physical properties of the base pair making them more secure. In this way, the evolutionary model defines an objective function, finding a maximum fit of which is the goal, with optimization being determined by the goals of that model. For example, a model of evolution that compares one nucleotide character site at a time may use inference methods like maximum parsimony, maximum likelihood, and Bayesian inference to find the best-scoring trees. Evaluating all possible trees is infeasible, so the technique is to quickly and roughly generate a starting tree, which is then subjected to rearrangements as a local search is performed with the aim of optimizing the fit [65].

Next let's consider how networks are constructed. Networks are constructed from a variety of inputs, let's focus on the example of constructing networks from a set of trees. For an intuitive understanding, we presented building trees in the context of taxa, but often trees are built on the level of genes. Trees are constructed to represent the evolutionary history of certain genes as sampled across various taxa. In fact, we track many genes in this way. So, for a set of taxa, we have a dataset consisting of many gene trees. From this set we wish to produce an evolutionary history (perhaps network-like) in which each gene's history is accounted for. For a given set of gene trees T, we preserve each gene's history by asking for a network $\mathcal{N}$ that displays every tree in T. Additionally, we demand $\mathcal{N}$ has the lowest possible reticulation number. The problem of finding the network with the least number of reticulations that displays a set of given phylogenetic trees is known as the *hybridization problem* [66] (in this context the reticulation number is called the *hybridization number*).

The hybridization problem is NP-hard on general networks, even with only two input trees [67] (though it is FPT by the hybridization number [68]). It is a well-studied problem [66, 69, 70, 71, 72]. The hybridization problem can be solved on

sets of two binary trees with an FPT algorithm [73] by characterizing the problem as finding a maximal set of agreeing substructures (a *maximum agreement forest*). When the hybridization problem is solved looking for a temporal network, it is called the *temporal hybridization problem*, which is NP-hard [74]. When the hybridization problem is solved looking for a TCN, it is called the *tree-child hybridization problem*, which is NP-hard [75]. The relaxed version of this problem, in which we ask if a network is displayed, is called the *network hybridization problem*, if this problem has TCN inputs, it is called the *tree-child network hybridization problem*[76].

In this way, the work of producing a network is in detecting and placing recombination events. Many candidate solutions may be produced by a model, and there are more associated subproblems that play a vital role in network construction. Foremost to this thesis, we require a systematic means of comparison for networks, a distance. Distances are generally used to evaluate network builds, for example one may wish to evaluate a new network construction method or software and do so by measuring the distance between output of the new method and known, accepted, or curated networks on the same data. When building a network, often we need to ask if a tree is displayed in a network, the *tree containment* problem. Containment has many variants, and it is a well-studied problem in its own right.

2.4.2 Network distance

A network *distance* is a measure of difference between two input networks. The output can be a non-negative integer, or normalized to be between 0 and 1 inclusively. With the former standard, the distance function d, for M a set of networks, is formally stated

$$d : M \times M \to \mathbb{Z}^{\geq 0}$$

To contrast, measures of similarity do exist, and can be transformed into a distance proper by subtracting from their maxima [77].

A distance can be *metric*. Specifically, for distance function d to be *metric* on M, we require, for any three networks $\mathcal{N}_a, \mathcal{N}_b, \mathcal{N}_c \in M$,

1. $d(\mathcal{N}_a, \mathcal{N}_b) = 0 \iff \mathcal{N}_a \cong \mathcal{N}_b$
2. $d(\mathcal{N}_a, \mathcal{N}_b) = d(\mathcal{N}_b, \mathcal{N}_a)$
3. $d(\mathcal{N}_a, \mathcal{N}_c) \leq d(\mathcal{N}_a, \mathcal{N}_b) + d(\mathcal{N}_b, \mathcal{N}_c)$

I will note that sometimes the concept of "positivity" ($d(\mathcal{N}_a, \mathcal{N}) > 0$) is included under these properties whereas here we have assumed all distances to be non-negative. Property 1 is also known as *identity*, property 2 is also known as *symmetry*, and property 3 is also called the *triangle inequality*. The benefit of a distance being metric is simply that metricity is a very well studied concept in mathematics, not necessarily required for a distance to be useful but may confer some desirable properties or results.

A *metric space* on networks is an ordered pair (M, d) where M is the set of networks and the distance d is metric for M, as defined above. To elucidate this and related concepts further, let's introduce an analogy of a generic graph operation $\texttt{op}$ on networks M. When $\texttt{op}$ is applied to a network $\mathcal{N} \in M$, we denote the resulting network as $\texttt{op}(\mathcal{N})$. Let's now describe the construction of a graph G. Let there be a vertex in G corresponding exactly to every network $\mathcal{N} \in M$, the network $\mathcal{N}$ can be referred to by referencing the corresponding vertex $v \in G$ and vice versa. Then, for every vertex pair $v, v' \in G$, draw an arc from v to v' if a network isomorphic to v' can be obtained from v by a single application of $\texttt{op}$, that is edge $(v, v') \in G \iff \texttt{op}(v) \simeq v'$. The distance $d(\mathcal{N}, \mathcal{N}')$ induced by $\texttt{op}$ is defined to be the length of the shortest path in G between $\mathcal{N}$ and $\mathcal{N}'$. The edges and paths are directed, so $d(\mathcal{N}, \mathcal{N}') = d(\mathcal{N}', \mathcal{N})$ is not enforced. If no path exists between two networks $\mathcal{N}$ and $\mathcal{N}'$, we say $d(\mathcal{N}, \mathcal{N}') = -\infty$ or that there is no solution.

The concept of *reachability* refers to being able to traverse between any two arbitrary networks in the space. If this holds for M under $\texttt{op}$, we say that M is *closed* under $\texttt{op}$. This property is often shown by proving that any arbitrary network in the class can be transformed into a certain specific network, which then serves as an intermediate on a path between any two arbitrary networks, proving such a path exists. Reachability more broadly implies we search and explore the complete space of networks. We can also ask of G what the longest possible shortest path is, or *diameter* of the graph. Indeed, this is also the diameter of the distance, that is, the largest possible distance given M. The diameter is often used to normalize the distance.

In this framing, we can also describe the axioms of metric space. To remain in the context the constructed graph G, let's assume d is metric. Property 1 refers to the *identity* of vertices, the fact that every unique network topology in M corresponds to no more and no less than one vertex in G. One direction of identity is the rule that non-isomorphic networks cannot be represented by the same vertex, called *separation*. The other direction says that no two distinct vertices can have a distance of 0. Property 2 refers to the *symmetry* of d, where every instance of $\texttt{op}$ can be "undone" by some inverse of it, say op^-, such that $op^-(op(\mathcal{N})) \cong \mathcal{N}$. In the context of G it can be thought of as all edges being undirected by identifying (u, v) with (v, u) since both must exist, thereby $d(\mathcal{N}, \mathcal{N}') = d(\mathcal{N}', \mathcal{N})$ for any two networks $\mathcal{N}, \mathcal{N}' \in M$ is enforced in a metric space. Finally, we have Property 3, the *triangle inequality*. This emphasizes the fact that every edge in a path contributes equally to the distance, and every path between vertices can serve as a subpath in some other connection. In essence, this property says that a shortcut cannot be found that travels through additional edges and nodes.

When evaluating a distance, we look to a few key properties that reflect the utility of the distance. First, we want the metric space to be well-connected in the sense that we should find a distance between two networks, that is, we expect reachability. Against this need we have that too much connectivity can reduce information from the distance: in the extreme a fully connected metric space will have every distance

to be one, a meaningless metric. Thus, a balance is struck and a good distance will have the sensitivity to detect small changes to the network, but not be so sensitive as to lose some "granularity", or expression of difference between other networks. In both, the desire is to be able to differentiate networks in a meaningful way.

While not strictly required of a distance or a metric, let's assume $\texttt{op}$ is closed on M, as we often do. If we assume this is the case, and that M consists of networks of differing reticulation, then we can observe that $\texttt{op}$ must have two different categories of moves: There are moves that maintain the reticulation number of the network ($r(\mathcal{N}) = r(\texttt{op}(\mathcal{N}))$), and *complexity-changing moves* which increase or decrease the reticulate complexity of the network being operated on ($r(\mathcal{N}) \neq r(\texttt{op}(\mathcal{N}))$). Now, consider searching for a path in G between a network $\mathcal{N}'$ from a given starting point $\mathcal{N}$. Unless $\mathcal{N} \simeq \mathcal{N}'$, we must traverse through intermediate networks/vertices in our search. Depending on the connectivity of $\texttt{op}$ in M, it is even possible to be required to traverse higher and lower planes of complexity in such a search (more in Section 3.1.2).

Chapter 3

Literature review

3.1 Tree and network distances

The next two sections will show some specific examples of distances defined on trees, some metric and some not. Each section will also provide examples of how each of these distances on trees has been adapted to networks.

3.1.1 RF and other cluster-based distance

The *Robinson-Foulds* (*RF*) *distance* on trees on the same leaf label set was introduced in 1981 [78] and is a popular distance due its ease of computation; the RF distance is a linear-time calculation [78, 79]. The distance is also a metric. There are multiple interpretations of the RF distance. This distance is the symmetric difference between clusters of the two input trees. The RF distance is also the number of edge contractions and expansions required to transform the first input tree into the second one. The RF distance is popular due to its speed and not its ability to differentiate; it tends to be sensitive to the placement of individual taxa, so a single errant leaf can lead to a high distance penalty.

In 2004, the RF distance for networks was introduced and proved metric on regular networks (networks such that every cluster is unique) [66]. In 2008, the RF distance was studied on tree-child time-consistent (TCTC) networks where it was also proved to be a metric [80, 81]. Interestingly, the RF distance is only a metric for this very restricted class and not for arbitrary networks in the tree-child and tree-sibling class. The diameter of RF distance between two TCTC networks is 0 when $|X(\mathcal{N})| < 2$ and $|X(\mathcal{N})|(|X(\mathcal{N})| - 1) - 3$ when $|X(\mathcal{N})| > 3$ [81]. In TCNs, the diameter is $2(m + 1)(|X| - 1)$ where m is the maximum in-degree across all reticulation vertices present [43].

A variation of the RF distance, called the *soft RF distance*, is determined by the symmetric difference of all clusters of the trees displayed by the network. Since its inception in 2010 the difference between the soft RF and RF has been considered

as unclear [57]. When an algorithm and implementation fast enough for practical use were introduced to calculate the soft RF distance in 2017, both distances were calculated on random networks to study their relationship. The authors concluded that the soft RF is more suitable than the RF distance due to the soft RF distance being skewed towards small distances and its larger range. This is in contrast to the fact that the RF distance has a normal distribution with a small mean and variance. In this way, we can see that the RF distance is less fine-grained than the soft RF distance. The soft RF distance is also one of the few implemented network distances that is openly available [82].

Another distance adapted to networks came with the introduction of tree-sibling and TCNs. TCNs were introduced as being identifiable by (and reconstructible from in polynomial time) by their *ancestral profile* (or *μ-representation*), a multiset of vectors consisting of all the *path multiplicity vectors* (or *μ-vectors*), a vector for each node of the network that represents the number of paths from that vertex to each leaf of the network. The symmetric difference (referring to multisets) of the ancestral profiles can also be taken on two input networks, the size of which produces the *μ-distance*, as first introduced on tree-sibling networks [83]. This distance is a metric on TCNs, and is identical to the RF distance when taken on trees [43]. Its diameter on TCTC networks is 0 when $|X(\mathcal{N})| \leq 2$ and $|X(\mathcal{N})|(|X(\mathcal{N})| - 1) - 1$ when $|X(\mathcal{N})| \geq 3$ [81].

Later, the identifiability results were extended to orchard networks, as it was found they are also uniquely determined by their ancestral profile (cherries and reticulated cherries can be determined by this representation) [84]. However, it was soon remarked that the proofs in this work omitted an important condition for uniqueness results to hold, i.e. that the networks be stack free [85]. This next work attempted to further generalize the result to include (stack free) semi-binary networks. This attempt was disproven by a counterexample [86], so still only holds for binary networks. To finally realize the uniqueness results of the μ-representation for the class of stack-free semi-binary networks, a modification to the μ-vectors was made to include the in-degree of vertices [87]. Interestingly, this result does not extend to non-binary networks, even if the encoding is further modified to include the vertex's out-degree. As a final note on the μ-representation of networks, the stack-free condition can be lifted, but only for binary networks, if a different modification is made to the encoding in which the number of paths from each vertex to reticulations is also accounted for, the *extended* μ-representation [88].

3.1.2 Rearrangement-based distances

A rearrangement-based distance starts with a graph operation that rearranges (makes changes without addition or removal of leaf labels) to a tree or network, for example by detaching and moving a branch. When the rearrangement operation is also symmetric and the triangle inequality holds, then it induces a metric on the network with identity

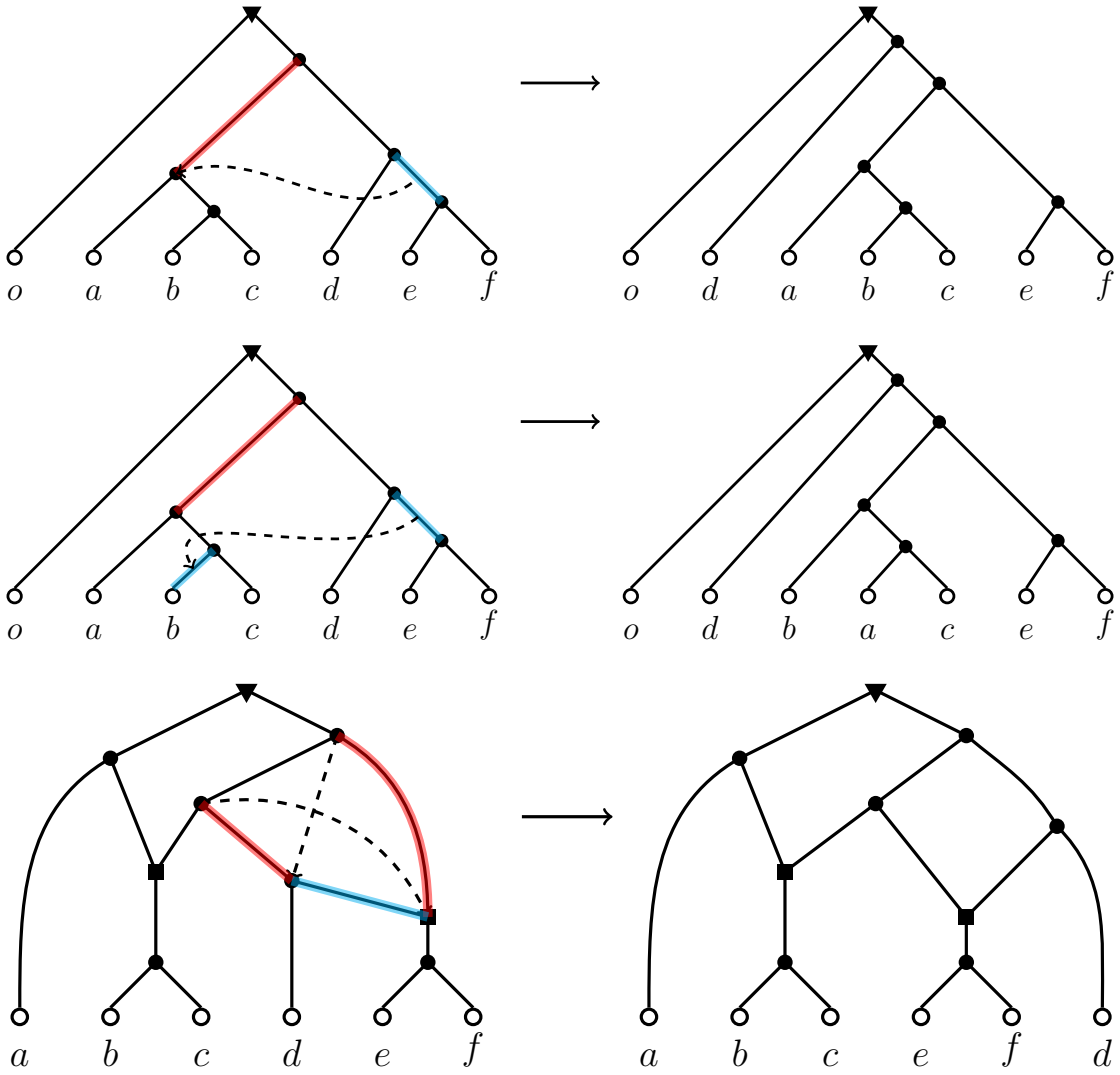

Figure 3.1: Rearrangement operations on trees and networks. Top: An SPR move on a tree where a target edge to be cut is in red, the reattachment site in cyan, and the new edge that will graft the cut subtree to the attachment site is dashed. Middle: A TBR operation on a tree where the edge being cut is highlighted in red. This time, two attachment sites are shown in cyan, one on the tree and one on the subtree (allowing it to be rerooted), the new grafting edge is dashed. The result is a similar, but slightly different from the network resulting from the SPR operation. Bottom: An rNNI move is demonstrated. The target edge is in cyan, with the red edges being swapped for the dashed edges.

being defined by network isomorphism. In this way sequences of operations define a distance based on the metric space. I present three well established rearrangement-based metrics (also called operation-based) on trees, and their definitions, then their various adaptations to networks. See Figure 3.1.

First is the nearest neighbour interchange (NNI) [89, 90] on trees. The *NNI operation* is an unrooted operation, so a rooted tree must first be unrooted. Then, an edge is selected. The subtrees on the end of this edge are swapped. Note, there can be multiple outcomes of an NNI operation, whichever outcome is selected, the tree must then be rerooted. Next, the subtree pruning and regrafting (SPR) [91] on trees is defined as follows. The *SPR operation* (or the *rooted SPR, rSPR* on rooted trees) chooses two target edges, one to cut and one as the reattachment point. Once the edge is cut, the subnetwork below the cut edge is rejoined to the attachment site. Finally, there is the tree bisection and reconnection (TBR) [92] on trees. The *TBR operation*, much like the SPR, starts by choosing an edge to cut, and a reattachment edge. Where the TBR differs however, is that we choose an edge from the cut subnetwork that will join with the reattachment edge. All three distances based on these operations are NP-hard to compute on rooted and unrooted trees [92, 93] (and by extension on networks). For all three of these rearrangement operations, the distance between two trees can be computed using FPT algorithms parameterized by the distance [92, 68], but none are considered fast enough for practical use.

We can consider each of these moves for their degree of locality which refers to the neighbourhood size. A smaller size will explore longer paths, and when the neighbourhood is larger we say the operation is more general. In terms of the generality of the above metrics, SPR, TBR, and NNI, we can see this play out by comparing the neighbourhood size of a vertex v in the metric space under each of these operations. Denote the neighbourhood of vertex v for the metric space on $\mathtt{op}$ by $N_{\mathtt{op}}(v)$ then $N_{NNI}(v) \subseteq N_{SPR}(v) \subseteq N_{TBR}(v)$ [94]. NNI is a special case of SPR where the reattachment occurs at a neighbour to the cut edge, TBR is an extension of SPR where the subtree/subnetwork is allowed to be rearranged before reconnection. As we see, we have that the least general of these operations is NNI/rNNI, followed by SPR/rSPR, with TBR being the most general.

Each of these operations and metrics have been adapted to networks; the *subnet prune and regraft* (SNPR) on rooted networks [95] (which induces a metric, the SNPR distance), and the SPR and TBR on unrooted networks [94] (which also induce metrics on unrooted networks). The SPR and TBR on networks are defined much like their tree counterparts, with the additional caveat that the resulting network should remain a valid phylogenetic network (no cycles, etc). The adaptations for NNI are more varied. First, unrooted phylogenetic networks with two moves adapting NNI are defined (with a complexity-changing move called the triangle operation) and it is shown that the class of unrooted networks is closed under these moves [96]. The *local subtree transfer* (*LST*) groups a *rooted NNI* (rNNI) move and two complexity-changing moves. The class of rooted level-1 networks was shown to be closed under LST [3], though moving between two networks with the same reticulation number may require intermediate networks with a different complexity, which the work of [97] immediately addressed. First by defining non-complexity-changing moves, *rNNI* and the *rooted SPR* (*rSPR*) moves and showing that rooted phylogenetic networks with

equal reticulation numbers form closed groups under these moves. Then, complexity changing moves are discussed. Namely, some options for arc insertion and removal are explored, and it is noted that very little is required of such moves in order to maintain the reachability result and to change the reticulation number, although the practicality of such moves should be considered to ensure the resulting networks maintain biological and mathematical relevance. Much like how TBR and SPR on trees are generalizations of the more local NNI move, an rNNI is a local move of an rSPR [97].

Let's look at how this change in level of complexity becomes a problem for a search that is optimizing for some "fit" of some data. Recall that when searching the metric space that may be induced by a rearrangement operation, it is often with the objective of fitting certain data or objective. Higher complexity networks are almost always a better fit due to the increase in structural combinations that the target network could match with. So in performing a local search through successive operations, if a complexity-increasing move is often available, it is possible we will tend towards higher complexity network. This is not ideal, since we always prefer a lower complexity which will give the tightest and therefore most meaningful fit. The result is that a search may get off track before reaching an optimal goal [97]. Once the search veers into higher complexity, it may be difficult for the search to consider moving to a lower complexity due to it being a "worse" fit. Consider what this means for selecting an appropriate rearrangement operation: We do not want the metric space to be so sparse that paths meandering "up" and "down" in complexity are required on a path between two networks. Again we see how there is a trade-off between enough connectivity to have a meaningful local selection of networks (neighbourhood size), and still be functional, in this case as a search operation.

3.2 Cherry picking and orchard networks

3.2.1 History and origins of cherry picking

Cherry picking has emerged as a new method for solving hybridization, which had until that point been dominated by a characterization by *agreement forests* (see Section 2.4.1), but work following this technique stalls for inputs that are not two binary trees. Scientists working on phylogenetic problems rarely have such clean data, so constructing binary input of two trees is not feasible in practice. Theorists were impelled to search for a technique to solve the hybridization problem on larger and more complex data sets. It was cherry picking that proved able to characterize larger input sets of complex trees into minimally hybridized networks.

First, cherry picking was enough to solve the hybridization problem on an arbitrary number of binary trees [98]. This early work found that there exists a network displaying the input set of trees if and only if a sequence of cherry-picking operations

exists that completely reduces every tree. The actual structure of this network is not found, but rather the hybridization number is calculated by a weight function based on the length of the sequence. These sequences were in a more primitive form than what will be described later. At first, this solution solved the temporal hybridization number, a problem that is not guaranteed to have a solution on even just two trees [74].

The culmination of work towards computing the hybridization number on an arbitrary size set of not necessarily binary input trees leads to [75]. This work built on previous efforts, and used cherry picking to characterize the tree-child hybridization number, which they showed to be guaranteed to have a solution. To extend this characterization to the space of all orchards, the authors introduced a leaf-attaching procedure that adds a set of leaves to the set of input trees. While they were able to bound the size of such a set by the TC hybridization number, finding where to attach these leaves on the input trees is itself a difficult problem. As before, sequences of cherry-picking operations were found and a weight function on their length is found. Interestingly, they also formulated an algorithm that constructs the resulting network rather than just finding the hybridization number. So the opposite operation of cherry picking arrives, though not formally defined and separated as such. The notation of the sequences are updated to use ordered pairs, which is now standard. The authors also found that imposing an order on the sequence ensures a TCN is constructed, which they call *tree-child sequences*. An FPT algorithm was designed and implemented in [99] parameterized by the reticulation number of the resulting network.

Also in [75], the authors formalized a reason why the agreement forests characterization of hybridization is not generalizable beyond two binary trees, thus solidifying cherry picking as the state-of-the-art for the hybridization problem on complex inputs. In essence, they established the NP-completeness (of a version for tree-child networks) which was originally posed in a conjecture from [100], that it is hard to construct the agreement forest for sets of more than three binary trees. Furthermore, there is no TCN that displays the set of trees being hybridized and also produces the appropriate agreement forest for calculating the tree-child hybridization number, ruling out being able to further exploit the agreement forest technique.

3.2.2 Cherry picking on networks

In 2019, as a part of my undergraduate honours thesis, I used cherry picking to prove that the complexity measures of level and tree-child hybridization number were preserved on *clustering*, a type of parallelization where subtrees that share a cluster are operated on independently [101]. Since cherry picking characterizes hybridization and can preserve tree-childness when tree-child sequences are used, it could be employed in a formal proof. The technique of this proof was to note that a cluster partition of a set of trees can be ordered topologically, therefore a tree-child sequence could be

constructed for the overall (hybridized) network composed of tree-child sequences of the cluster partition placed in order.

The technique of cherry picking was extended to a few other problems by authors Janssen and Murakami. First to arrive is a linear-time algorithm that can test for network containment and isomorphism on TCNs [102]. A network $\mathcal{N}'$ is *contained* in another network $\mathcal{N}$ when $\mathcal{N}'$ can be obtained from $\mathcal{N}$ only through operations of deleting leaves and reticulation edges. The idea behind the result is simple: If a network can be reduced by a tree-child sequence that reduces a second network, then the first is contained in the latter, which can further be used in both directions to show isomorphism. Helpfully, they showed that a TCN can be reduced in any order, i.e. there are many tree-child sequences that reduce the same network (also independently proved in [84]). This finding only goes in one direction, as the TCN that is built by a sequence is unique. This conference proceeding focused on an implementation of the algorithm which finds that the linear-time theoretical bound is achievable in practice.

The same authors, along with Mark Jones, continued this work and published another conference paper on combining networks [76]. First, they showed that the tree-child network hybridization problem can also be calculated with cherry picking. An FPT algorithm parameterized on the reticulation number of the produced network was introduced which extends the one from [99] that operates on trees. They also noted that the same leaf-attaching operation defined in [75] can be applied here to solve the network hybridization problem. Note that this network problem is NP-hard, given the tree version is proved to be so.

Janssen and Murakami then published a much larger and more comprehensive journal article on cherry picking and network containment [56]. This work elaborated deeply on network construction from cherry-picking sequences. They showed how networks are constructed with a reverse of the cherry picking operation. Differences in various construction definitions come down to if a new ordering is imposed, that is, if multifurcations or multireticulations are produced or not. Choices of these various construction definitions can be mixed and matched to produce classes of orchard networks. Furthermore, they showed that in some of these classes, a network can reliably be reconstructed from the same cherry-picking sequence that reduced it. These four classes of networks are named *reconstructible classes* and include binary networks. The reconstructible classes are the network classes closed under cherry reductions and cherry expansions. This work reiterated results from the conference paper on the same topic, giving polynomial algorithms for network containment and isomorphism for networks in the same reconstructible classes (in linear time for TCNs). Using cherry picking to show containment and isomorphism leads directly into the definition of distances with cherry picking by hinting at these sequences containing information about the relatedness of two networks. Many of the auxiliary results in this work also proved helpful in supporting our work on distances and are restated as needed in the language of this thesis.

3.2.3 Orchard networks

Recall that orchards are networks that are characterized by cherry reductions; they are the networks that can be reduced to a single leaf by cherry picking. Otherwise, they are the class that can be constructed by the reverse cherry operation. They are a relatively new class, but are promising regarding both their biological interpretation and the mathematical results found on them. The section will summarize characterizations of orchards and will shed some light on the properties and interpretations of orchards.

It has been clear from the outset that orchard networks are tantalizingly close to the class of tree-based networks. Orchards generalize TCNs, and TCN is the class where every embedded tree is a base tree. These intuitions were shown to be true in [50] which gave a natural characterization of orchards as being the class of networks that are trees with added horizontal edges. While tree-based networks are trees with added edges (assuming the addition does not induce a cycle), the edges being added to a tree in the orchard class are stipulated to be *horizontal*, an edge that happens "instantaneously" when a temporal ordering is imposed on the network vertices. In a time-consistent network, all reticulation edges must be horizontal, while in orchards one reticulation edge for each reticulate vertex does not have such a requirement, something that data may require naturally due to extinctions or undersampling. The authors show the utility of this characterization by using it to show that the class of orchards is connected under the rNNI rearrangement operation.

Recall how the μ-representation characterizes TCNs, and it was extended to characterize binary orchards (while inducing a metric on the class) by including in the path multiplicity vectors the number of paths from each vertex to reticulations as well. Orchard networks are a class that are the most generalized possible for which (it is proposed) any such representation can exist, since even a slight relaxation of the constraints of the class can lead to a graph isomorphism complete problem [88].

There was another characterization of orchards published in 2021 [103] that decomposes a network into (reticulated and non-reticulated) cherry shapes (groups of edges in a "cherry"-like configuration). If such a decomposition exists for a network, the network is shown to be tree-based. Cherry shapes in a decomposition can be ordered by their ancestry relationships in the network. The authors showed the network is an orchard if the decomposition also abides an acyclicity condition. Both of these results were then applied to non-binary orchards by generalizing the strict decomposition to a cover, which may have overlapping shapes. The authors of this work offered that characterizing orchards by cherry covers may provide more concise proofs as it may be easier to manipulate the cherry cover after certain operations than it is to adjust a sequence of cherry reductions. They also offered that cherry covers may be useful for solving enumeration problems on the cherry shapes. Unfortunately, cherry covers are not unique, they take polynomial time to find, and cannot be found naively (like cherry picking can be done, and in linear time). Furthermore, cherry

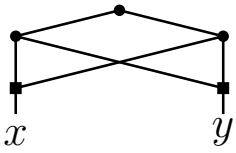

Figure 3.2: A binary phylogenetic network on $X = \{x, y\}$ that is not an orchard, there are no cherries in this network.

shapes may overlap in non-binary networks, so there is no relationship between the cover size and the size of network like there is with cherry operations (though the utility may hold for binary networks).

It has been clear thus far how we do not take for granted how information about the topology of a network or networks resides in a sequence of cherries that reduces it/them. What makes that information difficult to disentangle when it comes to comparing networks in this way is the multiplicity of possible sequences in which cherries in a network can be reduced. This next work exactly counted the number of different reducing sequences that exist for a TCN by equating it with finding the number of linear extensions [104]. A linear extension is a total order that is compatible with a partial order. Internal nodes form a set partially ordered by the ancestry relation, so the argument proceeds basically by equivocating the order of cherry reductions with an order of internal nodes of a network.

See Figure 3.2 for an example of a network that has two reticulations that are exclusive siblings, so is not an orchard. **In the work that follows, we assume networks are orchards.**

Chapter 4

Preliminary definitions

4.1 Networks

A phylogenetic X-*network*, $\mathcal{N} = (V, E, X)$, is a rooted directed acyclic graph, we write $V(\mathcal{N})$ and $E(\mathcal{N})$ to refer to its vertices and edges respectively. For $v \in V(\mathcal{N})$, we use v^- and v^+ to denote the in-degree and out-degree of v respectively.

The set $V(\mathcal{N})$ contains

- the *root* $\rho(\mathcal{N})$, which is the unique node satisfying $\rho(\mathcal{N})^- = 0$ and $\rho(\mathcal{N})^+ = 2$;
- the *leaves* $L(\mathcal{N})$, which satisfy $l^- = 1$ and $l^+ = 0$ for all $l \in L(\mathcal{N})$. This set is bijectively labeled by X. For convenience, we tend to refer to them interchangeably, with the understanding that $X = L(\mathcal{N})$;
- the *internal* vertices $V(\mathcal{N}) \setminus (L(\mathcal{N}) \cup \{\rho(\mathcal{N})\})$, which contains:
  - the *tree vertices* $T(\mathcal{N})$, which satisfy $v^- = 1$ and $v^+ = 2$ for all $v \in T(\mathcal{N})$;
  - the *reticulation vertices* $R(\mathcal{N})$, or simply *reticulations*, which satisfy $v^- = 2$ and $v^+ = 1$ for all $v \in R(\mathcal{N})$.

For a network $\mathcal{N}$, $V(\mathcal{N}) = \{\rho(\mathcal{N})\} \cup T(\mathcal{N}) \cup R(\mathcal{N}) \cup L(\mathcal{N})$.

The above conditions placed on the internal vertices mean that all networks in this work will be binary for the remainder of this document. For a binary network, the reticulation number is equal to the number of reticulation nodes, that is $r(\mathcal{N}) = |R(\mathcal{N})|$. The reticulation number of a network is a common measure of the network's complexity.

Edges are directed from the unique root to the leaves of the network. The edges of the form $(u, r)|r \in R(\mathcal{N})$ are called *reticulation edges*, denoted $E_R(\mathcal{N})$. For an edge $(u, v) \in E(\mathcal{N})$ with $v \in T(\mathcal{N}) \cup L(\mathcal{N})$, we say that u is the *parent* of v and v is a child of u, denoted $p(v) = u$. Vertices u and v are *siblings* if $p(u), p(v)$ are defined

and $p(u) = p(v)$. Generally, if there is a path from u to v, then u is an *ancestor* of v and v is a *descendant* of u.

A vertex v is *unary* if $v^- = v^+ = 1$, which we do not permit in the network. So, if needed, we *suppress* each unary vertex, as follows: add an edge from $p(v)$ to the child of v, and removing v with its incident edges.

The leaves are traditionally bijectively labeled by a set of taxa X. We start with this manner of singly labeling leaves, but will later introduce the utility of a surjective style of labeling, in which every leaf has *at least* 1 label. In both cases, all leaves are labelled and the label sets of leaves are pairwise disjoint. We refer to the cluster of a vertex v (the set of leaf labels below v) as $X(v)$.

4.2 Cherry operations

In phylogenetic trees, a cherry usually refers to two leaves that have the same parent. In networks, the authors of [56] proposed to distinguish two types of cherries. Let x, y be two distinct leaves of a network $\mathcal{N}$. We say that (x, y) is a *simple cherry* if $p(x) = p(y)$. Note that in this case, (y, x) refers to the same cherry. Analogously, we say that (x, y) is a *reticulated cherry* if $(p(y), p(x)) \in E_R(\mathcal{N})$ (i.e. $p(x) \in R(\mathcal{N})$). Note that in this case (y, x) is not a cherry. We let $\mathcal{C}_c(\mathcal{N})$ and $\mathcal{C}_r(\mathcal{N})$ denote the sets of simple and reticulated cherries of a network $\mathcal{N}$, respectively, and denote $\mathcal{C}(\mathcal{N}) = \mathcal{C}_c(\mathcal{N}) \cup \mathcal{C}_r(\mathcal{N})$.

Cherry operations, described momentarily, are graph operations that delete or add a cherry to a network. Cherry operations are often performed in batches. An ordered sequence of cherries $S = \langle (x_1, y_1), \ldots, (x_n, y_n) \rangle$ is called a *cherry sequence* (CS) and is used to define a series of cherry operations. The reverse of S is denoted $rev(S) = \langle (x_n, y_n), \ldots, (x_1, y_1) \rangle$. The concatenation of two CSs S_1 and S_2 is denoted $S_1 \cdot S_2$. CSs have hitherto in the literature been referred to as a cherry-picking sequence, but here we prefer a more general term, since CSs will refer to either network reduction operations, or network expansion operations. As a matter of language, performing a cherry operation is equivalent to saying a cherry is *applied* to a network. Not every cherry that is applied necessarily changes the network, since the operation depends on certain structures be present (or absent). As such, when equations are presented for each operation relating their number with the size of the network, they are merely bounds.

Often, the goal of applying cherry operations is to reduce a network to a minimum size. When a network is at its minimum size, we can express this special case in two different ways: the minimum network can have either one vertex (a *singleton*) or two vertices (a *single-leaf network*). We use both definitions here distinguished by this change in name, though in the original publications we used both of these terms interchangeably to mean either type of network. Chapter 5 assumes a singleton network where $|V(\mathcal{N})| = 1$, $V(\mathcal{N}) = L(\mathcal{N})$, and $\{\rho(\mathcal{N})\} = \emptyset$, and Chapter 6 assumes

a single-leaf network where $V(\mathcal{N}) = L(\mathcal{N}) \cup \{\rho(\mathcal{N})\}$. The required change in the cherry operation for this special case will be noted when introducing Chapter 6. Assume for now the minimum network is a singleton.

4.2.1 Cherry reduction

A *cherry reduction*, or *cherry picking*, is a cherry operation that modifies a network by deleting a cherry of either type. To be more specific, let $\mathcal{N}$ be a network. Then a cherry reduction on $\mathcal{N}$ of (x, y) consists of one of the following, depending on the cherry type of (x, y):

- If $(x, y) \in \mathcal{C}_c(\mathcal{N})$, then remove x and its incident edge from $\mathcal{N}$. If $p(x)$ is not the root of $\mathcal{N}$, then suppress $p(x)$. If $p(x)$ is the root of $\mathcal{N}$, then remove $p(x)$ and its incident edge. We call this a *simple reduction*.
- If $(x, y) \in \mathcal{C}_r(\mathcal{N})$, then remove the edge $(p(y), p(x))$ and suppress the two resulting unary vertices. We call this a *reticulated reduction*.
- If $(x, y) \notin \mathcal{C}(\mathcal{N})$, then the cherry reduction of (x, y) has no effect on $\mathcal{N}$. Such a reduction will be called *trivial*, and *non-trivial* otherwise.

Figure 4.1 illustrates both cases of non-trivial cherry reductions (from (a) to (b) and from (b) to (c)). For a CS S, $\mathcal{N}\langle S\rangle$ denotes the network that results from reducing the network $\mathcal{N}$ by the cherries of S in order.

Assume that a cherry sequence S reduces $\mathcal{N}$ to a singleton network. Note that cherry reductions of the first type reduce $|L(\mathcal{N})|$ by one (and one leaf alone cannot be reduced), cherry reductions of the second type reduce $r(\mathcal{N})$ by one, and as a result we have:

$$|S| \geq |L(\mathcal{N})| - 1 + r(\mathcal{N}) \tag{4.1}$$

Furthermore, because every cherry reduction removes two vertices, one that is either a leaf or a reticulation, and one that is suppressed, we get:

$$|S| \geq \frac{|V(\mathcal{N})| - 1}{2} \tag{4.2}$$

noting that every cherry reduction removes two vertices and binary networks always have an odd number of vertices. It is this latter formula that proves quite useful, later we default to a characterization of cherry distance in terms of the network size.

4.2.2 Cherry expansion

A *cherry expansion* is a cherry operation that adds a cherry leaf or a reticulation. More precisely, let $\mathcal{N}$ be a network. Then the cherry expansion on $\mathcal{N}$ of (x, y) consists of the following, depending on its type:

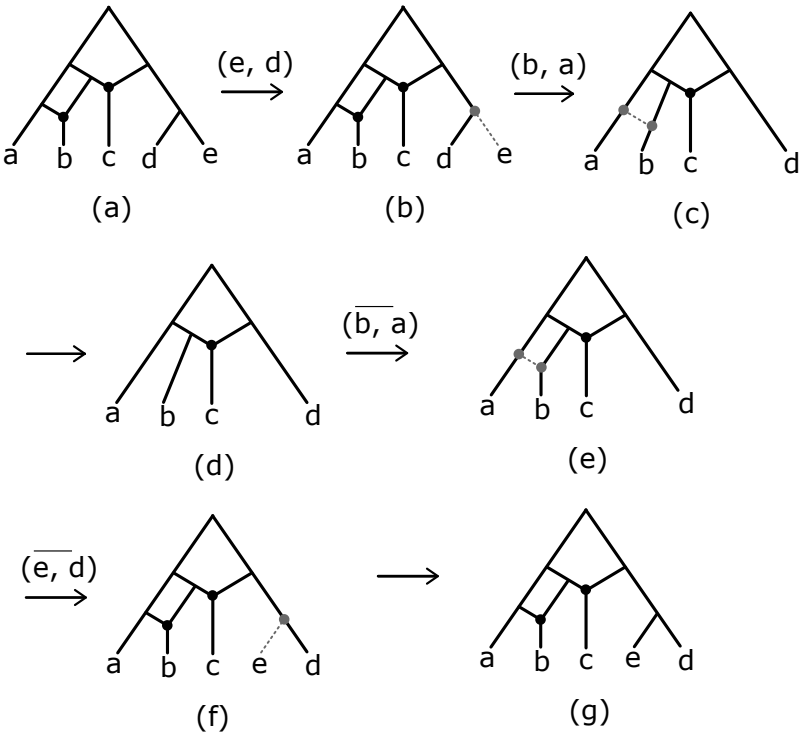

Figure 4.1: Networks (a) through (g) have reticulated vertices indicated in black. Vertices and edges targeted by an operation are shown in grey and dotted. Networks (a) through (d) show steps in the reduction of the network in (a) by a simple cherry (e,d) and a reticulated cherry (b,a). Networks (d) through (g) illustrate the cherry expansion that reverses the previous reduction operations, giving the network in (g) that is isomorphic to the network in (a).

- If $x \notin L(\mathcal{N})$ and $y \in L(\mathcal{N})$, then check if $p(y)$ exists. If so, subdivide the edge $(p(y), y)$ creating the new vertex u to which the new leaf x is to be attached by a new edge (u, x). If $p(y)$ does not exist, then $\mathcal{N}$ is a singleton network. In this case, create a new root with children x and y.

- If $x \in L(\mathcal{N})$ and $y \in L(\mathcal{N})$, subdivide the edge $(p(y), y)$ creating a vertex u, and subdivide the edge $(p(x), x)$ creating a vertex v. Then insert the new edge (u, v) so that v becomes a reticulation.

- If $y \notin L(\mathcal{N})$ then (x, y) for any $x \in L(\mathcal{N})$ or $x \notin L(\mathcal{N})$, has no effect on $\mathcal{N}$. Such an expansion will be called *trivial*, and *non-trivial* otherwise.

For a CS S, we denote by $\mathcal{N}\langle\overline{S}\rangle$ the network obtained by applying each cherry expansion in S in order (the *overline* notation emphasizes that the pairs in S must be interpreted as expansions rather than reductions). Assuming $y \in L(\mathcal{N})$, then a

cherry expansion of the first type increases $|L(\mathcal{N})|$ by one and a cherry expansion of the second type increase $r(\mathcal{N})$ by one. So for a network $\mathcal{N}$ and a CS S of cherry expansions we can relate the size of $\mathcal{N}$ before and after application of S as

$$|L(\mathcal{N}\langle\overline{S}\rangle)| + r(\mathcal{N}\langle\overline{S}\rangle) \leq |L(\mathcal{N})| + r(\mathcal{N}) + |S| \tag{4.3}$$

and

$$|V(\mathcal{N}\langle\overline{S}\rangle)| \leq |V(\mathcal{N})| + 2|S| \tag{4.4}$$

Figure 4.1 (d) to (g) demonstrates both non-trivial cherry expansions.

The cherry expansion is the reverse operation to a cherry reduction, where the class of binary orchards (orchards being the networks constructed and deconstructed by cherry operations) have been described by Janssen and Murakami as being a *reconstructible class* [56]. There are two corollaries to this feature important to our purposes. First, for a network $\mathcal{N}$ with a cherry $c \in \mathcal{C}(\mathcal{N})$, $\mathcal{N}\langle c\rangle\langle\overline{c}\rangle \simeq \mathcal{N}$ holds, which we refer to as the *reversibility of cherry picking*. Second, from any singleton network $\mathcal{N}$ where $L(\mathcal{N}) = \{x\}$, every binary orchard containing x can be reached from $\mathcal{N}$ by some sequence of cherry expansions.

Though it is not directly used in our work, it is also possible to build a network from a sequence of cherry expansions when no leaves are currently present, i.e. when there is no network to expand. To *construct* a network $\mathcal{N}$ from a CS S, take the first entry of S, say it is (x, y), and create three vertices, leaves x and y, and root vertex $\rho(\mathcal{N})$. Edges $(\rho(\mathcal{N}), y)$ and $(\rho(\mathcal{N}), x)$ are introduced such that $p(x) = p(y) = \rho(\mathcal{N})$. Expansion of $\mathcal{N}$ by S can now continue on from the second entry of S. In the same way, some versions of CSs are defined such that the last entry cherry only has the first coordinate as a member of $L(\mathcal{N})$ (the second coordinate always being '-') , which are used to build or reduce a network on one leaf.

We will refer to a CS of cherry reductions as a *reducing CS*, and refer to a CS of cherry expansions as an *expanding CS*. It might sometimes be desirable for a CS to contain both types of operations, with the overline and no-overline notation being used on a cherry-by-cherry basis to indicate the intended operation. We call a CS of this kind a *mixed CS*. For instance in Figure 4.1, the sequence from networks (a) to (g) is a mixed CS.

4.3 Complete cherry sequences and cherry-reduced subnetworks

For the remainder, we will assume that cherry operations in a CS S always affect the network, that S contains no trivial operation. Let S_i denote the sequence with the i first elements of S and let (x, y) be the $(i + 1)$-th element of S. If (x, y) is a cherry reduction, we assume that $(x, y) \in \mathcal{C}(\mathcal{N}\langle S_i\rangle)$, and if (x, y) is a cherry expansion, we

assume at least $y \in L(\mathcal{N}\langle S_i \rangle)$. If S satisfies the above property for all i, then S is called *minimal*. Under these constraints, equations 4.1,4.2, 4.3, and 4.4 are always equalities. If a reducing CS S is minimal and reduces a network $\mathcal{N}$ to a singleton network, then we say S is *complete* for $\mathcal{N}$. It is worth noting that a complete CS can always be found for an orchard. It is also known that applying a cherry reduction on an orchard results in another orchard [56, Observation 1].

Chapter 5

Project 1: Defining new cherry-based network distances

5.1 Front matter

5.1.1 Abstract

In phylogenetic networks, picking a cherry consists of removing a leaf that shares a parent with another leaf, or removing a reticulate edge whose endpoints are parents of leaves. Cherry-picking operations were recently shown to have several structural and algorithmic applications in the study of networks, for instance in determining their reconstructibility or in solving the network hybridization and network containment problems. In particular, some networks within certain classes are isomorphic if they can be reduced to a single vertex by the same sequence of cherry-picking operations. Therefore, cherry-picking sequences contain information on the level of similarity between two networks. In this paper, we expand on this idea by devising four novel distances on networks based on cherry picking and their reverse operation. We provide bounds between these distances and show that three of them are equal despite their different formulations. We also show that computing these three equivalent distances is NP-hard, even when restricted to comparing a tree and a network. On the positive side, we show that they can be computed in cubic time on two trees, providing a new comparative measure for phylogenetic trees that can be computed efficiently.

5.1.2 Dissemination

This work was first presented at the 7th-8th International Conference on Algorithms for Computational Biology (AICoB), delivered virtually in 2021. The full bibliographic entry for the conference version is below.

[105] 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*, pp. 57–81, Springer International Publishing, 2021

Following this, we were invited to publish an extended version of the work in a special edition of journal IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB). This edition includes some additional analysis of various distance properties such as metricity and diameter. This is the version presented here, and the full bibliographic entry follows.

[106] K. Landry, A. Teodocio, M. Lafond, and O. Tremblay-Savard, "Defining phylogenetic network distances using cherry operations," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 3, pp. 1654–1666, 2022

5.1.3 Contribution to authorship

My role in this research project was to make all preliminary definitions, and in conjunction with my collaborators, to define new phylogenetic network distances. I was responsible for researching and writing all proofs equating and bounding these network distances. I also formed, researched, and wrote, with collaborative input, a discussion on network diameter, including the counter-example presented, and the concluding discussion. I provided verification and feedback on all work including the NP-hardness proof. Finally, I was responsible for the majority of realizing figures and formatting.

During the writing of this manuscript, the work of Olivier Tremblay-Savard was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) grant number RGPIN-2016-06051. The work of Manuel Lafond was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) grant number RGPIN-2019-05817.

5.1.4 Manuscript modifications

Throughout this original work we use the convention of "cherry picking" to refer to the cherry reduction, including in definitions and use "CP" in acronyms. Most

notably when defining the MACRS and the CR-SUBTREE problem. In our next published work, we adjust these to use a convention of "cherry reduced" and "CR", because we are now differentiating between two cherry operations. Along these same lines, we refer to the orchard class of networks by their historical name, cherry-picking network (CPN). Additionally, at the time of this publication, we did not yet land on a name for simple cherries yet, and instead referred to them as "non-reticulated". In this version, all of these forms are updated to match the latter, preferred standard. We also keep references to the minimal network consistent with this document's definitional difference between a singleton and a single-leaf network.

Other changes to the version presented here are for the purposes of formatting and legibility. Spelling, word choice, grammar, and formatting are updated for correctness and consistency. Some references were updated in preference of citing the published article over a preprint. Next, two notation changes are made to follow the standards of our later work, that of in-degree and out-degree and the notation indicating the root of a network.

Finally, in the original publication, we claimed the runtime of the algorithm that computes some of the new distances between two trees was $O(|V(T_1)||V(T_2)|)$, but found it to be cubic ($O(n^3)$ for n the maximum of $|V(T_1)|$ and $|V(T_2)|$) upon review. This work has been updated to reflect this correction.

5.2 Introduction

5.2.1 New definitions

It is in this work that we first introduce the concept of cherry-reduced subnetworks. From a subnetwork, we expand to consider agreement between multiple networks. This notion is the cherry reduction analogue to the notion of the well-studied agreement subtrees [107] and agreement subnetworks [108].

- Given two networks $\mathcal{N}$ and $\mathcal{N}'$, we say that $\mathcal{N}'$ is a *cherry-reduced subnetwork* of $\mathcal{N}$ if there exists a reducing CS S such that $\mathcal{N}\langle S\rangle \simeq \mathcal{N}'$. If this is the case, we will write $\mathcal{N}' \subseteq_{cr} \mathcal{N}$.

- An *agreement cherry-reduced subnetwork* (ACRS) of two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ is a network $\mathcal{N}'$ satisfying $\mathcal{N}' \subseteq_{cr} \mathcal{N}_1$ and $\mathcal{N}' \subseteq_{cr} \mathcal{N}_2$.

- A *maximum agreement cherry-reduced subnetwork* (MACRS) for networks $\mathcal{N}_1$ and $\mathcal{N}_2$ is an ACRS that maximizes the total number of vertices it contains.

As we shall see, MACRS are closely related to cherry distances defined here, but it is NP-hard to decide whether $\mathcal{N}_1 \subseteq_{cr} \mathcal{N}_2$, as we show in Section 5.5. This new definition of MACRS will prove to be critical in later work and in Chapter 6 will be restated as a formal problem.

As a preliminary matter, here we state a general-purpose result that is analogous to Lemma 12 in [56]. Recall how reducing CSs with non-trivial entries can be used to determine whether two networks are isomorphic. The following shows that the complete reduction of a binary network by the complete reducing CS for another network implies the former is a subnetwork of the latter. In the case where there are complete reducing CSs for both networks that are the same, the lemma implies isomorphism between the networks by showing set equality, each input being a subnetwork of the other.

Lemma 1. *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two networks with at least two leaves, and assume that there is a CS S that is complete for both $\mathcal{N}_1$ and $\mathcal{N}_2$. Then $\mathcal{N}_1 \simeq \mathcal{N}_2$.*

Proof. We prove the statement by induction on $|S|$. As a base case, assume that $|S| = 1$. Then $S = \langle (x, y) \rangle$ for some x, y and, since $\mathcal{N}_1\langle S \rangle$ and $N_2\langle S \rangle$ are both singleton networks and (x, y) is non-trivial, it must be that $\mathcal{N}_1$ and $\mathcal{N}_2$ both consist of a tree with two leaves x and y. Hence $\mathcal{N}_1 \simeq \mathcal{N}_2$.

For the induction step, suppose that $|S| \geq 2$. Let (x, y) be the first element of S and let S' be obtained from S by removing (x, y) (so that $S = \langle (x, y) \rangle \cdot S'$). Let $\mathcal{N}_1' = \mathcal{N}_1\langle (x, y) \rangle$ and $\mathcal{N}_2' = \mathcal{N}_2\langle (x, y) \rangle$. Notice that S' is complete for both $\mathcal{N}_1'$ and $\mathcal{N}_2'$ since it reduces them to a singleton network and does not contain trivial reductions (as otherwise, S would contain trivial reductions). Thus by induction, $\mathcal{N}_1' \simeq \mathcal{N}_2'$. Now consider the (x, y) reduction. First assume that $(x, y) \in \mathcal{C}_c(\mathcal{N}_1)$. Then x is removed from $\mathcal{N}_1$ when (x, y) is applied to $\mathcal{N}_1$. Since $\mathcal{N}_1' \simeq \mathcal{N}_2'$, x cannot be a leaf of $\mathcal{N}_2'$, and since (x, y) is non-trivial with respect to $\mathcal{N}_2$, it must be that x is removed from $\mathcal{N}_2$ by the (x, y) reduction. That is, $(x, y) \in \mathcal{C}_c(\mathcal{N}_2)$ as well. It follows that $\mathcal{N}_1 \simeq \mathcal{N}_2$ because both networks are obtained from $\mathcal{N}_1'$ and $\mathcal{N}_2'$ by adding the x leaf back on the branch from y to its parent.

Assume instead that $(x, y) \in C_r(x, y)$. Then when applied on $\mathcal{N}_1$, (x, y) removes $(p(y), p(x))$ from $\mathcal{N}_1$ and thus $x \in L(\mathcal{N}_1')$. Therefore, because $\mathcal{N}_1' \simeq \mathcal{N}_2'$, applying (x, y) cannot remove x from $L(\mathcal{N}_2)$ and, since (x, y) is non-trivial with respect to $\mathcal{N}_2$, it must be that $(x, y) \in \mathcal{C}_r(\mathcal{N}_2)$ as well. It follows that $\mathcal{N}_1 \simeq \mathcal{N}_2$ because both networks are obtained from $\mathcal{N}_1'$ and $\mathcal{N}_2'$ by adding an edge from the branch above y to the branch above x. □

5.2.2 Outline

In this paper, we begin by presenting several different distances between orchards based on cherry operations in Section 5.3. We first propose novel operational distances based on the minimum number of cherry operations required to transform one network into the other, or that are required to make the networks reach a common substructure. We also introduce the tail distance, which is based on the structure of the networks' CSs and asks for a CS on each network that maximizes a common suffix. We show that three of our proposed distances are equivalent. We provide an

algorithm to calculate the three equal distances between two trees (Section 5.4), that can be computed in polynomial time, providing a novel measure between phylogenetic trees. However, we then show that they are NP-hard to compute even when comparing a network and a tree (Section 5.5). To develop our NP-hardness reductions, we also introduce the CR-SUBTREE problem, which asks whether a tree can be obtained from a network by applying reducing cherry operations. This problem may be of independent interest since it represents a new form of tree containment, a widely studied problem in the area of phylogenetic networks.

5.3 Distances based on cherry sequences

5.3.1 Definitions

This section introduces new network distances based on cherry operations and CSs. Hereafter, we assume that $\mathcal{N}_1$ and $\mathcal{N}_2$ are cherry-picking networks, not necessarily on the same set of leaves. We do assume however that $L(\mathcal{N}_1) \cap L(\mathcal{N}_2) \neq \emptyset$ since this condition ensures a finite distance, all distances are ∞ otherwise. Similarly, if one of $\mathcal{N}_1$ or $\mathcal{N}_2$ is not a cherry-picking network, then all the distances are defined to be ∞.

1. The *Mixed distance*, $d_m(\mathcal{N}_1, \mathcal{N}_2)$, for networks $\mathcal{N}_1$ and $\mathcal{N}_2$, is the length of a minimum mixed CS that transforms network $\mathcal{N}_1$ into $\mathcal{N}_2$ when its cherry operations are applied in order.

2. The *Construction distance*, $d_c(\mathcal{N}_1, \mathcal{N}_2)$, for networks $\mathcal{N}_1$ and $\mathcal{N}_2$, is the minimum combined length of a reducing CS S^- and an expanding CS S^+ where $\mathcal{N}_1$ reaches $\mathcal{N}_2$ when reduced by S^- then expanded by S^+. In other words, $d_c(\mathcal{N}_1, \mathcal{N}_2)$ is the minimum of $|S^-| + |S^+|$ over all CS pairs S^- and S^+ satisfying $\mathcal{N}_1\langle S^-\rangle\langle\overline{S^+}\rangle \simeq \mathcal{N}_2$.

3. The *Deconstruction distance*, $d_d(\mathcal{N}_1, \mathcal{N}_2)$, for networks $\mathcal{N}_1$ and $\mathcal{N}_2$, is the minimum of $|S_1| + |S_2|$ over all reducing CS pairs S_1 and S_2 satisfying $\mathcal{N}_1\langle S_1\rangle \simeq \mathcal{N}_2\langle S_2\rangle$.

4. The *Tail distance*, $d_t(\mathcal{N}_1, \mathcal{N}_2)$, for networks $\mathcal{N}_1$ and $\mathcal{N}_2$, is the minimum combined length of reducing CSs S_1 and S_2 that can both be concatenated with a common CS S to produce complete CSs for $\mathcal{N}_1$ and $\mathcal{N}_2$. In other words, $d_t(\mathcal{N}_1, \mathcal{N}_2)$ is the minimum of $|S_1| + |S_2|$ over all pairs of *complete* CSs $S_1 \cdot S$ and $S_2 \cdot S$ for $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively.

Intuitively speaking, d_m minimizes the number of operations required to turn $\mathcal{N}_1$ into $\mathcal{N}_2$, whereas d_c requires first applying reductions, followed by expansions. On the other hand, d_d minimizes the number of reductions required on both networks to

reach a common subnetwork and d_t focuses on finding complete sequences with the maximum number of common operations in the suffix.

In the rest of this section, we show how each of these distances are related. Particularly, we show that d_c, d_d and d_t are equal, but not always equal to d_m.

Theorem 1. *For any networks $\mathcal{N}_1$ and $\mathcal{N}_2$, $d_m(\mathcal{N}_1, \mathcal{N}_2) \leq d_c(\mathcal{N}_1, \mathcal{N}_2)$.*

Proof. If $d_c(\mathcal{N}_1, \mathcal{N}_2) = \infty$, there is nothing to prove, so assume otherwise. Let S^- and S^+ be CSs satisfying $\mathcal{N}_1\langle S^-\rangle\langle\overline{S^+}\rangle \simeq \mathcal{N}_2$ such that $d_c(\mathcal{N}_1, \mathcal{N}_2) = |S^-| + |S^+|$. Then $S' = S^- \cdot S^+$ is a mixed CS and applying S' on $\mathcal{N}_1$ transforms it into $\mathcal{N}_2$, implying $d_m(\mathcal{N}_1, \mathcal{N}_2) \leq d_c(\mathcal{N}_1, \mathcal{N}_2)$. □

To further elaborate on the finding of Theorem 1, Figure 5.1 shows that $d_m(\mathcal{N}_1, \mathcal{N}_2) < d_c(\mathcal{N}_1, \mathcal{N}_2)$ is possible. As illustrated, the difference between these distances can be arbitrarily large, so d_c can hardly be used to approximate d_m.

Theorem 2. *For any networks $\mathcal{N}_1$ and $\mathcal{N}_2$, $d_c(\mathcal{N}_1, \mathcal{N}_2) = d_d(\mathcal{N}_1, \mathcal{N}_2)$.*

Proof. Assume that $d_c(\mathcal{N}_1, \mathcal{N}_2) \neq \infty$ and that $d_d(\mathcal{N}_1, \mathcal{N}_2) \neq \infty$. For the first direction we show that $d_d(\mathcal{N}_1, \mathcal{N}_2) \leq d_c(\mathcal{N}_1, \mathcal{N}_2)$. Let S^- be a reducing CS and S^+ be an expanding CS that minimize $|S^-| + |S^+|$ such that $\mathcal{N}_1\langle S^-\rangle\langle\overline{S^+}\rangle \simeq \mathcal{N}_2$. Thus $d_c(\mathcal{N}_1, \mathcal{N}_2) = |S^-| + |S^+|$. By the reversibility of cherry picking $\mathcal{N}_1\langle S^-\rangle \simeq \mathcal{N}_2\langle rev(S^+)\rangle$. It follows that $d_d(\mathcal{N}_1, \mathcal{N}_2) \leq |S^-| + |rev(S^+)| = |S^-| + |S^+| = d_c(\mathcal{N}_1, \mathcal{N}_2)$.

For the second direction we show that $d_c(\mathcal{N}_1, \mathcal{N}_2) \leq d_d(\mathcal{N}_1, \mathcal{N}_2)$. Let S_1 and S_2 be reducing CSs that minimize $|S_1| + |S_2|$ such that $\mathcal{N}_1\langle S_1\rangle \simeq \mathcal{N}_2\langle S_2\rangle$. Thus $d_d(\mathcal{N}_1, \mathcal{N}_2) = |S_1| + |S_2|$. By reversibility of cherry picking, $\mathcal{N}_1\langle S_1\rangle\langle\overline{rev(S_2)}\rangle \simeq \mathcal{N}_2$. It follows that $d_c(\mathcal{N}_1, \mathcal{N}_2) \leq |S_1| + |rev(S_2)| = |S_1| + |S_2| = d_d(\mathcal{N}_1, \mathcal{N}_2)$. □

Theorem 3. *For any networks $\mathcal{N}_1$ and $\mathcal{N}_2$, $d_t(\mathcal{N}_1, \mathcal{N}_2) = d_d(\mathcal{N}_1, \mathcal{N}_2)$.*

Proof. We first show that $d_d(\mathcal{N}_1, \mathcal{N}_2) \leq d_t(\mathcal{N}_1, \mathcal{N}_2)$. We may assume that $d_t(\mathcal{N}_1, \mathcal{N}_2) \neq \infty$. Let S_1 (respectively S_2) be a reducing CS that is complete for $\mathcal{N}_1$ (respectively $\mathcal{N}_2$), such that $S_1 = S_1' \cdot S$ and $S_2 = S_2' \cdot S$ for some S of maximum size. Thus $d_t(\mathcal{N}_1, \mathcal{N}_2) = |S_1'| + |S_2'|$. Since S_1 and S_2 are complete, we know that S reduces $\mathcal{N}_1\langle S_1'\rangle$ and $\mathcal{N}_2\langle S_2'\rangle$ to a singleton network and does not contain trivial entries. By Lemma 1, $\mathcal{N}_1\langle S_1'\rangle \simeq \mathcal{N}_2\langle S_2'\rangle$. Since S_1' and S_2' are possible CSs for d_d, it follows that $d_d(\mathcal{N}_1, \mathcal{N}_2) \leq |S_1'| + |S_2'| = d_t(\mathcal{N}_1, \mathcal{N}_2)$.

We next show that $d_t(\mathcal{N}_1, \mathcal{N}_2) \leq d_d(\mathcal{N}_1, \mathcal{N}_2)$. We may assume that $d_d(\mathcal{N}_1, \mathcal{N}_2) \neq \infty$. Let S_1, S_2 be CSs such that $\mathcal{N}_1\langle S_1\rangle \simeq \mathcal{N}_2\langle S_2\rangle$ and such that $|S_1|+|S_2|$ is minimum. Then $d_d(\mathcal{N}_1, \mathcal{N}_2) = |S_1| + |S_2|$. Define $\mathcal{N}^* = \mathcal{N}_1\langle S_1\rangle$ (which is an orchard and is isomorphic to $\mathcal{N}_2\langle S_2\rangle$) and let S be a complete reducing CS for $\mathcal{N}^*$. Then $\mathcal{N}_1\langle S_1 \cdot S\rangle$ is a singleton network and $\mathcal{N}_2\langle S_2 \cdot S\rangle$ is also a singleton network. Since S is a common suffix, it follows that $d_t(\mathcal{N}_1, \mathcal{N}_2) \leq |S_1| + |S_2| = d_d(\mathcal{N}_1, \mathcal{N}_2)$. □

It has now been shown that, for networks $\mathcal{N}_1$ and $\mathcal{N}_2$, $d_t(\mathcal{N}_1, \mathcal{N}_2)$, $d_d(\mathcal{N}_1, \mathcal{N}_2)$, and $d_c(\mathcal{N}_1, \mathcal{N}_2)$ are equivalent and bounded from below by $d_m(\mathcal{N}_1, \mathcal{N}_2)$. We finish this section by establishing a relationship between these distances and agreement cherry-picking subnetworks.

Theorem 4. *Let $\mathcal{N}_1 = (V_1, E_1)$ and $\mathcal{N}_2 = (V_2, E_2)$ be networks and let $\mathcal{N}^* = (V^*, E^*)$ be an* MACRS *of $\mathcal{N}_1$ and $\mathcal{N}_2$. Then, $d_d(\mathcal{N}_1, \mathcal{N}_2) = \frac{|V_1|}{2} + \frac{|V_2|}{2} - |V^*|$.*

Proof. Let S_1 and S_2 be reducing CSs such that $\mathcal{N}_1\langle S_1\rangle \simeq \mathcal{N}^*$ and $\mathcal{N}_2\langle S_2\rangle \simeq \mathcal{N}^*$. It is clear that $d_d(\mathcal{N}_1, \mathcal{N}_2) \leq |S_1| + |S_2|$. We show that equality holds. Assume for contradiction that $d_d(\mathcal{N}_1, \mathcal{N}_2) < |S_1| + |S_2|$. Then there are S_1', S_2' such that $\mathcal{N}_1\langle S_1'\rangle \simeq \mathcal{N}_2\langle S_2'\rangle$. Thus $\mathcal{N}_1\langle S_1'\rangle$ is an ACPS of $\mathcal{N}_1$ and $\mathcal{N}_2$. However, since $|S_1'| + |S_2'| < |S_1| + |S_2|$, we must have $|S_1'| < |S_1|$ or $|S_2'| < |S_2|$ (or both). Assume without loss of generality that $|S_1'| < |S_1|$. Then $\mathcal{N}_1\langle S_1'\rangle$ has strictly more vertices than $\mathcal{N}^* = \mathcal{N}_1\langle S_1\rangle$ since fewer vertices need to be removed. This contradicts the fact that $\mathcal{N}^*$ is an MACRS. Therefore, $d_d(\mathcal{N}_1, \mathcal{N}_2) = |S_1| + |S_2|$.

Now, let S^* be a complete CS for $\mathcal{N}^*$. Then

$$
\begin{aligned}
& d_d(\mathcal{N}_1, \mathcal{N}_2) = |S_1| + |S_2| \\
=& (|S_1| + |S^*|) - |S^*| + (|S_2| + |S^*|) - |S^*| \\
=& \frac{|V_1| - 1}{2} - \frac{|V^*| - 1}{2} + \frac{|V_2| - 1}{2} - \frac{|V^*| - 1}{2} && \text{by eq. 4.2} \\
=& \frac{|V_1| - 1}{2} + \frac{|V_2| - 1}{2} - |V^*| + 1
\end{aligned}
$$

□

5.3.2 Properties of tail and mixed distances

The tail and mixed distance satisfy two conditions of a metric. The first is *identity*, isomorphic networks have a tail and a mixed distance of 0. The second is *symmetry*, proved by reversibility of cherry picking. The third condition of a metric, *the triangle inequality*, does not apply to the tail distance, as demonstrated by counterexample in Fig 5.1. What we see in this example is that, for the tail distance, N_2 serves as a "shortcut" between N_1 and N_3 that simulates a mixed distance, which is shorter than the tail distance in some cases. It follows that the tail distance is not a metric.

As for the mixed distance, the triangle inequality is indeed satisfied, making the mixed distance a metric. Take any N_1, N_2, N_3 such that $L(N_1) \cap L(N_2) \cap L(N_3) \neq \emptyset$ and find mixed cherry sequences S_{12} corresponding to a mixed distance $d_m(N_1, N_2)$, and S_{23} corresponding to $d_m(N_2, N_3)$. Since mixed distance allows any sequence of reductions/expansions, the concatenation $S_{12} \cdot S_{23}$ is a valid mixed cherry sequence that will transform N_1 to N_3 and whose length is at least $d_m(N_1, N_3)$ by definition.

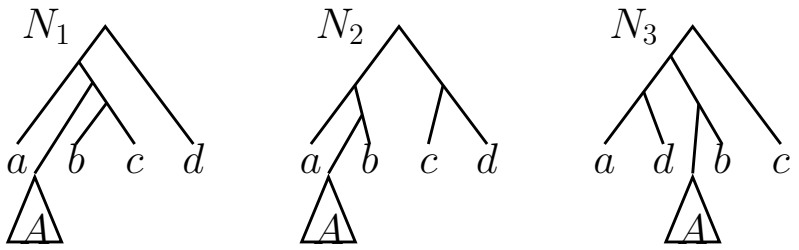

Figure 5.1: Note that $L(N_1) = L(N_2) = L(N_3) = \{a, b, c, d, L(A)\}$ where A is an unspecified but equal subnetwork in all networks. The construction distance $d_c(N_1, N_3)$ can be determined by cherry sequence $S_{13} = (c, b)S_A(b, a)(d, a)\overline{(c, a)(b, a)S'_A(d, a)}$ where S_A, S'_A reduces and expands respectively the subnetwork A (and would include one leaf outside of the subnetwork A to fully reduce/expand the subnetwork), and where all reductions come before all expansions. The mixed distance $d_m(N_1, N_2)$ can be determined by mixed cherry sequence $S_{12} = (c, b)\overline{(c, d)}$ and $d_m(N_2, N_3)$ by sequence $S_{23} = (d, c)\overline{(d, a)}$. We see that $d_m(N_1, N_2) + d_m(N_2, N_3) = d_m(N_1, N_3) = 4$ while $d_t(N_1, N_3)$ can tend arbitrarily larger depending on the size of A.

The diameter (the upper bound) of the tail distance can be determined by finding the size of the smallest possible MACRS between a network $N_1 = (V_1, E_1, X_1)$ and a network $N_2 = (V_2, E_2, X_2)$ such that $X_1 = X_2$. We will show this lower bound on the MACRS size explicitly stated in Theorem 5. It is from this theorem that we can infer the diameter using the number of vertices in the input networks along with the size of the smallest possible MACRS. In the trivial case, the MACRS($\mathcal{N}_1$,$\mathcal{N}_2$) is a singleton when $|X_1 \cap X_2| = 1$. If we impose the condition that $X_1 = X_2$, then it is not hard to show that there are networks for which the MACRS has only two leaves (e.g. caterpillars with reversed order of leaves). Conversely, we can show that any MACRS contains at least two leaves, i.e. at least a cherry remains. Although this can easily be shown to hold on trees, this is surprisingly difficult to prove on networks.

The following arguments require a new definition of a leaf subset on a network $\mathcal{N} = (V, E, X)$, in relation to a reducing CS S. For any leaf $x \in X$ and a CS $S = (S_1, \ldots, S_k)$, we define the set $subsume(x, S)$. Roughly speaking, when we delete a leaf y while it was in a cherry with x, $subsume(x, S)$ remembers that it "ate" y and all those that y ate before. To put this formally, if $|S| = 0$, define $subsume(x, S) = \{x\}$ for all $x \in X$. If $|S| > 0$, let $S' = (S_1, \ldots, S_{k-1})$. If S_k is a reticulated cherry in $\mathcal{N}$, then $subsume(x, S) = subsume(x, S')$ for all $x \in X$. If $S_k = (y, x)$ is simple, then $subsume(x, S) = subsume(x, S') \cup subsume(y, S')$, and $subsume(z, S) = subsume(z, S')$ for all $z \in X \setminus \{x\}$.

To elucidate this definition further, consider a small example. Let there be a tree T as in Figure 5.2 (a) and consider applying an empty CS E or the reducing CS $S = (b, a)(d, e)$ which produces the tree in Figure 5.2 (c). In this case, $subsume(a, E) = \{a\}$, $subsume(a, S) = \{a, b\}$, $subsume(c, E) = \{c\}$, $subsume(c, S) = \{c\}$, $subsume(e, E) =$

$\{e\}$, and $subsume(e, S) = \{d, e\}$.

Lemma 2. *Let there be network $\mathcal{N} = (V, E, X)$ and reducing CS S such that $\mathcal{N}\langle S\rangle = (V', E', X')$. Then $subsume(x, S) \cap subsume(y, S) = \emptyset$ for any distinct $x, y \in X'$ and $\bigcup_{x \in X'} subsume(x,S) = X$. That is, the subsume sets of all remaining leaves form a partition of X.*

Proof. The proof is by induction on the number of entries in S. In the base case, we have $|S| = 0$. By definition, every $l \in X$ has $subsume(l, S) = \{l\}$, and the lemma holds.

Then, in the inductive step, assume that S contains $k + 1$ elements and that the lemma holds for any CS of length k. Let S' be the CS (of length k) obtained by removing the last entry of S.

Then the $(k + 1)$-th entry of S, cherry (y, x), falls under two possibilities. In the first, (y, x) is a reticulated cherry in which case no changes are made to the *subsume* sets and the lemma holds. In the second case, we have that (y, x) is a simple cherry. Note that y is not in $\mathcal{N}\langle S\rangle$ and that each $subsume(z, S) = subsume(z, S')$, except that we have $subsume(x, S) = subsume(x, S') \cup subsume(y, S')$. By induction, no two $subsume(w, S')$ and $subsume(z, S')$ sets intersect. This clearly still holds for $subsume(w, S)$ and $subsume(z, S)$ if $w, z \neq x$. If, say, $w = x$, then by induction, $subsume(z, S')$ did not intersect either $subsume(x, S')$ nor $subsume(y, S')$, so it still does not intersect $subsume(x, S)$. Thus the intersection property holds.

As for the union property, it suffices to observe that from $\mathcal{N}\langle S'\rangle$ to $\mathcal{N}\langle S\rangle$, we removed y but $subsume(y, S')$ is added to $subsume(x, S)$, with every other *subsume* set remaining unchanged. It follows that

$$X = \bigcup_{l \in L(\mathcal{N}\langle S'\rangle)} subsume(l, S') = \bigcup_{l \in L(\mathcal{N}\langle S\rangle)} subsume(l, S)$$

.

□

Lemma 3. *For any network $\mathcal{N}$, reducing CS S of $\mathcal{N}$, leaf $x \in \mathcal{N}$ and $x' \in subsume(x, S)$, there exists some reducing CS S' such that $\mathcal{N}\langle S'\rangle$ is identical to $\mathcal{N}\langle S\rangle$ except that the leaf x is replaced by x'.*

Proof. This proof will proceed by induction on the length of the sequence S. In the base case, $|S| = 0$. In this case, $x' \in subsume(x, S)$ without any reduction, therefore $x' = x$ and the claim holds.

In the inductive step, assume the claim holds for all $|S| \leq k$, and let there be S such that $|S| = k + 1$. Consider the state of the network reduced by all but the $(k + 1)$-th, entry in S, let S inclusive up to the k-th entry be called S_k. Then, there are two cases to consider. In the first case $x' \in subsume(x, S_k)$ and the conditions of the inductive hypothesis implies there exists S', $|S'| = k$, such that x' has replaced x.

Take that S' and, if the $(k+1)$-th entry of S refers to the leaf x in the form of (l, x) for some l, append (l, x') to S', $|S'| = k+1$ and the claim holds. The last entry of S will not be (x, l) because $x \in L(\mathcal{N}\langle S\rangle)$.

In the second case $x' \notin subsume(x, S_k)$. Since we know $x' \in subsume(x, S)$, it must be that the $(k+1)$-th entry of S is the simple cherry (x', x). Take S_k as S' and append the cherry (x, x') such that $|S'| = k+1$ and the claim holds. □

Theorem 5. *For any two orchard networks $\mathcal{N}_1 = (V_1, E_1, X_1)$ and $\mathcal{N}_2 = (V_2, E_2, X_2)$ such that $|V_1|, |V_2| \geq 3$ and $X_1 = X_2$, the tail distance between them is $d_t(\mathcal{N}_1, \mathcal{N}_2) \leq \frac{|V_1|}{2} + \frac{|V_2|}{2} - 3$.*

Proof. Proving that the MACRS$(\mathcal{N}_1, \mathcal{N}_2)$ has a minimum of three vertices (a cherry network with two leaves and one root), then by referring to Theorem 4 the theorem is proved.

Consider reducing $\mathcal{N}_1$ by some reducing CS S_1 such that $\mathcal{N}_1\langle S_1\rangle$ results in a simple cherry of leaves a and x with resulting sets $subsume(a, S_1)$ and $subsume(x, S_1)$. We may assume that such a CS exists, as otherwise $\mathcal{N}_1$ is not an orchard.

The leaf a could be replaced by any $a' \in subsume(a, S_1)$ by Lemma 3, and since the two sets $subsume(a, S_1)$ and $subsume(x, S_1)$ don't intersect (by Lemma 2), the same is true of x and any $x' \in subsume(x, S_1)$. We want to argue that MACRS$(\mathcal{N}_1, \mathcal{N}_2)$ is not a singleton, i.e. has at least three vertices. If the MACRS$(\mathcal{N}_1, \mathcal{N}_2)$ were indeed a singleton, then *every* CS reducing $\mathcal{N}_2$ to a simple cherry would remove all of $subsume(a, S_1)$ or all of $subsume(x, S_1)$.

Towards a contradiction, let S_2 be a CS such that $\mathcal{N}_2\langle S_2\rangle$ is a simple cherry (again, such a CS exists if we assume that $\mathcal{N}_2$ is an orchard), and assume that, w.l.o.g., all members of $subsume(a, S_1)$ are removed by S_2. Then, the two leaves of $\mathcal{N}_2\langle S_2\rangle$ are in $subsume(x, S_1)$. Moreover, there must be some leaf x' of $\mathcal{N}_2\langle S_2\rangle$ such that $subsume(x', S_2)$ contains a (this is because of Lemma 2, which states that the *subsume* sets of $\mathcal{N}_2\langle S_2\rangle$ form a partition of X). Then by Lemma 3, there is an alternate CS S_2' that replaces x' by a. The resulting network $\mathcal{N}_2\langle S_2'\rangle$ has a leaf from $subsume(a, S_1)$, namely a, and a leaf from $subsume(x, S_1)$, because $\mathcal{N}_2\langle S_2\rangle$ is a non-singleton. This contradicts our initial assumption.

□

5.4 Computing the tail distance between two trees

In this section, we show that computing $d_t(T_1, T_2)$ between two trees T_1 and T_2 can be done in polynomial time, even if $L(T_1) \neq L(T_2)$. This is achieved by showing that computing d_t is equivalent to finding a special type of maximum agreement subtree, which can be found in time $O(n^3)$ (for n being the maximum of $|V(T_1)|$ and $|V(T_2)|$) using dynamic programming. This polynomial time solvability does not extend as we later show the problem to be NP-hard for a network-tree comparison.

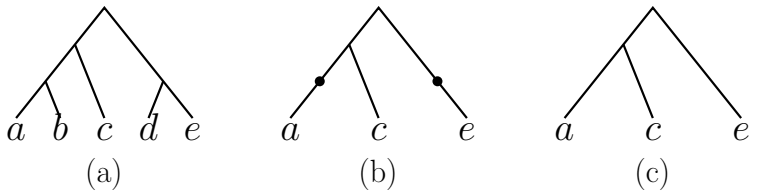

Figure 5.2: (a) A tree T on $X = \{a, b, c, d, e\}$. (b) The restriction $T|S$, where $S = \{a, c, e\}$ with unary vertices marked. (c) The restriction $T||S$, noting that this tree is a leaf-restricted subtree of the tree in (a).

We define a *tree* as a network with no reticulations. Let T be a tree. For $x, y \in L(T)$, $lca_T(x, y)$ is the lowest common ancestor of x and y, which is well-defined. Let $S \subseteq L(T)$. By $T|S$, we denote the smallest connected subgraph of T that contains each leaf in S. Note that $T|S$ may contain unary vertices. By $T||S$, we denote the tree obtained from $T|S$ after suppressing unary vertices.

Focusing on a certain restriction, for non-empty $S \subseteq L(T)$, we say that $T||S$ is a *leaf-restricted subtree* of T if either $|S| = 1$, or if both of the following hold:

1. there exist $x, y \in S$ such that $lca_T(x, y) = r(T)$;
2. if v is a unary vertex of $T|S$, then every descendant of v in $T|S$ is either a unary vertex or a leaf.

We write $T' \subseteq_{lr} T$ if T' is isomorphic to $T||L(T')$ and if $T||L(T')$ is a leaf-restricted subtree of T. Roughly speaking, unless $|L(T')| = 1$, to have $T' \subseteq_{lr} T$, Condition 1 requires $r(T)$ to be part of T', and Condition 2 requires that each removed subtree has the property that the parent of their root leads only to unary vertices and a leaf (see Figure 5.2). Note that the name "leaf-restricted" subtree was chosen to illustrate that all the unary vertices that remain after pruning are restricted to have a single leaf as a descendant. This lets us establish the connection with cherry reductions.

Given two phylogenetic trees T_1 and T_2, a *leaf-restricted agreement subtree* between T_1 and T_2 is a tree T^* such that both $T^* \subseteq_{lr} T_1$ and $T^* \subseteq_{lr} T_2$ hold. A *leaf-restricted maximum agreement subtree*, or LR-MAST for short, is a leaf-restricted agreement subtree with the maximum number of leaves. Note that since an LR-MAST T^* is a tree (without unary vertices), $|V(T^*)| = 2|L(T^*)| - 1$, so maximizing the number of vertices is the same as maximizing the number of leaves.

We first prove the transitivity of the $\subseteq_{lr}$ relationship.

Lemma 4. *If $T_1 \subseteq_{lr} T_2$ and $T_2 \subseteq_{lr} T_3$, then $T_1 \subseteq_{lr} T_3$.*

Proof. For the duration of the proof, denote $L_1 = L(T_1)$, $L_2 = L(T_2)$ and $L_3 = L(T_3)$. First notice that by the conditions of the lemma, $L_1 \subseteq L_2 \subseteq L_3$. Since T_2 is isomorphic to $T_3||L_2$, it follows that for any $X \subseteq L_2$, $T_2||X$ is isomorphic to $T_3||X$. In particular, $T_2||L_1$ is isomorphic to $T_3||L_1$, which in turn is isomorphic to T_1 (because $T_1 \subseteq_{lr} T_2$,

and thus $T_1 \simeq T_2||L_1$). It only remains to show that $T_3||L_1$ satisfies both conditions of leaf-restricted subtrees.

If $|L_1| = 1$, then this is trivially the case, so suppose that $|L_1| \geq 2$. Since $T_2||L_1$ is a leaf-restricted subtree of T_2, there must be $x, y \in L_1$ such that $lca_{T_2}(x, y) = r(T_2)$. In a similar vein, since $T_2 \subseteq_{lr} T_3$, there are $x', y' \in L_2$ such that $lca_{T_3}(x', y') = r(T_3)$. Moreover, because T_2 is isomorphic to $T_3||L_2$, $lca_{T_2}(x', y') = r(T_2)$ must hold. The fact that $lca_{T_2}(x, y) = lca_{T_2}(x', y')$ then implies that $lca_{T_3}(x, y) = r(T_3)$, and so $T_3||L_1$ satisfies Condition 1.

Now assume that $T_3||L_1$ does not satisfy Condition 2. That is, there is a unary vertex v in $T_3|L_1$ that has a descendant w with two children. Let v be the deepest vertex of $T_3|L_1$ with this property. Then we may assume that v is the parent of w in $T_3|L_1$ (and thus in T_3). Let a, b be leaves of $T_3|L_1$ such that $lca_{T_3|L_1}(a, b) = w$.

Since $L_1 \subseteq L_2$, a, b, w and v are all in $T_3|L_2$. However, v is not unary in $T_3|L_2$ because $T_2 \subseteq_{lr} T_3$. Therefore, there exists $d \in L_2$ such that $lca_{T_3}(d, a) = v$. Let $w' = lca_{T_2}(a, b)$ and $v' = lca_{T_2}(d, a)$. Since T_2 is isomorphic to $T_3||L_2$ and v is the parent of w in T_3, v' is the parent of w' in T_2. But then, v' must be unary in $T_2|L_1$, since otherwise, there would exist $d' \in L_1$ such that $lca_{T_2|L_1}(d', a)$ is the parent of $lca_{T_2|L_1}(a, b)$ in $T_2|L_1$. If this was the case, v would not be unary in $T_3|L_1$. Also, w' is not unary in $T_2|L_1$ since $a, b \in L_1$. The existence of w' and v' contradict $T_1 \subseteq_{lr} T_2$. We deduce that $T_3||L_1$ satisfies Condition 2, and therefore that $T_1 \subseteq_{lr} T_3$. □

We can now connect $\subseteq_{lr}$ with cherry sequences, as it turns out to be equivalent to the $\subseteq_{cr}$ relationship.

Lemma 5. *Let T and T' be two trees. Then $T' \subseteq_{lr} T$ if and only if $T' \subseteq_{cr} T$.*

Proof. We first note that if T' has only one vertex, it consists of a leaf of T. In that case, $T' \subseteq_{lr} T$ holds, and one can easily find C such that $T\langle C\rangle = T'$. Thus we assume that T', and thus T, has more than one leaf, and prove the two directions of the lemma separately.

($\Rightarrow$) We first show that if $T' \subseteq_{lr} T$, then $T' = T\langle C\rangle$ for some cherry sequence C. We prove the statement by induction over the quantity $k := |L(T)| - |L(T')|$. As a base case, assume that $k = 0$. Then T' is isomorphic to T and the empty reducing CS proves this case.

Now, consider the inductive step with $k > 0$. For convenience, denote $S = L(T')$, noting that T' is isomorphic to $T||S$. Assume that in T, there is a cherry $c = (x, y)$ such that $x \notin S$. Because $T\langle c\rangle$ is obtained by deleting x and suppressing $p(x)$, it is not hard to see that $T||S$ is isomorphic to $T\langle c\rangle||S$, and thus that $T' \subseteq_{lr} T\langle c\rangle$. Furthermore, we have $|L(T\langle c\rangle)| - |L(T')| = k - 1$, and we know by induction that there is a reducing CS transforming $T\langle c\rangle$ into T', hence a cherry reduction from T to T'.

So suppose that no cherry c as described above exists. That is, for every cherry (x, y) of T, both $x, y \in S$. We know that $S \neq L(T)$ since $k > 0$, so there exists some

leaf $x \in V(T) \setminus S$. By our assumption, x cannot belong to a cherry of T, and so the child w of $p(x)$ other than x must be an internal vertex of T. Thus, there is a cherry (y, z) in T that belongs to the subtree induced by w and its descendants. By our assumption, $y, z \in S$, and it follows that the parent w' of y and z in T must be in $T|S$. But then, by the first condition of leaf-restricted subtrees, $r(T)$ must be in $T|S$. Moreover, $p(x)$ must be in $T|S$ since it lies on the path between the root and w'. However, $p(x)$ has only one child in $T|S$ since $x \notin S$, and $p(x)$ has descendant w' with two children in $T|S$. This contradicts the second condition of the definition of $T' \subseteq_{lr} T$, and so this case does not occur. We deduce that there does exist a CS from T to T'.

($\Leftarrow$) We now show that if there exists a CS $C = (c_1, \ldots, c_k)$ such that $T' \simeq T\langle C\rangle$, then $T\langle C\rangle \subseteq_{lr} T$. In fact, by Lemma 4, it suffices to show that if T' is obtained from T after applying one cherry reduction, then $T' \subseteq_{lr} T$. The statement will follow, since if T_i denotes the i-th tree after applying the i-th cherry reduction of C, this shows that $T_k \subseteq_{lr} T_{k-1} \subseteq_{lr} \ldots \subseteq_{lr} T_1 \subseteq_{lr} T$, and then $T_k \subseteq_{lr} T$ by transitivity (Lemma 4).

Therefore, assume that $T' = T\langle c\rangle$ for some cherry $c = (x, y)$. Then from T to T', we remove x and suppress its parent $p(x)$. By defining $S = L(T) \setminus \{x\}$, we have that $T\langle c\rangle \simeq T||S$. Moreover, x could not be a child of the root since otherwise, c would not be a cherry (recall that $|L(T')| > 1$). This implies that $r(T)$ is in $T||S$, satisfying Condition 1. Moreover, only $p(x)$ can be unary in $T|S$ and Condition 2 is satisfied since $p(x)$ has two children leaves. Therefore, $T\langle c\rangle \subseteq_{lr} T$, as desired. □

Next, we show that d_t can be computed easily if an LR-MAST T^* is known.

Theorem 6. *Let T_1, T_2 be two trees, and let T^* be an LR-MAST of T_1 and T_2. Then $d_t(T_1, T_2) = (|V(T_1)| + |V(T_2)|)/2 - |V(T^*)|$.*

Proof. Let us denote $n_1 := |V(T_1)|, n_2 = |V(T_2)|$ and $n^* = |V(T^*)|$. Let $C_1 = (a_1, \ldots, a_k)$ and $C_2 = (b_1, \ldots, b_h)$ be reducing CSs to apply on T_1 and T_2, respectively, so that $T_1\langle C_1\rangle$ and $T_2\langle C_2\rangle$ are isomorphic, and such that $k + h$ is minimum among all possibilities. By Theorem 3, we know that $d_t(T_1, T_2) = k + h$. Furthermore, by Lemma 5, we know that $T_1\langle C_1\rangle \subseteq_{lr} T$ and $T_2\langle C_2\rangle \subseteq_{lr} T$, and thus $T_1\langle C_1\rangle$ is a leaf-restricted agreement subtree of T_1 and T_2. Denote $\hat{n} = |V(T_1\langle C_1\rangle)|$ (which is equal to $|V(T_2\langle C_2\rangle)|$). Notice that $\hat{n} \leq n^*$ because T^* maximizes the number of vertices among all leaf-restricted agreement subtrees. Also note that each cherry reduction removes two vertices, namely a leaf and its parent. Therefore, $T_1\langle C_1\rangle$ has $n_1 - 2k$ vertices and $T_2\langle C_2\rangle$ has $n_2 - 2h$ vertices, i.e. $\hat{n} = n_1 - 2k = n_2 - 2h$. Isolating k and h, it follows that

$$\begin{aligned} d_t(T_1, T_2) = k + h &= (n_1 - \hat{n})/2 + (n_2 - \hat{n})/2 \\ &= (n_1 + n_2)/2 - \hat{n} \\ &\geq (n_1 + n_2)/2 - n^* \end{aligned}$$

This proves the first bound of the proof. Consider now the complementary bound. By Lemma 5, since $T^* \subseteq_{lr} T_1$ and $T^* \subseteq_{lr} T_2$, there is a reducing CS C_1 from T_1 to T^*, and another reducing CS C_2 from T_2 to T^*. Because a cherry reduction removes two vertices, C_1 must contain $(n_1 - n^*)/2$ elements, and C_2 must contain $(n_2 - n^*)/2$ elements. Since $T_1\langle C_1\rangle$ and $T_2\langle C_2\rangle$ are isomorphic, it follows that $d_t(T_1, T_2) \leq (n_1 - n^*)/2 + (n_2 - n^*)/2 = (n_1 + n_2)/2 - n^*$. This concludes the proof. □

5.4.1 Computing an LR-MAST

Because maximizing $|V(T^*)|$ is equivalent to maximizing $|L(T^*)|$, Theorem 6 shows that, on trees, computing $d_t(T_1, T_2)$ is equivalent to computing an LR-MAST. This can be achieved using a dynamic programming procedure, described below, which leads to the following, proved in this section.

Theorem 7. *For two given trees T_1 and T_2, $d_t(T_1, T_2)$, and therefore $d_c(T_1, T_2)$ and $d_d(T_1, T_2)$, can be computed in time $O(n^3)$ where n is the larger of $|V(T_1)|$ and $|V(T_2)|$.*

For a tree T and $v \in V(T)$, $T[v]$ denotes the subtree induced' by v and all of its descendants. We may write $L(v)$ instead of $L(T[v])$.

For $u \in V(T_1)$ and $v \in V(T_2)$, $M[u, v]$ denotes the number of leaves in an LR-MAST between $T_1[u]$ and $T_2[v]$. The idea is to take an LR-MAST on both sides of u and v and combine them into a larger LR-MAST, when possible. This is similar to the classical MAST algorithm, but with fewer cases to check because of the additional restrictions. We now describe the recurrence for M.

Leaf case. If u or v is a leaf, then $M[u, v] = 1$ if $L(u) \cap L(v) \neq \emptyset$, and $M[u, v] = 0$ otherwise.

Internal vertex case. Assume that u and v are both internal vertices. Let u_1, u_2 be the children of u, and v_1, v_2 be the children of v. We compute two temporary values $M_1[u, v]$ and $M_2[u, v]$, and take the maximum of the two. More precisely, apply the recurrence definition in Figure 5.3.

Roughly speaking, M_1 corresponds to mapping u_1 to v_1 and u_2 to v_2, and M_2 corresponds to mapping u_1 to v_2 and u_2 to v_1. We now show that these recurrences are correct.

Theorem 8. *$M[u, v]$ as defined above is the number of leaves in an LR-MAST between T_1 (rooted at u) and T_2 (rooted at v).*

Proof. We proceed by induction over the height of the trees $T_1[u]$ and $T_2[v]$. As a base case, assume that one of u or v is a leaf, say u without loss of generality. If $L(u) \cap L(v) \neq \emptyset$, then $u \in L(v)$ and the tree consisting of only u is an LR-MAST, hence $M[u, v] = 1$. Otherwise, there is no leaf in common, there cannot exist an LR-MAST, and $M[u, v] = 0$ is correct.

$$M_1[u,v] = \begin{cases} 0 & \text{if } L(u_1) \cap L(v_1) = \emptyset \text{ and } L(u_2) \cap L(v_2) = \emptyset \\ 1 & \text{if } L(u_1) \cap L(v_1) \neq \emptyset \text{ and } L(u_2) \cap L(v_2) = \emptyset \\ 1 & \text{if } L(u_1) \cap L(v_1) = \emptyset \text{ and } L(u_2) \cap L(v_2) \neq \emptyset \\ M[u_1,v_1] + M[u_2,v_2] & \text{if } L(u_1) \cap L(v_1) \neq \emptyset \text{ and } L(u_2) \cap L(v_2) \neq \emptyset \end{cases}$$

$$M_2[u,v] = \begin{cases} 0 & \text{if } L(u_1) \cap L(v_2) = \emptyset \text{ and } L(u_2) \cap L(v_1) = \emptyset \\ 1 & \text{if } L(u_1) \cap L(v_2) \neq \emptyset \text{ and } L(u_2) \cap L(v_1) = \emptyset \\ 1 & \text{if } L(u_1) \cap L(v_2) = \emptyset \text{ and } L(u_2) \cap L(v_1) \neq \emptyset \\ M[u_1,v_2] + M[u_2,v_1] & \text{if } L(u_1) \cap L(v_2) \neq \emptyset \text{ and } L(u_2) \cap L(v_1) \neq \emptyset \end{cases}$$

and put

$$M[u,v] = \max(M_1[u,v], M_2[u,v])$$

Figure 5.3: Recurrence equation defining M.

Now assume that u and v are internal vertices. Let T^* be an LR-MAST between $T_1[u]$ and $T_2[v]$, and let S be such that T^* is isomorphic to $T_1[u]||S$ and $T_2[v]||S$. First assume that $S = \emptyset$, i.e. T^* has no vertex. Then it must be that $L(u) \cap L(v) = \emptyset$. If follows that $L(u_i) \cap L(v_j) = \emptyset$ for all $i, j \in 1, 2$, in which case the recurrence specifies $M_1[u,v] = M_2[u,v] = 0$. Thus $M[u,v] = 0$ will be correctly set.

Assume now that $|S| = 1$. Then $L(u) \cap L(v) \geq 1$. Thus one of $L(u_1) \cap L(v_1), L(u_1) \cap L(v_2), L(u_2) \cap L(v_1)$ or $L(u_2) \cap L(v_2)$ is non-empty, implying that one or both of $M_1[u,v]$ or $M_2[u,v]$ is at least 1. Moreover, one of $L(u_1) \cap L(v_1)$ and $L(u_2) \cap L(v_2)$ must be empty, as otherwise a leaf-restricted agreement subtree with two leaves can be found, contradicting $|S| = 1$. Similarly, one of $L(u_1) \cap L(v_2)$ and $L(u_2) \cap L(v_1)$ must be empty. It follows that $M_1[u,v] \leq 1$ and $M_2[u,v] \leq 1$, and that $\max(M_1[u,v], M_2[u,v]) = 1$. Thus $M[u,v] = 1$ will be correctly set.

Now assume that $|S| \geq 2$. We first argue that $M[u,v] \geq |L(T^*)|$. Let $U_1 = L(u_1) \cap S$ and $U_2 = L(u_2) \cap S$. Similarly, let $V_1 = L(v_1) \cap S$ and $V_2 = L(v_2) \cap S$. By the Condition 1 of leaf-restricted subtrees, S must contain leaves a, b such that $lca_{T_1[u]}(a,b) = u$, and leaves a', b' such that $lca_{T_2[v]}(a',b') = v$. This implies that none of U_1, U_2, V_1, V_2 is empty. We have that $T_1[u]||S$ contains u and $T_2[v]||S$ contains v, and since both subtrees are isomorphic, it follows that either:
(1) $T_1[u_1]||U_1$ is isomorphic to $T_2[v_1]||V_1$ and $T_1[u_2]||U_2$ is isomorphic to $T_2[v_2]||V_2$; or that
(2) $T_1[u_1]||U_1$ is isomorphic to $T_2[v_2]||V_2$ and $T_1[u_2]||U_2$ is isomorphic to $T_2[v_1]||V_1$.

Assume that (1) holds. Then $L(u_1) \cap L(v_1) \neq \emptyset$ and $L(u_2) \cap L(v_2) \neq \emptyset$, so the $M_1[u,v]$ entry will be set by the fourth case in the recurrence. We now argue that $T_1[u_1]||U_1$ is a leaf-restricted agreement subtree of $T_1[u_1]$. If $|U_1| = 1$, this trivially holds, so assume $|U_1| \geq 2$. We prove that both conditions of leaf-restricted subtrees

hold:

- Condition 1. If there are no $a, b \in U_1$ such that $lca_{T_1[u_1]}(a, b) = u_1$, then all members of U_1 are below one child of u_1. This means that in $T_1[u]||S$, u_1 is unary, but has a descendant of degree 2 since $|U_1| \geq 2$, a contradiction to the fact that $T_1[u]||S$ is a leaf-restricted subtree. Thus Condition 1 holds for $T_1[u_1]||U_1$.

- Condition 2. There cannot be a unary vertex w with a binary descendant in $T_1[u_1]||U_1$, as otherwise w would also be unary in $T_1[u]||U_1$ and would also have a binary descendant.

It follows that $T_1[u_1]||U_1$ is a leaf-restricted subtree of $T_1[u_1]$. By the same reasoning, $T_2[v_1]||V_1$ is a leaf-restricted subtree of $T_2[v_1]$. And since $T_1[u_1]||U_1$ and $T_2[v_1]||V_1$ are isomorphic, they are both leaf-restricted agreement subtrees of $T_1[u_1]$ and $T_2[v_1]$. By symmetry, $T_1[u_2]||U_2$ is a leaf-restricted agreement subtree of $T_1[u_2]$ and $T_2[v_2]$.

We do not know if $T_1[u_1]||U_1$ and $T_1[u_2]||U_2$ maximize the number of vertices among all agreement subtrees, but, by induction, we know that $M[u_1, v_1] \geq |U_1|$ and $M[u_2, v_2] \geq |U_2|$, and thus

$$M[u, v] \geq M[u_1, v_1] + M[u_2, v_2] \geq |U_1| + |U_2| = |L(T^*)|$$

If (2) holds, a similar reasoning shows that $M[u, v] \geq M[u_1, v_2] + M[u_2, v_1] \geq |U_1| + |U_2|$, and again $M[u, v] \geq |L(T^*)|$.

We now show that $M[u, v] \leq |L(T^*)|$. Suppose that $M_1[u, v] \geq M_2[u, v]$, so that $M[u, v] = M_1[u, v]$ (the case $M_2[u, v] \geq M_1[u, v]$ is identical). If $M_1[u, v] \leq 1$, then $M[u, v] = M_1[u, v] \leq 2 \leq |L(T^*)|$, as desired.

Thus assume $M_1[u, v] \geq 2$. This is only possible if $L(u_1) \cap L(v_1)$ and $L(u_2) \cap L(v_2)$ are non-empty. It follows that $M[u_1, v_1] > 0$ and $M[u_2, v_2] > 0$. By induction, there exist $A \subseteq L(u_1) \cap L(v_1)$ such that $T_1[u_1]||A$ is an LR-MAST of $T_1[u_1]$ and $T_2[v_1]$ with $|A| = M[u_1, v_1]$ leaves, and there exists $B \subseteq L(u_2) \cap L(v_2)$ such that $T_1[u_2]||B$ is an LR-MAST of $T_1[u_2]$ and $T_2[v_2]$ with $|B| = M[u_2, v_2]$ leaves. To ease notation, write

$$T_A = T_1[u_1]||A \quad \text{and} \quad T_B = T_1[u_2]||B$$

Consider the tree T' obtained by joining the roots of T_A and T_B under a common parent. Note that T' has $M[u_1, v_1] + M[u_2, v_2]$ leaves. We want to argue that $T' \subseteq_{lr} T[u]$ and $T' \subseteq_{lr} T[v]$.

By this induction, it is clear that T' is isomorphic to $T_1[u]||(A \cup B)$ and to $T_2[v]||(A \cup B)$. Thus it suffices to show that T' satisfies the conditions of leaf-restricted subtrees. Since T_A and T_B are non-empty, T' contains a leaf from both sides of u and v and Condition 1 of leaf-restricted subtrees is satisfied. Consider Condition 2. All the vertices of T' have the same number of children as in T_A or T_B, except the root of T' which is not present in either tree. This root has two children, so we could not

have created a new unary vertex with a binary descendant. Therefore Condition 2 is also satisfied.

We deduce that T' is a leaf-restricted agreement subtree of $T_1[u]$ and $T_2[v]$, and thus $|L(T')| \leq |L(T^*)|$. We therefore have

$$M[u,v] = M[u_1,v_1] + M[u_2,v_2] = |L(T')| \leq |L(T^*)|$$

as desired.

The case $M_2[u,v] \geq M_1[u,v]$ can be handled in the same manner by symmetry.

We have therefore shown that $|L(T^*)| \leq M[u,v] \leq |L(T^*)|$, proving equality. □

The value of interest here is $M[r(T_1), r(T_2)]$. It can be obtained by calculating $M[u,v]$ for each $u \in V(T_1)$ and each $v \in V(T_2)$ in a bottom-up fashion. Let n be the maximum of $|V(T_1)|$ and $|V(T_2)|$. Then the calculation of each $M[u,v]$ entry can be done in time $O(n)$: While filling the value and the table lookup (assuming the entry is pre-filled) is constant time, there are a constant number of set intersections ($O(n)$) that may be done to decide which case applies. Thus, the total time required to fill M is $O(n^3)$.

5.5 NP-hardness of tail distance between a tree and a network

We show that computing $d_t(T,N)$ between a tree T and a network N is NP-hard, holding even when $L(T) \subset L(N)$. In fact, we show something stronger, that deciding whether $T \subseteq_{cr} N$ is NP-hard. We will show in Theorem 10 that this implies that computing d_t is NP-hard.

The CR-SUBTREE problem.
Input: a tree T and a network N such that $L(T) \subseteq L(N)$.
Question: is $T \subseteq_{cr} N$?

Roughly speaking, the hardness of CR-SUBTREE implies the hardness of computing d_t since, given a network N and a tree T, the smallest d_t one can hope for is $|V(N) - V(T)|/2$. This is because we need to at least make N and T have the same number of vertices, and each cherry reduction affects two vertices. Furthermore, we can attain this distance if and only if $T \subseteq_{cr} N$, establishing the equivalence between the two problems.

Now we show the CR-SUBTREE is NP-hard with a reduction from the 3-SAT-3-OCC problem, known to be NP-hard(see [109] and [110, Problem 1]). This version of 3-SAT gives sets of clauses, each of which contains either 2 or 3 literals related by logical *or* denoted $\vee$. Each variable occurs exactly three times as a literal, twice positively and once negatively (a literal is a boolean variable x_i or its negation $\overline{x}_i$).

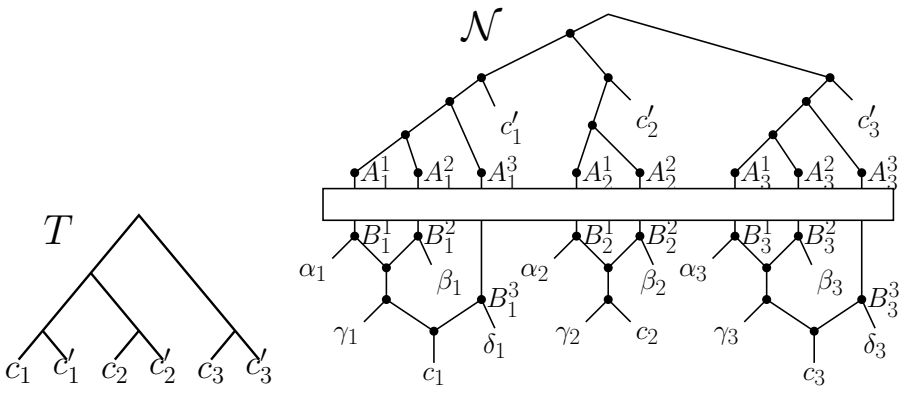

Figure 5.4: The main structure of T and N obtained from an instance with three clauses C_1, C_2, C_3. Here, N is obtained from T by replacing each leaf c_i by a substructure. In this example, C_1 and C_3 have 3 literals whereas C_2 has 2 literals. Not shown: children of the A_i^j vertices and parents of the B_j^i vertices, which make the connections between clauses and variables.

Examples of clauses are $C_1 = (x_1 \vee \overline{x}_2)$ or $C_2 = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$. The goal is to decide if an assignment of the x_i variables exists satisfying every clause.

Let ϕ be a 3-SAT-3-OCC instance, with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. We construct a corresponding tree T and network N. Let T be any tree with $2m$ leaves and m cherries, each cherry representing a clause C_i. Name the cherry's leaves that correspond to C_i by c_i and c'_i. Obtain a network N by starting at T, then replacing each leaf c_i by a substructure as in Figure 5.4.

More precisely, from T, replace each leaf c_i as follows, depending on the number of literals of clause C_i (we will specify how the A_i^j vertices are connected to the B_i^j vertices later):

- If C_i has 2 literals, replace leaf c_i by a tree with 2 leaves A_i^1, A_i^2. Add vertices B_i^1, B_i^2 and leaves $\alpha_i, \beta_i, \gamma_i, c_i$ as in the middle substructure in Figure 5.4.
- If C_i has 3 literals, replace leaf c_i by a tree with 3 leaves A_i^1, A_i^2, A_i^3. Add vertices B_i^1, B_i^2, B_i^3 and leaves $\alpha_i, \beta_i, \gamma_i, \delta_i, c_i$ as in the left or right substructure in Figure 5.4.

Conceptually, A_i^1, B_i^1 represent the first literal that can satisfy clause C_i, A_i^2, B_i^2 represent the second literal and A_i^3, B_i^3 the third literal (if present).

We can now specify the connections between the A_i^j's and B_i^j's. Consider a variable x_h and recall that it has two positive occurrences and one negative. Assume that x_h occurs positively in clauses C_i and C_j and negatively in clause C_k. Then there are two pairs of vertices A_i^a, B_i^a and A_j^b, B_j^b representing the positive occurrences x_h, and one

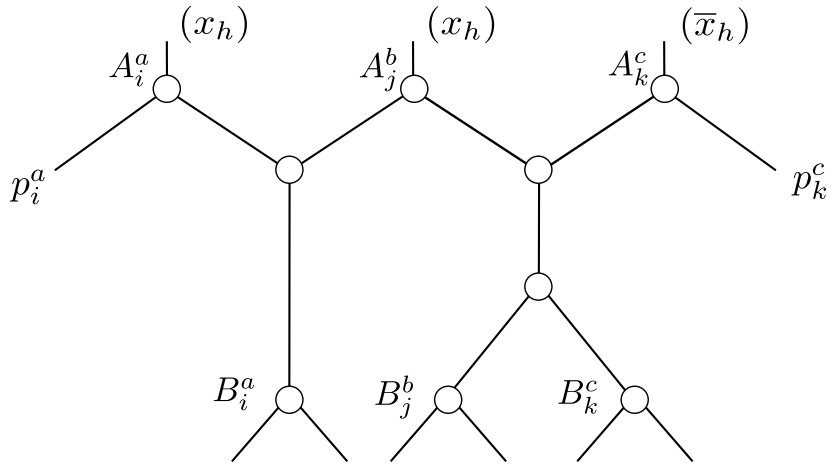

Figure 5.5: The x_h gadget, exhibiting the relationship between the A and B vertices corresponding to the occurrences of x_h. Here, A_i^a and B_i^a are associated with choosing x_h to satisfy C_i; A_j^b, B_j^b are associated with choosing x_h to satisfy C_j; and A_k^c, B_k^c are associated with choosing $\overline{x}_h$ to satisfy C_k.

pair of vertices A_k^c, B_k^c representing the negative occurrence $\overline{x}_h$, with $a, b, c \in \{1, 2, 3\}$. Add vertices and edges between these vertices as in Figure 5.5, thereby creating two new leaves p_i^a and p_k^c. We will refer to the subgraph illustrated in Figure 5.5 as the x_h gadget. We construct this gadget for each variable x_h. After this, since each A_i^a, B_i^a pair represents a distinct literal, each A_i^a vertex has exactly two children and each B_i^a vertex has exactly one parent.

This concludes the construction. The main intuition is that we need to get rid of dummy leaves to turn N into T. To achieve this, each c_i leaf will need to be "brought up" to its c_i' sibling by eliminating the substructures of N. We must thus choose a path for each c_i, either through A_i^1, A_i^2 or A_i^3 (if present). Such a choice will correspond to choosing a literal to satisfy each C_i, and the x_h gadgets prevent us from choosing two paths/literals that are contradictory.

Theorem 9. *The CR-SUBTREE problem is NP-hard.*

Proof. Let ϕ be an instance of 3-SAT-3-OCC. Let T and N be constructed as described above. For the remainder of the proof, a leaf in $L(N) \setminus L(T)$ will be called a *dummy* leaf. We show that ϕ is satisfiable if and only if there exists a cherry sequence C such that $N\langle C\rangle$ is isomorphic to T.

($\Rightarrow$) : suppose that ϕ is satisfiable by some assignment Q of the x_i variables. We show that there is a cherry sequence that transforms N into T. Consider a clause C_i. Then Q satisfies the j-th literal of C_i for some $j \in \{1, 2, 3\}$. If there are multiple choices for j, choose one arbitrarily. Depending on j, apply one of the cherry sequences appearing in Figure 5.6. The figure assumes a clause C_i with three literals. If C_i has two literals, then δ_i and B_i^3 do not exist, c_i is a sibling of γ_i and we have

$j \in \{1, 2\}$. For such a clause, the cases for $j = 1$ or $j = 2$ can be applied as in Figure 5.6, ignoring the first (c_i, δ_i) move.

As a result, for each $i \in \{1, \ldots, m\}$, B_i^j becomes replaced by c_i when Q satisfies the j-th literal of C_i. Moreover, each $B_i^{j'}, j' \neq j$, has been replaced by a dummy leaf.

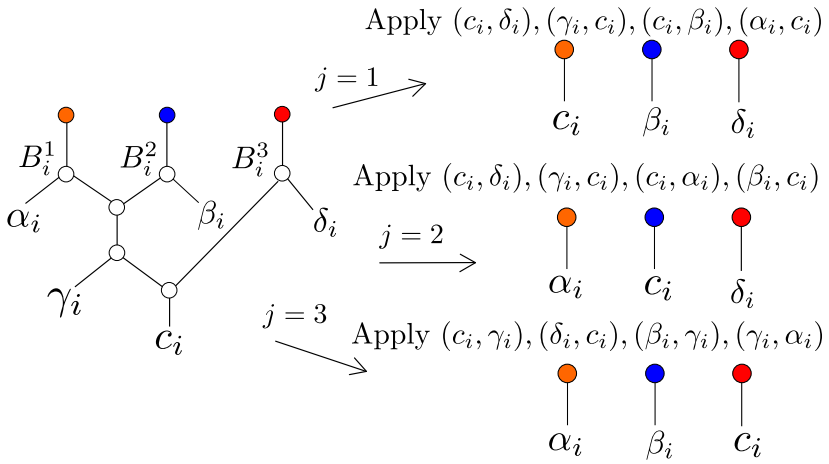

Figure 5.6: The three possible ways to handle leaf c_i, depending on whether we satisfy clause C_i with its first, second or third literal, respectively corresponding to $j = 1, j = 2$ and $j = 3$. The colors on the vertices are there to emphasize which vertex receives c_i as a child after the cherry reduction sequence.

Next, consider a variable x_h. Assume that x_h occurs positively in clauses C_i and C_j and negatively in clause C_k. After applying the above cherry sequence for each clause, the x_h gadget has three leaves that have replaced the B_i^a, B_j^b and B_k^c vertices. From this point, we distinguish three possible cases:
(1) none of c_i, c_j or c_k is a leaf of the x_h gadget;
(2) c_i and/or c_j is a leaf of the x_h gadget. In this case, c_k cannot be a leaf of the x_h gadget, since this would imply that assignment Q assigns x_h positively and negatively. Indeed, in our described sequence B_i^a gets replaced by c_i when $x_h = true$ satisfies clause C_i, whereas B_k^c gets replaced by c_k when $x_h = false$. The same holds for B_j^b versus B_k^c.
(3) c_k is a leaf of the x_h gadget. In this case, none of c_i and c_j is a leaf of the x_h gadget because again, Q would otherwise assign x_h both positively and negatively.

The handling of cases (2) and (3) is shown in Figure 5.7, top and bottom respectively.

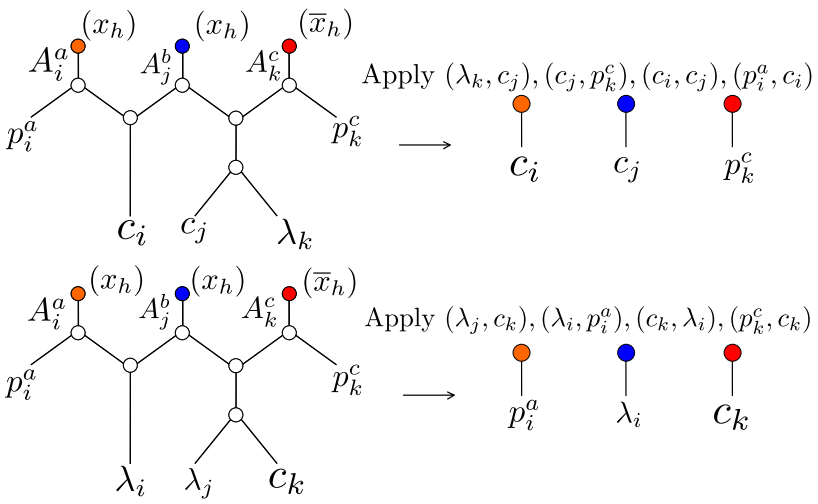

Figure 5.7: Handling of Case 2 (top) and Case 3 (bottom) for the x_h gadget. Note that $\lambda_i, \lambda_j, \lambda_k$ are used to denote dummy leaves. Also, on top, one of c_i or c_j could be a dummy leaf.

The handling of Case 1 can be performed as in Case 2, except that c_i and c_j are dummy leaves instead. The same applies in Case 2 when one of c_i or c_j is not present.

After applying this to every x_h gadget, each A_i^j vertex of N is replaced by either c_i or some dummy leaf. Importantly, notice that in all cases, every c_i leaf ends up replacing an A_i^j vertex, $j \in \{1, 2, 3\}$, and not some A_k^j leaf with $k \neq i$. In other words, N has become a tree such that for every $i \in \{1, \ldots, m\}$, each of A_i^1, A_i^2, and A_i^3 if present, has become a leaf, one of which is c_i (and the others are dummy leaves). It follows that at this point, N has become a tree identical to T but with extra dummy leaves. It is then easy to see that these can be removed by cherry reduction moves to obtain T. Indeed, for each clause C_i, if C_i has two variables, A_i^1, A_i^2 are replaced by leaves that form a cherry containing c_i and a dummy leaf. We simply pick the dummy leaf and make c_i, c_i' siblings. If C_i has three variables, A_i^1, A_i^2 are replaced by leaves that form a cherry. At least one of these leaves is a dummy, so we pick it. After that, we are left with a cherry with the leaf that contains A_i^3 and again, we pick whichever is not c_i, making c_i and c_i' siblings. Therefore, $T \subseteq_{cr} N$.

($\Leftarrow$) : suppose that there exists a cherry sequence C that transforms N into T. We build a satisfying assignment Q for ϕ. The assignment is constructed by adding literals to Q — our goal is to add one literal per clause, without adding two contradictory literals x_h and $\overline{x}_h$.

Consider a c_i leaf of N, and let L_i be the literals that can satisfy clause C_i. If C_i has 2 literals, let $L_i = \{\ell_1, \ell_2\}$, and if C_i has 3 literals, let $L_i = \{\ell_1, \ell_2, \ell_3\}$. Let X be the first cherry reduction move of C that contains c_i (which must exist since otherwise, c_i would remain the child of a reticulation). We consider all possible cases.

1. If C_i has 2 literals, then c_i is a sibling of γ_i. In that case, go to Case 3 below.

2. If C_i has 3 literals, then either $X = (c_i, \gamma_i)$ or $X = (c_i, \delta_i)$. If $X = (c_i, \gamma_i)$, then add ℓ_3 to Q. In this case, notice that the next cherry reduction affecting c_i must delete δ_i, and that B_i^3 gets replaced by c_i.

   If instead $X = (c_i, \delta_i)$, then c_i becomes a sibling of γ_i. Then go to Case 3 below.

3. If c_i is a sibling of γ_i, notice that the next cherry reduction affecting c_i must delete γ_i. After γ_i is deleted, consider the next cherry reduction X' that contains c_i. Whether C_i has 2 or 3 variables, it must be that either $X' = (c_i, \alpha_i)$ or $X' = (c_i, \beta_i)$.

   - If $X' = (c_i, \alpha_i)$, then add ℓ_2 to Q. In this case, notice that the next cherry reduction of C affecting c_i must delete β_i, and that B_i^2 gets replaced by c_i.
   - If $X' = (c_i, \beta_i)$, then add ℓ_1 to Q. In this case, notice that the next cherry reduction of C affecting c_i must delete α_i, and that B_i^1 gets replaced by c_i.

It follows that in all cases, we add a literal ℓ_j to Q satisfying C_i. Thus Q satisfies all the clauses of ϕ. It remains to show that we have not added two contradictory literals x_h and $\overline{x}_h$.

Let us assume that we did add both x_h and $\overline{x}_h$. Assume that x_h occurs positively in clauses C_i and C_j and negatively in clause C_k. Let A_i^a, B_i^a and A_j^b, B_j^b correspond to the occurrences of x_h, and A_k^c, B_k^c correspond to $\overline{x}_h$. By our construction of Q, we know that $\overline{x}_h$ was added to Q because c_k ended up replacing B_k^c. Moreover, since x_h was added to Q, then c_i replaced B_i^a or c_j replaced B_j^b (or both). Now, notice that the next cherry reduction affecting c_k must be with the leaf that eventually replaces B_j^b, by the construction of the x_h gadget. Therefore, c_j cannot end up replacing B_j^b, since otherwise the cherry sequence C would have to remove one of c_j or c_k, preventing it from transforming N into T. Therefore, we may assume that x_h was added to Q because c_i ended up replacing B_i^a.

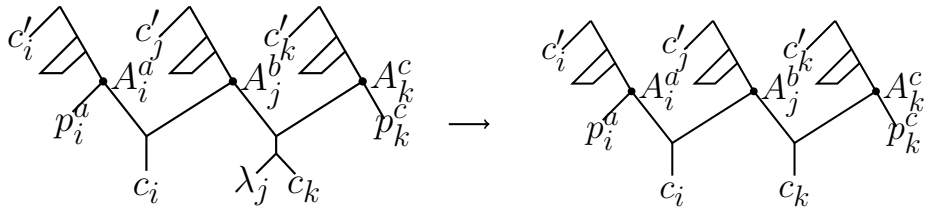

Figure 5.8: The x_h gadget when C_i gets satisfied by x_h and C_k by $\overline{x}_h$.

This situation is illustrated in Figure 5.8. The reader may bear in mind that the situation shown on the left of Figure 5.8 is not entirely general, since we do not know when exactly c_i replaces B_i^a and when c_k replaces B_k^c. That is, c_i might replace B_i^a first, and then more cherry operations might affect c_i before c_k replaces B_k^c, and

hence the exact network on the left of Figure 5.8 might not be reached. Nonetheless, our arguments hold only under the assumption that eventually, c_i replaces B_i^a and eventually, c_k replaces B_k^c.

We argue that after c_i replaces B_i^a, (c_i, p_i^a) cannot be in the cherry sequence C. If this were the case, (c_i, p_i^a) would remove the edge from A_i^a leading to c_i, and the parent of c_j' would be on the path between the (unique) lowest ancestor of c_i and c_i'. At this point, to make c_i and c_i' siblings using cherry picking, we would eventually reach a situation where c_i and c_j' are siblings. We cannot delete either, making it impossible to reach T. Therefore, we can never remove the unique edge incident to A_i^a that leads to c_i. By a similar argument, because we know that c_k replaces B_k^c eventually, we can never remove the unique edge from A_k^c that leads to c_k. It follows that at some point, we must reach the situation in Figure 5.8 on the right. But then, the only two possible moves that involve one of c_i or c_k are (c_i, p_i^a) and (c_k, p_k^c). As we argued, both of these prevent us from reaching T, which is a contradiction.

We conclude that we could not have added two contradictory literals in Q, and thus ϕ is satisfiable. □

Theorem 10. *The problem of deciding whether $d_t(T, N) \leq k$ for given network N, tree T and integer k is NP-hard.*

Proof. We show that there is a polynomial-time reduction from CR-SUBTREE to the problem of computing d_t. Let N, T be an instance of CR-SUBTREE. Our instance of computing d_t is also N, T, and we put $k = |V(N) - V(T)|/2$. We show that $T \subseteq_{cr} N$ if and only if $d_t(N, T) = \frac{|V(N)-V(T)|}{2}$.

($\Rightarrow$) Assume $T \subseteq_{cr} N$. First, note that the tail distance is equivalent to the deconstruction distance by Theorem 3.

This direction is easy to see in light of Theorem 4. The assumption $T \subseteq_{cr} N$ implies that T is an MACRS of N and T. This is because T is the largest possible agreement subnetwork of T and N that can be reached by cherry picking. It follows from Theorem 4 that

$$\begin{aligned} d_t(N, T) &= d_d(N, T) \\ &= \frac{|V(N)| - 1}{2} + \frac{|V(T)| - 1}{2} - |V(T)| + 1 \\ &= \frac{|V(N)| - |V(T)|}{2} \end{aligned}$$

($\Leftarrow$) Assume that $d_t(N, T) = \frac{|V(N)-V(T)|}{2}$. Let S^T be a complete CS for T, and let CS S^N be a complete CS for N. Say S^T and S^N have a maximum common ending subsequence S, so that

$$d_t(N, T) = |S^N| + |S^T| - 2|S|$$

Substituting the given value of k for $d_t(N,T)$ we get

$$\frac{|V(N)| - |V(T)|}{2} = |S^N| + |S^T| - 2|S|$$

We also know from equation 4.2 the length of a complete CS, so we substitute for the value of S^N.

$$\frac{|V(N)|}{2} - \frac{|V(T)|}{2} = \frac{|V(N)|}{2} - \frac{1}{2} + |S^T| - 2|S|$$
$$2|S| = |S^T| + \frac{|V(T)|}{2} - \frac{1}{2}$$

It happens to be that the last two terms on the right hand side express the length of a complete reducing CS for T. Thus we have

$$2|S| = |S^T| + |S^T|$$
$$|S| = |S^T|$$

and since S is a suffix of S^T with the same length as S^T, it follows that S is equal to S^T, and so S is a complete reducing sequence for T. If we write $S^N = S' \cdot S$, $N\langle S'\rangle$ yields a network for which S is a complete CS. By Lemma 1, $N\langle S'\rangle$ is isomorphic to T, implying that $T \subseteq_{cr} N$. □

Corollary 1. *The problem of computing the distances d_t, d_c or d_d on two given networks is NP-hard.*

By slightly adapting the argument of the last result, we can also show that computing the mixed distance is hard.

Theorem 11. *The problem of deciding whether $d_m(T,N) \leq k$ for given network N, tree T and integer k is NP-hard.*

Proof. We reduce the CR-SUBTREE problem to computing d_m. Let N, T be an instance of CR-SUBTREE. Our instance of computing d_m is also N, T, and we put $k = |V(N) - V(T)|/2$. We show that $T \subseteq_{cr} N$ if and only if $d_m(N,T) \leq \frac{|V(N)-V(T)|}{2}$.

($\Rightarrow$) : assume that $T \subseteq_{cr} N$. Then by definition, there exists a CS S that contains only cherry reductions such that $N\langle S\rangle = T$. Notice that each element of S removes exactly two vertices of the network. It follows that $|S| = \frac{|V(N)|-|V(T)|}{2} = k$. Moreover, since the mixed cherry distance allows cherry reduction operations, we obtain $d_m(N,T) \leq k$.

($\Leftarrow$) : suppose that $d_m(N,T) \leq k$. Let S be a mixed CS of at most k operations that can transform N into T. It is not hard to see that S contains only cherry reductions. Indeed, let ρ be the number of cherry reductions in S (which each decrease the number of vertices by 2) and let ε be the number of cherry expansions in S (which

each increase the number of vertices by 2). Observe that $|S| = \rho + \varepsilon \leq k$. In particular, $\rho - \varepsilon \leq k - 2\varepsilon$. Then the number of vertices in $N\langle S\rangle$ is $|V(N)| - 2\rho + 2\varepsilon = |V(T)|$ (the latter because $N\langle S\rangle = T$). Therefore,

$$k = \frac{|V(N)| - |V(T)|}{2} = \rho - \varepsilon \leq k - 2\varepsilon$$

If $\varepsilon > 0$, this yields a contradiction. We deduce that there is no cherry expansion in S. Therefore, S transforms into T using only cherry reductions, and therefore $T \subseteq_{cp} N$. □

5.6 Concluding remarks

5.6.1 Naming conventions

In this work, we define four distances based on cherry operations. The mixed distance is not returned to in later projects in this program. We showed that the tail distance, the construction distance, and the reconstruction distance are equivalent. For convenience and clarity, from now on we refer to these three distances collectively as *cherry distance*.

It is in this work that we establish the connection between MACRS and cherry distance in Theorem 4. We return to this finding later as justification to calculate cherry distance by solving MACRS, indeed the multiplicity of cherry sequences for a given network, and the ability to take advantage of known network structure make MACRS an enticing proxy.

5.6.2 Cherry distance bias

There is a clear "bottom bias" to cherry distances in which disagreements in the networks closer to the leaves are easier to access with cherry operations than those disagreements closer to the root. A specific result of this bias is that the cherry distance won't penalize misplaced leaves near other leaves. However, differences in the networks have a gradually increasing cost with the further the occurrence is from a cherry. Therefore, where the difference between two networks may only be one disagreeing element, our cherry operation-based distances will give different results depending on whether that element is already part of a cherry, or if it is further from a cherry. The mixed distance somewhat mitigates this bias, by having the ability to "switch" leaf labels without the need to clear non-leaf structures below the target leaf, as observed in Figure 5.1. Use of the mixed distance does not completely relieve the bias, for example in cases where a leaf should be removed rather than just change label. We further explore this attribute of cherry distance in Section 7.4.2, where we demonstrate that this "bias" can be considered as reflection of how many leaves are affected by a disagreeing element in a network.

5.6.3 Algorithms

In this work, we have defined several distances, but have not provided any algorithm except in the case of the inputs being two trees. Of course, comparing CSs to calculate cherry distance naively would be exponential in the network size. Previous works have demonstrated a way to define a master CS corresponding to a network [56], from which we may take inspiration in order to reduce the selection pool of CSs for comparison. Not reported in this document due to lack of success, many efforts were made to compare CSs based on heuristics.

In all, working with the MACRS directly yielded more promising results. Parameterization based on the number of reticulations looked quite promising for any future work in engineering an exact algorithm to calculate cherry distance, which is exactly the strategy we take next.

Chapter 6

Project 2: Exact FPT algorithm to find cherry-based distance

6.1 Front matter

6.1.1 Abstract

Phylogenetic networks are increasingly being considered as better suited to represent the complexity of the evolutionary relationships between species. One class of phylogenetic networks that has received a lot of attention recently is the class of orchard networks, which is composed of networks that can be reduced to a single leaf using cherry reductions. Cherry reductions, also called cherry-picking operations, remove either a leaf of a simple cherry (sibling leaves sharing a parent) or a reticulate edge of a reticulate cherry (two leaves whose parents are connected by a reticulate edge). In this paper, we present a fixed-parameter tractable algorithm to solve the problem of finding a maximum agreement cherry-reduced subnetwork (MACRS) between two rooted binary level-1 networks. This is the first exact algorithm proposed to solve the MACRS problem. As proven in earlier work, there is a direct relationship between finding an MACRS and calculating a distance based on cherry operations. As a result, the proposed algorithm also provides a distance that can be used for the comparison of level-1 networks.

6.1.2 Dissemination

This work was first presented at the the 20th satellite conference on comparative genomics (RECOMB-CG), colocated with the international conference Research in Computational Molecular Biology (RECOMB), delivered in person in April 2023 in

İstanbul, Türkiye. The full bibliographic entry for the conference version follows.

[111] K. Landry, O. Tremblay-Savard, and M. Lafond, "Finding agreement cherry-reduced subnetworks in level-1 networks," in *RECOMB International Workshop on Comparative Genomics* (K. Jahn and T. Vinař, eds.), pp. 179–195, Springer Nature Switzerland Cham, Springer Nature Switzerland, 2023

Following this, we were invited to publish an extended version of the work in a special edition of journal Journal of Computational Biology (JCB). This edition includes some additional work that provides a more fine-grained analysis of runtime based on the specific input through the abstraction of the input to a newly defined tree graph, the *dependency tree*, that models the ancestry relationships of reticulations in the network. This is the version presented here, and the full bibliographic entry follows.

[112] K. Landry, O. Tremblay-Savard, and M. Lafond, "A fixed-parameter tractable algorithm for finding agreement cherry-reduced subnetworks in level-1 orchard networks," *Journal of Computational Biology*, vol. 31, no. 4, pp. 360–379, 2024

6.1.3 Contribution to authorship

In this research project my main responsibility was the formalization and realization of the main algorithm with collaborative design meetings. I heavily contributed in proving the correctness of the algorithm. I was also responsible for research and realization of a section on "Counting the number of reticulation-trimmed subnetworks" along with valuable input from my coauthors. Finally, I was responsible for the writing of the first draft, for the majority of concluding discussion, realizing figures, and formatting.

During the writing of this manuscript, the work of Olivier Tremblay-Savard was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) grant number RGPIN-2016-06051. The work of Manuel Lafond was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) grant number RGPIN-2019-05817 and, in part, by Fonds de recherche du Québec—Nature et technologies (FRQNT) grant number 2022-NC-299552.

6.1.4 Manuscript modifications

Some figure captions have been improved for legibility. The language in Lemma 9 has been updated to improve clarity with respect to some intermediary claims of the proof.

6.2 Introduction

6.2.1 Motivation

The previous work shows NP-hardness of the problem of cherry distance on general networks. In fact, the construction of the proof requires a high network level. Yet, in this section, we assume networks are level-1. The motivation behind this decision is to first construct an algorithm design on a more simple case. Though "simple" in the sense that the elements of network complexity is more constrained, the algorithm design was not, with significant enough details to warrant a work of its own, presented here. The intention is for this level-1 algorithm design to serve as a guide and backstop for future iterations.

6.2.2 New definitions

New network labeling and isomorphism

We now refine network labeling. We use X to denote the set of all taxa and the leaves of a network $\mathcal{N}$ are labeled by one *or more* taxa. For $l \in L(\mathcal{N})$, we will use $X(l)$ to denote the set of taxa that label l. We require that $X(l) \neq \emptyset$, and that for any distinct leaves $l_1, l_2 \in L(\mathcal{N})$, $X(l_1) \cap X(l_2) = \emptyset$.

With our new labeling strategy, we also refine the idea of isomorphism. Two networks $\mathcal{N}_1, \mathcal{N}_2$ are *weakly isomorphic* if there exists a bijection $\sigma : V(\mathcal{N}_1) \to V(\mathcal{N}_2)$ such that $(u, v) \in E(\mathcal{N}_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(\mathcal{N}_2)$, and such that for each $l \in L(\mathcal{N}_1)$, $X(l) \cap X(\sigma(l)) \neq \emptyset$. For this we use the notation $\mathcal{N}_1 \simeq \mathcal{N}_2$. If, for each $l \in L(\mathcal{N}_1)$, $X(l) = X(\sigma(l))$, then we say $\mathcal{N}_1$ and $\mathcal{N}_2$ are *strongly isomorphic* which we denote by $\mathcal{N}_1 = \mathcal{N}_2$. Put simply, a strong isomorphism preserves every label, which has been our assumption until now. A weak isomorphism only requires one label be preserved on each leaf.

In the following work, we often refer to vertices, reticulations, and leaves below a vertex v. The descendants of v are denoted $reach(v, \mathcal{N})$ while its ancestors are denoted $reach^{-}(v, \mathcal{N})$ (note that v itself is in both sets). The union of the labels in $reach(v, \mathcal{N}) \cap L(\mathcal{N})$ is denoted $X(v)$. We denote by $R(v)$ the set of reticulations in $reach(v, \mathcal{N})$.

Cherry reductions and sequences

We now consider a minimally reduced network to have two vertices: a root and a leaf, which we refer to as a *single-leaf network*. Along with modifications regarding the labeling described below, we also adjust how the last cherry is reduced. We do so in the expected way, by removing the special case required for the last reduction, of say (x, y), and simply removing the leaf labeled with y and its incident edge. A similar

change can be done for the first cherry expansion. In Section 6.5, we introduce a new definition for a certain tree structure. For legibility, this manuscript has changed (from the original publication) how it refers to the minimum such tree, to match how minimum networks are now referred to, i.e. singletons have one vertex.

If (x, y) is a simple cherry of $\mathcal{N}$, then its reduction is newly defined here. As before, the leaf x is removed, then the edge $(p(x), x)$, suppressing the resulting unary node, if any, and re-assigning $X(y) = X(y) \cup X(x)$. It is in this reassignment of labels that the operation we introduce here differs from the cherry reduction operation described in previous work, where both the leaf x and the set $X(x)$ are deleted. The purpose of our new definition is to preserve a reference to which label *could* have been assigned to y. This is to say that the labels on a given leaf are interchangeable [106, Lemma 3]. For more intuition, recall the if (x, y) is a cherry of $\mathcal{N}$, then so is (y, x). In this way, we were previously making an arbitrary decision of which label to keep, now we decide to keep all labels while working towards an MACRS of the input networks. By forcing all leaf labels to be conserved as the network is reduced, by subsuming labels of a removed leaf onto its cherry sibling that remains, we keep a "memory" of reductions and this compressed representation of networks allows to restore all possible alternative bijective leaf labelings from it, as proved in the following work. This new labeling scheme is a natural adaptation/use of the *subsume* set described in Section 5.3.2.

See Figure 6.1 for an illustration of the two cherry reduction operations, and the concepts of isomorphism. In this figure we see $\mathcal{N}_1$ undergo a simple reduction to become $\mathcal{N}_2$ with a multilabeled leaf, then we see $\mathcal{N}_2$ undergo a reticulated reduction to become $\mathcal{N}_3$. The networks $\mathcal{N}_4$ and $\mathcal{N}_5$ are weakly (but not strongly) isomorphic.

When a CS S reduces a network $\mathcal{N}$ to a single-leaf, then we say S is complete for $\mathcal{N}$. In contrast to previous works, we now tend to work with partial cherry sequences, and thus introduce new notation for ease. The number of pairs in a CS S is denoted $|S|$. The pair at position i of a CS S is referred to by S_i. We use $\mathcal{N}\langle S\rangle$ to denote the network obtained from $\mathcal{N}$ by first applying cherry reduction S_1 on $\mathcal{N}$, then S_2 on the resulting network, and so on until $S_{|S|}$ is applied. Note that we allow S to contain pairs that do not modify the network (e.g. non-cherries). The subsequence from (including) the first cherry to (excluding) the ith cherry in S is $S_{(0:i)}$.

Cherries on a network can be reduced in any order. We restate a theorem of [56, Theorem 1] (also shown in [84]) that we adapt to our formalism[1].

Theorem 12. *Let $\mathcal{N}$ be a network, let (x, y) be a cherry of $\mathcal{N}$, and let S be a CS that contains (x, y). Then there exists a CS S' such that $\mathcal{N}\langle S\rangle = \mathcal{N}\langle S'\rangle$ and $|S| = |S'|$, and whose first element is (x, y).*

[1]Note that the authors prove the statement under the assumption that S is complete, and that leaves are single-labeled. However the proof is easy to adapt to our context.

Biconnected components

Biconnected components become critical in the algorithm design presented here, so they are formalized with notation here.

While a network $\mathcal{N}$ is directed, there is an undirected version of $\mathcal{N}$ on the same vertex set and with an undirected edge $\{u, v\}$ present for every $(u, v) \in E(\mathcal{N})$ which we call the *underlying graph*. It is on this underlying graph that we identify the set of *biconnected components* of $\mathcal{N}$. Such a component is a maximal subgraph B that cannot be disconnected by the removal of an edge (if it exists) $(u, v)|\{u, v\} \in B$. Note that every individual leaf and some tree vertices alone constitute a biconnected component, so we refer to such single vertex components ($|B| = 1$) as *trivial*, and all others as *non-trivial*.

For a set of biconnected components $B_1, \ldots, B_b$ on a network $\mathcal{N}$, a *bridge* is an edge (u, v) such that $u \in B_i$, $v \in B_j$ for any arbitrary $1 \leq i \neq j \leq b$. Intuitively, a bridge disconnects the underlying graph when removed. In a tree, every biconnected component is trivial (each vertex) and every edge is a bridge. In a network, biconnected components that are non-trivial contain a reticulation, they always have more than two vertices since parallel edges are not allowed.

We say $\rho(\mathcal{N})$ *roots* $\mathcal{N}$. If, for a vertex v, and for all vertices $v' \in reach(v, \mathcal{N})$, if every path from $\rho(\mathcal{N})$ to v' goes through v, then we say v roots the subnetwork composed of all the vertices in $reach(v, \mathcal{N})$. This idea is essential in describing biconnected components, which are rooted by a unique vertex.

6.2.3 MACRS problem statement

We restate MACRS as a formal problem.

The *Maximum Agreement Cherry-Reduced Subnetwork* (MACRS) problem.
Input: Two orchard networks $\mathcal{N}_1$ and $\mathcal{N}_2$.
Find: A network $\mathcal{N}^*$ with the maximum number of vertices that satisfies $\mathcal{N}^* \subseteq_{cr} \mathcal{N}_1$ and $\mathcal{N}^* \subseteq_{cr} \mathcal{N}_2$.

We can require that the input have either $X(\mathcal{N}_1) = X(\mathcal{N}_2)$ or that $X(\mathcal{N}_1) \cap X(\mathcal{N}_2) \neq \emptyset$. We call a solution $\mathcal{N}^*$ to the above problem an MACRS of $\mathcal{N}_1$ and $\mathcal{N}_2$.This implies the existence of CSs S_1 and S_2 such that $\mathcal{N}_1\langle S_1\rangle \simeq \mathcal{N}_2\langle S_2\rangle$, note how we only require weak isomorphism for this problem.

6.2.4 Dynamic programming and level of the network

In the previous work we give a dynamic programming algorithm that can calculate the cherry distance between trees in polynomial time. We build off this design by considering how reticulations should be handled. The main question we ask is, under the assumption we are searching for an MACRS, how are the reticulations affected by

cherry reductions that are leading to such an MACRS? For any reticulation vertex, there are up to three options; it will remain intact (because it forms part of the MACRS between the input networks), or it will be reduced by a cherry on one of its two reticulation edges. So for each reticulation in each input network, we fix one of these decisions, enumerating all such choices for the combined set of reticulations. We reduced the networks in a minimum way according to such choices. Next, for each pair of these reduced networks, we then calculate a form of the MACRS problem that considers only simple reductions. It is strictly simple reductions present on trees, so here is where we utilize a version of our original dynamic programming algorithm.

To facilitate the calculation, we rely on the specific structure that an "untangled" reticulation appears in, so specifying that our inputs are level-1 so that we can handle whole biconnected components as individual entries in our dynamic programming algorithm, i.e. since the nontrivial biconnected components have a very predictable form, we basically hard code how to operate on them. Recall from Section 2.3.2 that every level-1 network is an orchard.

6.2.5 Outline

We present an exact fixed-parameter tractable (FPT) algorithm to compute an MACRS of two rooted binary level-1 orchard networks that is exponential in the sum of reticulations present in both networks (Section 6.3). More precisely, our algorithm runs in $O(3^r n^3)$, where r is the sum of reticulations on the two input networks and n represents the maximum number of vertices of the input networks. Our approach essentially consists of enumerating a certain set of subnetworks of the input networks in which all possible combinations of reticulation edges have been removed. Then, it makes use of a dynamic programming algorithm that finds whether there is an MACRS (and what it is, if it exists) or not between two level-1 networks in which reticulations that are remaining cannot be removed (we call this problem MACRS-Simple). We prove that the initial MACRS problem can be solved by solving the MACRS-Simple problem on all combinations of enumerated subnetworks. We go on to describe how to solve both the enumeration and the MACRS subproblems in Section 6.4. Although this enumeration is responsible for the 3^r factor in our algorithm complexity, this is a worst-case bound that is only achieved by certain networks, whereas others only require enumerating a linear number of subnetworks. In the latter case, our algorithm can take polynomial time even on some networks with a large number of reticulations. In Section 6.5, to formalize this, we describe a tree structure, called the *dependency tree*, that represents the hierarchical relationships between the reticulations in a level-1 network. We then use this tree to count how many subnetworks can be enumerated from a particular network, and present a few examples to demonstrate how the network structure can affect the number of enumerated subnetworks. This counting work on the dependency tree model will go on to inspire work in Chapter 7.

6.3 An MACRS algorithm on level-1 networks

We show that the MACRS problem can be solved in time $O(3^r n^3)$ for

$$n = \max(|V(\mathcal{N}_1)|, |V(\mathcal{N}_2)|) \text{ and } r = |R(\mathcal{N}_1)| + |R(\mathcal{N}_2)|$$

on level-1 networks. We employ a two-step strategy. We first enumerate a number of inputs that have been specially reduced to a selected set of remaining reticulations. Second, these inputs are provided to a cubic time dynamic programming algorithm on an easier version of MACRS that uses only simple reductions. Because the number of special inputs is limited by 3^r, we get an FPT algorithm. MACRS is thus split into two subproblems. We first introduce them and show how they can be used to solve MACRS. The later sections then focus on each problem separately.

Let $\mathcal{N}$ be a network and let $F \subseteq E_R(\mathcal{N})$ be a subset of reticulation edges. We wish to generate all the maximal cherry-reduced subnetworks of $\mathcal{N}$ under the restriction that the reticulation edges removed by cherry operations coincide with F. Thus, we say that a network $\mathcal{N}'$ is a *reticulation-trimmed subnetwork* of $\mathcal{N}$ *with respect to* F if there exists a CS S such that $\mathcal{N}\langle S\rangle = \mathcal{N}'$, and such that $(u, v) \in F$ if and only if S contains a reticulated cherry reduction that removes (u, v), and S is of minimum length, i.e. we require that there is no other CS S' with $|S'| < |S|$ that satisfies the same properties.

Furthermore, we say that $\mathcal{N}'$ is a *reticulation-trimmed subnetwork* of $\mathcal{N}$ if there exists a set $F \subseteq E_R(\mathcal{N})$ such that $\mathcal{N}'$ is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F.

The Reticulation-Trimmed Enumeration problem:
Input: An orchard network $\mathcal{N}$.
Find: The set of all reticulation-trimmed subnetworks of $\mathcal{N}$.

Note that the size of the set of reticulation-trimmed subnetworks depends heavily on the network structure. For instance, it is possible to show that it is linear when all reticulations are arranged in a path, and exponential when all reticulations are independent (none is an ancestor of the other). It is possible to bound the size of this set by algorithmic means through an abstraction of the network structure. We analyze this parameter in a later section.

Once the set of edges to remove by reticulation reductions have been guessed, it remains to infer the set of non-reticulated cherry operations. A *simple CS* is a CS that contains only simple cherries. In this way, $R(\mathcal{N}) = R(\mathcal{N}\langle S\rangle)$ for any simple CS S. For networks $\mathcal{N}$ and $\mathcal{N}'$, when there exists a simple CS S such that $\mathcal{N}\langle S\rangle \simeq \mathcal{N}'$, we say that $\mathcal{N}'$ is a CRS-SIMPLE of $\mathcal{N}$. Note that owing to our definition of weak isomorphism, $\mathcal{N}\langle S\rangle \simeq \mathcal{N}'$ does not mean that S transforms $\mathcal{N}$ into $\mathcal{N}'$. A better intuition would rather be that after applying S on $\mathcal{N}$, we could choose one label in the label set of each leaf of $N\langle S\rangle$ and of $\mathcal{N}'$, such that the resulting networks would

be isomorphic in the traditional sense.

The *Simple Maximum Agreement Cherry-Reduced Subnetwork* (MACRS-Simple) problem.
Input: Two orchard networks $\mathcal{N}_1$ and $\mathcal{N}_2$.
Find: A network $\mathcal{N}^*$ with a maximum number of vertices such that $\mathcal{N}^*$ is a CRS-SIMPLE of $\mathcal{N}_1$ and a CRS-SIMPLE of $\mathcal{N}_2$.

A solution $\mathcal{N}^*$ to the above problem will be called an MACRS-Simple of $\mathcal{N}_1$ and $\mathcal{N}_2$.

For the standard MACRS problem on orchard networks $\mathcal{N}_1$ and $\mathcal{N}_2$, there is always a solution as long as $X(\mathcal{N}_1) \cap X(\mathcal{N}_2) \neq \emptyset$, however since reticulations cannot be removed by simple CS, the MACRS-Simple problem may not have a solution (for instance when the two networks have a different number of reticulation vertices). We can now describe our main algorithm, where we assume that the MACRS-Simple routine correctly returns an optimal solution.

Algorithm 1 MACRS Finder

Input Two networks $\mathcal{N}_1$ and $\mathcal{N}_2$
Output An MACRS of $\mathcal{N}_1$ and $\mathcal{N}_2$

$\tilde{\mathcal{N}} \leftarrow$ empty network
for each reticulation-trimmed subnetwork $\mathcal{N}_1'$ of $\mathcal{N}_1$ **do**
    for each reticulation-trimmed subnetwork $\mathcal{N}_2'$ of $\mathcal{N}_2$ **do**
        Let $\mathcal{N}'$ be a MACRS-Simple of $\mathcal{N}_1'$ and $\mathcal{N}_2'$
        if $\mathcal{N}'$ exists and $|V(\mathcal{N}')| > |V(\tilde{\mathcal{N}})|$ **then** $\tilde{\mathcal{N}} \leftarrow \mathcal{N}'$
    end for
end for
return $\tilde{\mathcal{N}}$

An optimization technique is evident here: as we mentioned, there is only a solution to MACRS-Simple ($\mathcal{N}_1$, $\mathcal{N}_2$) when $|R(\mathcal{N}_1)| = |R(\mathcal{N}_2)|$ since only simple reductions will be performed. Thus, we need only test such pairs. We do not expect this optimization to provide any theoretical improvement in general and it is not included in the analysis of bounds presented here. It may, however, provide an improvement in practice, a test which we defer to future work. We can also note that there is only a solution to MACRS-Simple ($\mathcal{N}_1$, $\mathcal{N}_2$) when $R(\mathcal{N}_1)$ and $R(\mathcal{N}_2)$ share the same topological structure. Again, we do not show any theoretical improvement of the bounds but we do show in a later section how to analyze improvements in specific cases with this fact in mind.

In the remainder of this section, we focus on proving that this algorithm works correctly. We will deal with the complexity of the algorithm once we have dealt with the Reticulation-Trimmed Enumeration and MACRS-Simple subproblems.

We begin by showing that one can always obtain a subnetwork by first going through a reticulation-trimmed subnetwork, and then using only simple cherry reductions.

Lemma 6. *Let $\mathcal{N}$ be a network. Then for any $\mathcal{N}' \subseteq_{cr} \mathcal{N}$, there exists a reticulation-trimmed subnetwork $\mathcal{N}''$ of $\mathcal{N}$ and a simple CS S such that $\mathcal{N}''\langle S\rangle = \mathcal{N}'$.*

Proof. Let $\mathcal{N}$ and $\mathcal{N}'$ be networks such that $\mathcal{N}' \subseteq_{cr} \mathcal{N}$. We use induction on $|E(\mathcal{N})|$ to prove a slightly stronger statement. We show that for any $F \subseteq E_R(\mathcal{N})$ such that there exists a CS S' satisfying $\mathcal{N}\langle S'\rangle = \mathcal{N}'$ that removes the set of reticulation edges F, there exists a reticulation-trimmed subnetwork $\mathcal{N}''$ of $\mathcal{N}$ with respect to F and a simple CS S such that $\mathcal{N}''\langle S\rangle = \mathcal{N}'$. In other words, if we can go from $\mathcal{N}$ to $\mathcal{N}'$ by removing F, then we can first remove F in a minimum way (as required by reticulation-trimmed subnetworks), and then proceed with a simple CS.

The base case is $|E(\mathcal{N})| = 1$. In this case we have a single-leaf network on a root, a single leaf, and an edge between them. There are no cherries and only $F = \emptyset$ is possible, and so all $\mathcal{N}' \subseteq_{cr} \mathcal{N}$ have $\mathcal{N}' = \mathcal{N}$ and $S = \emptyset$ for $\mathcal{N}\langle S\rangle = \mathcal{N}'$. So the claim is trivially true.

For the induction step, assume that the claim holds for all networks whose number of edges is strictly smaller than $|E(\mathcal{N})|$.

Let $F \subseteq E_R(\mathcal{N})$, and suppose there is a CS S' such that $\mathcal{N}\langle S'\rangle = \mathcal{N}'$, and that the set of reticulation edges removed by S' is F. If every reduction in S' is simple, then $F = \emptyset$ and $\mathcal{N}$ is itself a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to $F = \emptyset$, in which case the statement holds. Otherwise, let $(u, v) \in F$ be the first reticulation edge of $\mathcal{N}$ removed by a reduction in S', say (u, v) is removed by cherry S'_i. On network $\mathcal{N}\langle S'_{(0:i)}\rangle$, u and v must both have leaf children. This implies that there are no reticulations in $reach(u, \mathcal{N})$. Thus, all cherries (x, y) of $S'_{(0:i)}$ such that $\{x, y\} \in reach(u, \mathcal{N})$ are simple.

Assume for now that at least one simple cherry reduction in $S'_{(0:i)}$ exists, and let $S'_h = (x, y)$ be the first of them. With this, we see that (x, y) is a cherry on $\mathcal{N}$, and remains so throughout every step of the reduction of $\mathcal{N}$ by $S'_{(0:h)}$, including on $\mathcal{N}$. We may therefore assume that (x, y) is the first reduction of S' by Theorem 12.

Thus $\mathcal{N}'$ is obtained from $\mathcal{N}\langle(x, y)\rangle$ by applying the CS $S'_{(1:|S|)}$, which we know removes the set of reticulation edges F. By induction, there exists a reticulation-trimmed subnetwork $\mathcal{N}^*$ of $\mathcal{N}\langle(x, y)\rangle$ with respect to F and a simple CS S^* such that $\mathcal{N}^*\langle S^*\rangle = \mathcal{N}'$. Let T be a smallest (minimal) CS that results in $\mathcal{N}^*$ after applying it on $\mathcal{N}\langle(x, y)\rangle$. Then $\mathcal{N}^*$ can be obtained from $\mathcal{N}$ by applying the CS $(x, y) \cdot T$. We note that the reduction (x, y) is required in any CS that results in a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F (since the cherry (x, y) is below u and (u, v) needs to be removed) and it follows by the minimality of T on $\mathcal{N}\langle(x, y)\rangle$ that $(x, y) \cdot T$ is minimum on $\mathcal{N}$.

Thus $\mathcal{N}^*$ is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F and the claim holds.

It is also possible that (x, y) does not exist as a simple reduction, which occurs when u and v each already have a leaf child in $\mathcal{N}$. In this case, (x, y) is a reticulated cherry on $\mathcal{N}$ such that $p(x) = v$ and $p(y) = u$. As in the previous case, we may assume that (x, y) is the first reduction of S'. Let $F' = F \setminus \{(u, v)\}$. By induction there is a reticulation-trimmed subnetwork $\mathcal{N}^*$ of $\mathcal{N}\langle(x, y)\rangle$ with respect to F' and there exists S^* such that $\mathcal{N}^*\langle S^*\rangle = \mathcal{N}'$. We also have a CS T that is minimum and contains a reticulated reduction if and only if it removes an edge in F'. As before, the reduction (x, y) is required in any reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, and it follows by the minimality of T that $(x, y) \cdot T$ yields such a network. Again, $\mathcal{N}^*$ is a reticulation-reduced subnetwork of $\mathcal{N}$ with respect to F and there is the simple CS S^* such that $\mathcal{N}^*\langle S^*\rangle = \mathcal{N}'$. □

Theorem 13. *Algorithm 1 correctly finds a* MACRS *of* $\mathcal{N}_1$ *and* $\mathcal{N}_2$.

Proof. Let $\mathcal{N}^*$ be a MACRS of $\mathcal{N}_1$ and $\mathcal{N}_2$. Let $\tilde{\mathcal{N}}$ be the network returned by Algorithm 1. We first claim that if $\tilde{\mathcal{N}}$ is non-empty and that it satisfies $\tilde{\mathcal{N}} \subseteq_{cr} \mathcal{N}_1, \mathcal{N}_2$. To see this, note that every pair $\mathcal{N}_1', \mathcal{N}_2'$ of networks enumerated by Algorithm 1 satisfy $\mathcal{N}_1' \subseteq_{cr} \mathcal{N}_1$ and $\mathcal{N}_2' \subseteq_{cr} \mathcal{N}_2$, by the definition of reticulation-trimmed subnetworks. Moreover, if an MACRS-Simple $\mathcal{N}'$ of $\mathcal{N}_1', \mathcal{N}_2'$ exists, then by transitivity, $\mathcal{N}' \subseteq_{cr} \mathcal{N}_1' \subseteq_{cr} \mathcal{N}_1$ and $\mathcal{N}' \subseteq_{cr} \mathcal{N}_2' \subseteq_{cr} \mathcal{N}_2$. Since $\tilde{\mathcal{N}}$ is one of those $\mathcal{N}'$, this proves our claim.

Let us now focus on the optimality of $\tilde{\mathcal{N}}$. First note that $|V(\mathcal{N}^*)| \geq |V(\tilde{\mathcal{N}})|$ if $\tilde{\mathcal{N}}$ is an empty network, which is obvious, and otherwise, by our above claim, $\tilde{\mathcal{N}}$ is a cherry-reduced subnetwork of $\mathcal{N}_1$ and $\mathcal{N}_2$ and thus cannot be larger than $\mathcal{N}^*$.

Let us now show that $|V(\mathcal{N}^*)| \leq |V(\tilde{\mathcal{N}})|$. By Lemma 6, there exists a reticulation-trimmed subnetwork $\mathcal{N}_1'$ of $\mathcal{N}_1$ (resp. $\mathcal{N}_2'$ of $\mathcal{N}_2$) such that $\mathcal{N}^*$ can be obtained from $\mathcal{N}_1'$ (resp. $\mathcal{N}_2'$) using only simple CSs. Thus, $\mathcal{N}^*$ is a CRS-SIMPLE of $\mathcal{N}_1'$ and $\mathcal{N}_2'$. Algorithm 1 will eventually enumerate $\mathcal{N}_1'$ and $\mathcal{N}_2'$ and find a MACRS-Simple $\mathcal{N}'$ of them, which is of maximum size and thus has at least as many vertices as $\mathcal{N}^*$. Since the returned $\tilde{\mathcal{N}}$ is the $\mathcal{N}'$ of maximum size found by the algorithm, it follows that $|V(\mathcal{N}^*)| \leq |V(\tilde{\mathcal{N}})|$. □

6.4 Solving the MACRS subproblems

We now show how each subproblem required in our MACRS Finder algorithm can be solved.

6.4.1 Enumerating the set of reticulation-trimmed subnetworks

We now show how to enumerate the set of all reticulation-trimmed subnetworks of a network $\mathcal{N}$ in time $O(3^{|R(\mathcal{N})|}|V(\mathcal{N})|)$. The reticulation-trimmed subnetworks are

characterized by having no more reductions than what sufficiently removes the desired reticulation edges. Luckily, we will see that at most one such network can exist; we must only remove the complete subnetwork under both endpoints of each of the reduced reticulation edges. This is guaranteed possible by cherry reductions, assuming all reticulations below these endpoints have also been specified for removal. Algorithm 2 shows how to enumerate the relevant edges, and uses Algorithm 3 as a subroutine, which finds the reticulation-trimmed subnetwork with respect to a given edge set. We show that the reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to $F \subseteq R(\mathcal{N})$ is uniquely defined in Lemma 9. We say that a set of edges F is *disjoint* if, for any two distinct edges $(u, v), (x, y) \in F$, $\{u, v\} \cap \{x, y\} = \emptyset$.

Algorithm 2 REDUCED-SET-FINDER

Input A network $\mathcal{N}$
Output The set of all reticulation-trimmed subnetworks of $\mathcal{N}$

1: **for each** $F \in \{\mathcal{P}(E_R(\mathcal{N})) : (a, b), (c, b) \notin F \text{ for any } a, b, c\}$ **do**
2: $\quad \mathbf{N} \leftarrow \mathbf{N} \cup \text{RT-SUBNET-MAKER}(\mathcal{N}, F)$
3: **end for**
4: return $\mathbf{N}$

Lemma 7. *Let $\mathcal{N}$ be a network, $F \subseteq E_R(\mathcal{N})$ be a set, and $\mathcal{N}'$ be a network that is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F. Then F is disjoint.*

Proof. Say F is not disjoint, then there are two cases. First, say F contains edges (u, v) and (w, v). The reduction on the network must first reduce one of these edges, say (u, v). This reduction removes the vertex v, and thus the edge (w, v) can no longer be in the network. In this way no single CS can reduce both the reticulation edges leading into the same reticulation. Next, assume F is not disjoint because it contains edges (u, v) and (u, w). This is not possible in a level-1 network because reticulations v, w would be contained in the same biconnected component. Biconnected components in a level-1 network have at most one reticulation by definition. □

Next, for $F \subseteq E_R(\mathcal{N})$, a topological sort of F is an ordering of its elements such that for distinct edges $e_1, e_2 \in F$, if there is a path from a vertex of e_1 to a vertex of e_2 in $\mathcal{N}$, then e_1 comes later than e_2 in this ordering.

Lemma 8. *Let $\mathcal{N}$ be a network and $F \subseteq E_R(\mathcal{N})$ be a set such that there exists a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F. Then there exists a topological sort of F.*

Proof. This claim is evidenced by the topological partial ordering on $R(\mathcal{N})$, which exists on any network. Then, note that F is disjoint by Lemma 7. □

The next lemma is crucial, as it shows that reticulation-trimmed subnetworks with respect to a given F are either unique, or do not exist. This allows us to enumerate in reasonable time.

Lemma 9. *Let $\mathcal{N}$ be a network and let $F \subseteq E_R(\mathcal{N})$. Then there does not exist two non-strongly isomorphic reticulation-trimmed subnetworks of $\mathcal{N}$ with respect to F.*

Proof. Let $\mathcal{N}$ be a network and $F \subseteq E_R(\mathcal{N})$ be a set.

We claim there are not two non-strongly isomorphic reticulation-trimmed subnetworks with respect to F. We proceed by induction on $|V(\mathcal{N})| + |E(\mathcal{N})|$.

In the base case, $|V(\mathcal{N})|+|E(\mathcal{N})| \leq 3$ and $\mathcal{N}$ is a single-leaf network on one edge, with a root and a leaf as endpoints. In this case, $F = \emptyset$ and the reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F is $\mathcal{N}$ and $\mathcal{N} = \mathcal{N}$. There are no cherries on $\mathcal{N}$ so there is not more than one reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F.

Assume the claim holds for all networks with strictly fewer vertices and edges than $\mathcal{N}$. If there is no reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, then the claim holds. Assume such a network, $\mathcal{N}'$, exists. If $F = \emptyset$ then $\mathcal{N}$ is the only reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, as making any cherry reductions would not be minimum (since $\mathcal{N}\langle\emptyset\rangle = \mathcal{N}$ is already sufficient to remove empty F). The claim holds in this case, so we now assume $F \neq \emptyset$.

Let edge $e \in F$ be (arbitrarily) a lowest edge in the topological sort on F (which exists, by Lemma 8). Because $\mathcal{N}'$ exists, there is at least one cherry below an endpoint of e (e may be in the cherry itself). Let one such cherry be called (x, y).

Next assume (x, y) is such that $(p(y), p(x)) \neq e$. We know that (x, y) is not reticulated, as otherwise e would not be a lowest reticulation in F. Therefore (x, y) is not reticulated, and thus simple.

Because e is in F, it will have to be removed, and must have leaves under its endpoints to do so, thus (x, y) needs to be reduced in any CS S such that $\mathcal{N}\langle S\rangle$ is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F. Moreover, by Theorem 12, we may assume that any such CS S can start with (x, y) as (x, y) can be removed first without affecting the resulting network.

It follows that we may assume that, for every CS S such that $\mathcal{N}\langle S\rangle$ is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, S can be rearranged so that the first reduction applied can result in $\mathcal{N}\langle(x, y)\rangle$ and give the same result. Assume this rearrangement is done.

Then applying $S_{(1:|S|)}$ on $\mathcal{N}\langle(x, y)\rangle$ must yield a reticulation-trimmed subnetwork of $\mathcal{N}\langle(x, y)\rangle$ with respect to F (in particular, $|S_{(1:|S|)}|$ is minimum as otherwise, $|S|$ would not be minimum). By the induction hypothesis, $\mathcal{N}\langle(x, y)\rangle$ does not have two non-strongly isomorphic reticulation-trimmed subnetworks with respect to F. Since we say that all CSs S applicable to $\mathcal{N}$ that result in such a trimmed subnetwork go through $\mathcal{N}\langle(x, y)\rangle$ after the assumed rearrangement, it follows that $\mathcal{N}$ also does not have two non-strongly isomorphic reticulation-trimmed subnetworks with respect to F.

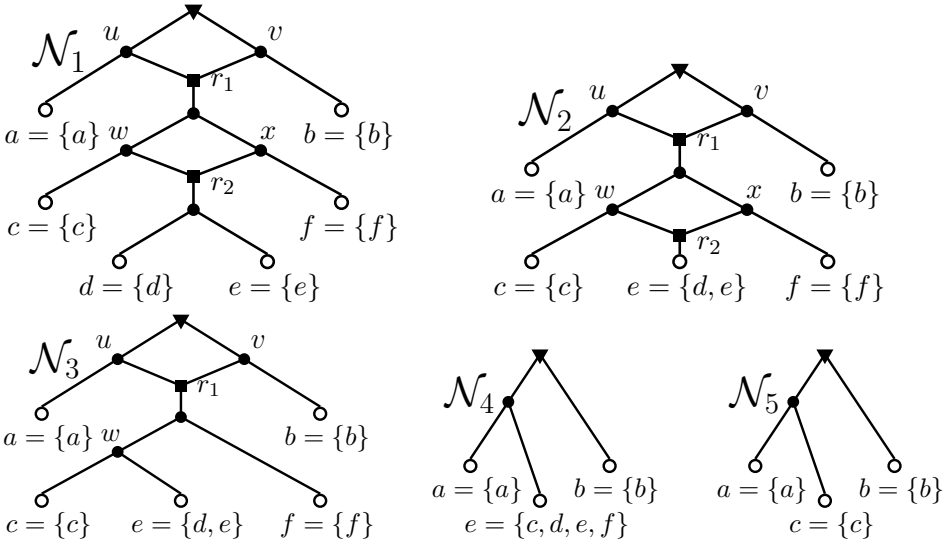

Figure 6.1: In this figure, leaves are represented by open circles, tree vertices as filled circles, reticulations as filled squares, and the root of the network as a filled, inverted triangle. Network $\mathcal{N}_1$ is a level-1 network with $|R(\mathcal{N})| = 2$. $\mathcal{N}_1$ is a reticulation-trimmed subnetwork of $\mathcal{N}_1$ with respect to $F = \emptyset$. Network $\mathcal{N}_2 = \mathcal{N}_1\langle(d, e)\rangle$, where (d, e) is a simple cherry reduction. Network $\mathcal{N}_3 = \mathcal{N}_2\langle(e, f)\rangle$ where (e, f) is a reticulated cherry reduction. $\mathcal{N}_3$ is reticulation-trimmed subnetwork of $\mathcal{N}_1$ and of $\mathcal{N}_2$ with respect to $F = \{(x, r_2)\}$. Network $\mathcal{N}_4 = \mathcal{N}_3\langle(c, e)(f, e)(e, b)\rangle$ and is a reticulation-trimmed subnetwork of $\mathcal{N}_1$ and of $\mathcal{N}_2$ with respect to $F = \{(x, r_2), (v, r_1)\}$ or to $F = \{(w, r_2), (v, r_1)\}$. Network $\mathcal{N}_5 \simeq \mathcal{N}_4$, in fact, there are CSs that may lead to leaf e being any of leaves c, d, e, or f. Each of these networks would have the same label set on that leaf, and all are weakly isomorphic with $\mathcal{N}_5$. No reticulation-trimmed subnetwork of $\mathcal{N}_1$ or of $\mathcal{N}_2$ is admitted with respect to $F = \{(u, r_1)\}$ or to $F = \{(v, r_1)\}$.

Finally, assume that (x, y) is reticulated and $(p(y), p(x)) = e$.

As in the previous case, note that (x, y) must be present in any CS S such that $\mathcal{N}\langle S\rangle$ is a reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, and we may further assume that, for any such CS, there is one with the same effect that can be constructed starting with (x, y). Then, any such CS first goes through $\mathcal{N}\langle(x, y)\rangle$, and then by minimality, results in a reticulation-trimmed subnetwork of $\mathcal{N}\langle(x, y)\rangle$ with respect to $F \setminus \{(p(y), p(x))\}$. By induction, there is only one such network, and thus also only one reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F. □

For an example, given a network $\mathcal{N}$, of an $F \subseteq E_R(\mathcal{N})$ that does not admit a reticulation-trimmed subnetwork of $\mathcal{N}$, consider $\mathcal{N}$ with 2 reticulations, r_1, r_2 such that $r_1 \in reach^-(r_2)$. Choosing $F = \{(p_1, r_1)\}$, for p_1 chosen arbitrarily between r_1's parents, will not admit a reticulation-trimmed subnetwork since reticulation r_1 must have leaves below its endpoints to be in a cherry, but this choice of F has no corresponding reticulated reductions of r_2 making it impossible to construct a CS S

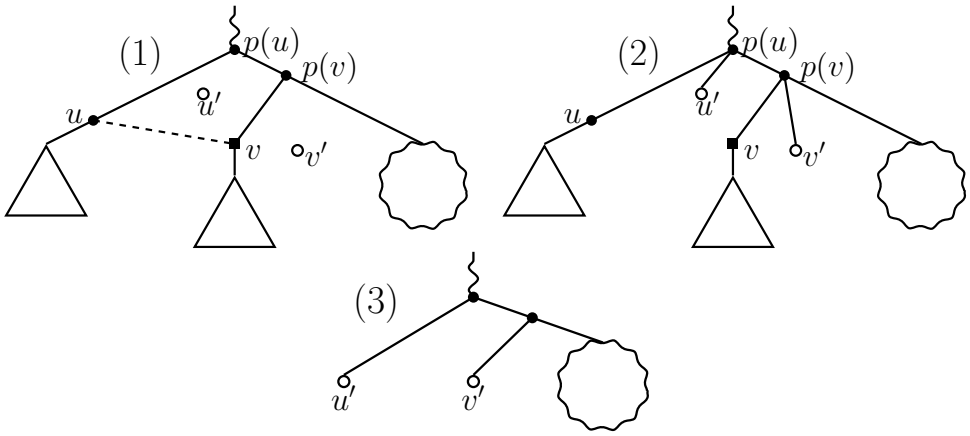

Figure 6.2: In this figure, leaves are represented by open circles, tree vertices as filled circles, reticulations as filled squares. A subnetwork without reticulations is represented by a large open triangle, a subnetwork that may be reticulated is represented by a large open blob. This figure shows an example of the operation of Algorithm 3; note how $R(u) \cup R(v) \setminus \{v\} = \emptyset$ in this example. Subnetwork under label (1) is an example network at line 7, the dotted line represents the removed reticulation edge (u, v) by line 5 and both leaves u' and v' have been constructed (leaf labels are not shown). The network under label (2) shows the state of network (1) at line 8 when edges $(p(u), u')$ and $(p(v), v')$ have been added. The network under label (3) shows the state of the network under (1) at line 9 when vertices in $reach(u, \mathcal{N}') \cup reach(v, \mathcal{N}')$ are removed.

that reduces only r_1. See Figure 6.1 for additional examples.

Algorithm 3 gives our procedure for producing the reticulation-trimmed network with respect to some given F, see Figure 6.2 for an illustration.

Lemma 10. *Algorithm 3 on $(\mathcal{N}, F)$ returns the reticulation-trimmed subnetwork $\mathcal{N}'$ of $\mathcal{N}$ with respect to F if it exists, and NULL if not, and runs in time $O(|V(\mathcal{N})|)$.*

Proof. First, there is always a topological sort on F (Lemma 8) and F is disjoint (Lemma 7).

If a network is not returned, then it must be that $R(u) \cup R(v) \setminus \{v\} \neq \emptyset$ on one of the (possibly) partially reduced subnetworks for some edge $(u, v) \in F$. In this case, F does not admit a reticulation-trimmed subnetwork since F does not contain edges that correspond to the reduction of all reticulations below u, and v, a requirement for the reduction of reticulation v.

Assume a network is returned, we claim the algorithm is correct in this case.

We will show this claim by constructing a CS S such that $\mathcal{N}\langle S \rangle = \mathcal{N}'$, and by counting the exact number of reticulation cherries which we show will remove the desired F. Finally, we will show S is minimum.

Say F' has the order $\langle (u_1, v_1)(u_2, v_2) \ldots \rangle$. For each (u_i, v_i), in order, we construct a CS S_i on the partially reduced $\mathcal{N}$, then let $S = S_1 \cdot S_2 \ldots$. Note that this means we

Algorithm 3 RT-SUBNET-MAKER

Input A network $\mathcal{N}$ and a disjoint set $F \subseteq E_R(\mathcal{N})$

Output The reticulation-trimmed subnetwork of $\mathcal{N}$ with respect to F, or NULL if it does not exist

$\mathcal{N}' \leftarrow \mathcal{N}$
Find a topological sort F' of F
for each $(u, v) \in F'$ in order **do**
    if $R(u) \cup R(v) \setminus \{v\} = \emptyset$ **then**
        delete edge (u, v)
        construct leaf u' such that $X(u') = X(u)$
        construct leaf v' such that $X(v') = X(v)$
        add edges $(p(u), u')$ and $(p(v), v')$ to $\mathcal{N}'$
        remove all vertices in $reach(u, \mathcal{N}') \cup reach(v, \mathcal{N}')$
    else
        return NULL
    end if
end for
return $\mathcal{N}'$

construct each S_i on $\mathcal{N}\langle S_1 \cdot S_2 \cdot \ldots \cdot S_{i-1}\rangle$. There are distinct subnetworks rooted on u_i and v_i so there are CSs S_i^u, S_i^v that are complete for each respective subnetwork. Note how, since we assume a network was returned, $R(u_i) \cup R(v_i) \setminus \{v_i\} = \emptyset$, so S_i^u and S_i^v are simple. Say they reduce the subnetworks to leaves l_i^u and l_i^v respectively (so that $p(l_i^u) = u$ and $p(l_i^v) = v$). Let $S_i = S_i^u \cdot S_i^v \cdot (l_i^v, l_i^u)$. The cherry (l_i^v, l_i^u) will reduce the reticulation edge (u, v) under these conditions.

In this way, there is a reticulation cherry in S if and only if it reduces an edge in F. Furthermore, S is minimum since we construct S_i on the network $\mathcal{N}\langle S_1 \cdot S_2 \cdot \ldots \cdot S_{i-1}\rangle$ and we have selected only the cherry reductions that are necessary and sufficient for the reduction of the targeted reticulation edges.

The claimed running time is straightforward. We can obtain a topological sort on F using a standard topological sort of $\mathcal{N}$ obtained in time $O(|V(\mathcal{N})| + |E(\mathcal{N})|) = O(|V(\mathcal{N})|)$ (since our networks are binary). Then the algorithm only iterates over F and replaces subnetworks by multi-labeled leaves, which can be handled in time $O(|V(\mathcal{N})|)$. □

Theorem 14. *Algorithm 2 correctly enumerates all reticulation-trimmed subnetworks of a network $\mathcal{N}$, and runs in time $O(3^{|R(\mathcal{N})|}|V(\mathcal{N})|)$.*

Proof. It is already proved (Lemma 7) that a non-disjoint F does not admit a reticulation-trimmed subnetwork, so it is correct to filter those. The remaining correctness follows from the exhaustive nature of the construction of all F and by the correctness of Algorithm 3.

As for the time complexity, filtering non-disjoint F implies a threefold choice on each reticulation (we either include one, or none of its incoming edges, but not both by disjointness). Thus the size of the set is $O(3^{|R(\mathcal{N})|})$. Recalling that Algorithm 3 can be implemented in time $O(|V(\mathcal{N})|)$, the total runtime for Algorithm 2 is in $O(3^{|R(\mathcal{N})|}|V(\mathcal{N})|)$.

□

6.4.2 An algorithm for MACRS-Simple

A dynamic programming algorithm that solves the MACRS-Simple problem in cubic time is given and is proven correct in this section.

Assume we have networks $\mathcal{N}_1$ and $\mathcal{N}_2$ as input to the MACRS-Simple problem. We assume that we have computed the set of biconnected components of $\mathcal{N}_1$ and $\mathcal{N}_2$ in a preprocessing step, along with the bridge edges. This can be done in time $O(|V(\mathcal{N})|)$, see [113]. Since the networks considered are level-1, each biconnected component B contains exactly one vertex u that has no in-neighbor in B, and exactly one vertex r that has no out-neighbor in B. If B is trivial, then $u = r$, and otherwise r is the only reticulation vertex and there are exactly two edge-disjoint paths from u to r in B [57]. We refer to these two paths as *component paths*. The vertex u will be called the *root* of B and denoted $\rho(B)$, and r will be called the *bottom* of B. We let $\mathcal{B}_1$ be the set of biconnected components of $\mathcal{N}_1$ and $\mathcal{B}_2$ be the set of biconnected components of $\mathcal{N}_2$. Finally for $i \in \{1, 2\}$, we denote $\rho(\mathcal{B}_i) = \{\rho(B) : B \in \mathcal{B}_i\}$, i.e. the set of roots in $\mathcal{B}_i$.

Using dynamic programming, we construct a table M whose rows are the roots in $\rho(\mathcal{B}_1)$ and whose columns are the roots in $\rho(\mathcal{B}_2)$. For $u \in \rho(\mathcal{B}_1), v \in \rho(\mathcal{B}_2)$, we define $\mathcal{N}_u$ as the subnetwork of $\mathcal{N}_1$ rooted at u, and $\mathcal{N}_v$ as the subnetwork of $\mathcal{N}_2$ rooted at v. We then define $M[u, v]$ as the number of *leaves* in a MACRS-Simple of $\mathcal{N}_u$ and $\mathcal{N}_v$. If u is a tree vertex, its children are denoted u_1 and u_2.

In $\mathcal{N}_1$, we denote the two component paths on the same non-trivial biconnected component by $\pi_l^1 = p_{l,1}^1 \ldots$ and $\pi_r^1 = p_{r,1}^1 \ldots$ and in $\mathcal{N}_2$ these paths will be denoted $\pi_l^2 = p_{l,1}^2 \ldots$ and $\pi_r^2 = p_{r,1}^2 \ldots$. For a vertex p_i on path $\pi = p_1 \ldots$, let $h(p_i)$ be the child vertex of p_i such that $h(p_i) \neq p_{i+1}$. In other words, the edge $(p_i, h(p_i))$ is a bridge pendant π leading to a different biconnected component where $h(p_i)$ is rooting a distinct subnetwork. See Figure 6.3 for an illustration of the component paths and the described labelings for an example $\mathcal{N}_1$ network.

We use Algorithm 4 to compute $M[u, v]$ for each $u \in \rho(\mathcal{B}_1), v \in \rho(\mathcal{B}_2)$ in postorder. The algorithm considers the possible cases for u and v. If one of them is a leaf, say u, then either we can reduce $\mathcal{N}_v$ to a leaf whose label intersects with u, or not (line 3). If u and v are trivial components, with children u_1, u_2 and v_1, v_2, respectively, an agreement subnetwork can either be obtained by "combining" agreement subnetworks between $\mathcal{N}_{u_1}, \mathcal{N}_{v_1}$ and between $\mathcal{N}_{u_2}, \mathcal{N}_{v_2}$, or "combining" agreement subnetworks between $\mathcal{N}_{u_1}, \mathcal{N}_{v_2}$ and between $\mathcal{N}_{u_2}, \mathcal{N}_{v_1}$. These cases are evaluated on line

Algorithm 4

Input: Two networks $\mathcal{N}_1$, $\mathcal{N}_2$, vertices $u \in \rho(\mathcal{B}_1), v \in \rho(\mathcal{B}_2)$
Output: $M[u, v]$

if both u and v are trivial components **then**
  if u or v is a leaf **then**

$$M[u,v] = \begin{cases} 1 & \text{if } X(u) \cap X(v) \neq \emptyset \text{ and } R(u) \cup R(v) = \emptyset \\ -\infty & \text{otherwise} \end{cases}$$

  else
    for each $i \in \{1,2\}, j \in \{1,2\}$, define $X_{ij} = X(u_i) \cap X(v_j)$
    $M[u, v] = max(M_1, M_2)$ where

$$M_1 = \begin{cases} 1 & \text{if } X_{11} \neq \emptyset \text{ and } X_{22} = \emptyset \text{ and } R(u) \cup R(v) = \emptyset \\ 1 & \text{if } X_{11} = \emptyset \text{ and } X_{22} \neq \emptyset \text{ and } R(u) \cup R(v) = \emptyset \\ M[u_1, v_1] + M[u_2, v_2] & \text{if } X_{11} \neq \emptyset \text{ and } X_{22} \neq \emptyset \\ -\infty & \text{otherwise} \end{cases}$$

$$M_2 = \begin{cases} 1 & \text{if } X_{12} \neq \emptyset \text{ and } X_{21} = \emptyset \text{ and } R(u) \cup R(v) = \emptyset \\ 1 & \text{if } X_{12} = \emptyset \text{ and } X_{21} \neq \emptyset \text{ and } R(u) \cup R(v) = \emptyset \\ M[u_1, v_2] + M[u_2, v_1] & \text{if } X_{12} \neq \emptyset \text{ and } X_{21} \neq \emptyset \\ -\infty & \text{otherwise} \end{cases}$$

  end if
else if u is a trivial biconnected component and v is in a non-trivial biconnected component (or vice versa) **then**
  $M[u, v] = -\infty$
else u and v are in non-trivial components with reticulations r_1, r_2 respectively and component paths π_l^1, π_r^1, π_l^2, π_r^2
  $M_1 = -\infty$
  $M_2 = -\infty$
  if $|\pi_l^1| = |\pi_l^2|$ and $|\pi_r^1| = |\pi_r^2|$ **then**

$$M_1 = M[r_1, r_2] + \sum_{i=1}^{|\pi_l^1|} M[h(p_{l,i}^1), h(p_{l,i}^2)] + \sum_{i=1}^{|\pi_r^1|} M[h(p_{r,i}^1), h(p_{r,i}^2)]$$

  end if
  if $|\pi_l^1| = |\pi_r^2|$ and $|\pi_r^1| = |\pi_l^2|$ **then**

$$M_2 = M[r_1, r_2] + \sum_{i=1}^{|\pi_l^1|} M[h(p_{l,i}^1), h(p_{r,i}^2)] + \sum_{i=1}^{|\pi_r^1|} M[h(p_{r,i}^1), h(p_{l,i}^2)]$$

  end if
  $M[u, v] = \max(M_1, M_2)$
end if

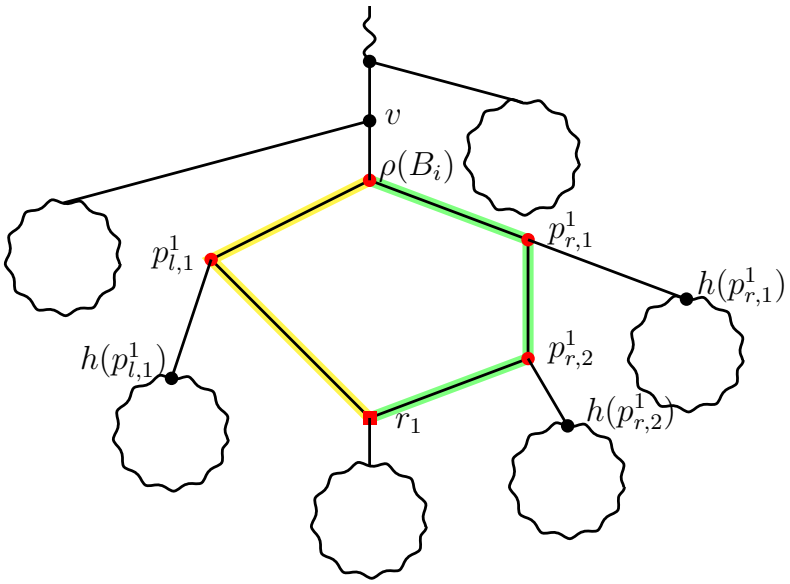

Figure 6.3: In this figure, tree vertices are filled circles and the reticulation is a filled square. A subnetwork is represented by a large open blob. Vertices in red are in the same non-trivial biconnected component. Yellow edges are on path π^1_l and green edges are on path π^1_r. Tree vertex v is a trivial biconnected component itself such that $R(v) \neq \emptyset$, it contains at least r_1.

6, with special cases when such combinations are not possible. When u is trivial and not v (or vice-versa), no such construction is possible and line 9 returns $-\infty$. When u and v are non-trivial, instead of mapping children of u and v as in the trivial component case, we map the component paths under u and v that lead to their descending reticulation (there may be two ways to map these paths). The smaller subnetworks to be combined are those "dangling" outside of these paths, and the possibilities are calculated on lines 11-19 (including the possibility that the whole expression evaluates to $-\infty$ if any term in the sum of either line 14 or 17 does).

We seek the result $M[\rho(\mathcal{N}_1), \rho(\mathcal{N}_2)] + |R(\mathcal{N}_1)|$ as M records only the number of leaves in an MACRS-Simple of $\mathcal{N}_1$ and $\mathcal{N}_2$. From this information we can calculate more about the general size of the network because they are binary, so $|V(\mathcal{N})| = 2|L(\mathcal{N})| + 2|R(\mathcal{N})| - 1$. Luckily, the number of reticulations in the solution is known ahead of time since it must have the same number of reticulations as each of the inputs. Note that we can also reconstruct the network that corresponds to the optimal size of the MACRS-Simple of $\mathcal{N}_1$ and $\mathcal{N}_2$ by performing a traceback in the dynamic programming table.

Theorem 15. *Algorithm 4 runs in time* $O(|V(\mathcal{N}_1)||V(\mathcal{N}_2)|(|V(\mathcal{N}_1)| + |V(\mathcal{N}_2)|))$.

Proof. The algorithm fills a table M, a table of maximum size $|V(\mathcal{N}_1)||V(\mathcal{N}_2)|$, thus if we can show that each table entry is calculated in at most linear $(|V(\mathcal{N}_1)| + |V(\mathcal{N}_2)|)$

time, then the algorithm is cubic as claimed.

The preprocessing step to determine and label biconnected components is linear as it requires a modified depth-first search [113]. Then, the calculations being performed for lines 1 through 9 consist of finding and checking the labelled components (linear), and checking up to 12 set intersections (linear) of a vertex's descendants leaves (linear to find). Lines 10 and on perform a linear number of table lookups/calls. The paths themselves are also linear to find as they are simply the paths that leave each child of the rooting vertex of the biconnected component and end on the next reticulation, the length of which can also be calculated on a single pass. Thus the claim holds. □

Lemma 11 and Corollary 2 are required to justify the arguments in the proof for Theorem 16, which is stated later.

Lemma 11. *Let $\mathcal{N}$ be a network and S be any CS on $\mathcal{N}$. If, for $r \in R(\mathcal{N})$, if $r \in \mathcal{N}\langle S\rangle$, then $v \in reach^-(r,\mathcal{N}) \subseteq \mathcal{N}\langle S\rangle$.*

Proof. Assume that there exists an $r \in \mathcal{N}$ such that $r \in \mathcal{N}\langle S\rangle$ and $v \in reach^-(r,\mathcal{N})$ but $v \notin \mathcal{N}\langle S\rangle$. First, v is not a leaf as a leaf cannot be above any other vertex. By assumption, there must be some cherry in S that removes v, say $S_i = (x,y)$ such that either $v = p(x)$, $v = p(y)$, or $v = p(x) = p(y)$ in $\mathcal{N}\langle S_{(0:i)}\rangle$. Since $r \in \mathcal{N}\langle S\rangle$, we have that $r \in \mathcal{N}\langle S_{(0:i)}\rangle$ and since $v \in reach^-(r,\mathcal{N})$ we have that $v \in reach^-(r,\mathcal{N}\langle S_{(0:i)}\rangle)$. In $\mathcal{N}\langle S_{(0:i)}\rangle$, because v has a non-leaf descendant r, v cannot be the parent of two leaves, but it must be the parent of one leaf to be in a cherry and therefore reduced by S_i. Thus, the only orientation we can have is $v = p(y)$ or $v = p(x)$ in the reticulated cherry $(x,y) \in \mathcal{N}\langle S_{(0:i)}\rangle$. In either case, we have that $v = r$ or $v = p(r)$ $(p(x) = r)$, and both cases imply that r is removed by the cherry S_i, contradicting that $r \in \mathcal{N}\langle S\rangle$. □

Note that Lemma 11 implies that after any simple reduction by a CS S on a network $\mathcal{N}$, that since all $r \in R(\mathcal{N}) \cap V(\mathcal{N}\langle S\rangle)$ then $\bigcup_{r\in R(\mathcal{N})} reach^-(r,\mathcal{N}) \subseteq \mathcal{N}\langle S\rangle$. This is to say, all vertices above any reticulation in either input network is retained after a simple reduction, these vertices must therefore align should a MACRS-Simple exist. Noting this, it follows that

Corollary 2. *For all networks $\mathcal{N}_1$, $\mathcal{N}_2$ and $\forall\, \mathcal{N}^* = \text{MACRS-Simple}(\mathcal{N}_1,\mathcal{N}_2)$ such that $\mathcal{N}^*$ is non-null, There exists an edge-preserving bijective function*

$$f : \bigcup_{r\in R(\mathcal{N}_1)} reach^-(r,\mathcal{N}_1) \to \bigcup_{r\in R(\mathcal{N}_2)} reach^-(r,\mathcal{N}_2)$$

.

Theorem 16. *The entry $M[\rho(\mathcal{N}_1),\rho(\mathcal{N}_2)]$ correctly contains $|L(\mathcal{N}^*)|$ for $\mathcal{N}^*$ a* MACRS-Simple *of $\mathcal{N}_1$ and $\mathcal{N}_2$ if one exists, and $-\infty$ otherwise.*

Proof. Given input networks $\mathcal{N}_1$ and $\mathcal{N}_2$, for any $u \in \rho(\mathcal{B}_1), v \in \rho(\mathcal{B}_2)$, let the subnetwork of $\mathcal{N}_1$ rooted on u be called $\mathcal{N}_u$ and let the subnetwork of $\mathcal{N}_2$ rooted on v be called $\mathcal{N}_v$. We claim that $M[u, v]$ as we defined it always contains the number of leaves of a MACRS-Simple between $\mathcal{N}_u$ and $\mathcal{N}_v$. In particular, this will show our desired result with $u = \rho(\mathcal{N}_1), v = \rho(\mathcal{N}_2)$. The proof is by induction on $|V(\mathcal{N}_u)| + |V(\mathcal{N}_v)|$.

As a base case, suppose that $|V(\mathcal{N}_u)| = 1$ and $|V(\mathcal{N}_v)| = 1$, i.e. u and v are both leaves. If $X(u) \cap X(v) \neq \emptyset$, we put $M[u, v] = 1$, which is correct since $\mathcal{N}_u$ and $\mathcal{N}_v$ are weakly isomorphic. If $X(u) \cap X(v) = \emptyset$, we put $M[u, v] = -\infty$, which is correct since there is no MACRS-Simple between networks that do not share a leaf label.

Let us now consider the inductive step. For the rest of the proof, for any $u' \in V(\mathcal{N}_u), v' \in V(\mathcal{N}_v)$, $u' \neq u$, $v' \neq v$, we denote by $\mathcal{N}^*_{u',v'}$ a MACRS-Simple between $\mathcal{N}_{u'}$ and $\mathcal{N}_{v'}$, if one exists (otherwise, $\mathcal{N}^*_{u',v'}$ is undefined). As an inductive hypothesis, we assume that $M[u', v']$ is the number of leaves in $\mathcal{N}^*_{u',v'}$, for any u', v' such that $|V(\mathcal{N}_{u'})| + |V(\mathcal{N}_{v'})| < |V(\mathcal{N}_u)| + |V(\mathcal{N}_v)|$.

The proof is split into cases.

Case: u and v are trivial, one is a leaf. If $X(u) \cap X(v) \neq \emptyset$ and $R(u) \cup R(v) = \emptyset$ then $M[u, v] = 1$. This is correct since a complete reduction can proceed on both networks without any reticulations and with at least one leaf in common, a requirement for any network to be isomorphic with a single-leaf network. Moreover, there cannot be more than one leaf since u or v is itself a leaf. For the same reason, when $X(u) \cap X(v) = \emptyset$, $M[u, v] = -\infty$ is obviously correct. When $R(u) \cup R(v) \neq \emptyset$, $M[u, v] = -\infty$ is correct because a MACRS-Simple of u and v can only be a leaf, but this cannot be achieved since one of the networks has an unremovable reticulation.

Case: u and v are trivial and $R(u) \cup R(v) = \emptyset$, and at least one of the following is true ($X(u_1) \cap X(v_1) \neq \emptyset$ and $X(u_2) \cap X(v_2) = \emptyset$) or ($X(u_1) \cap X(v_1) = \emptyset$ and $X(u_2) \cap X(v_2) \neq \emptyset$) or ($X(u_1) \cap X(v_2) \neq \emptyset$ and $X(u_2) \cap X(v_1) = \emptyset$) or ($X(u_1) \cap X(v_2) = \emptyset$ and $X(u_2) \cap X(v_1) \neq \emptyset$). In this case, $M[u, v] = 1$ by line 6. Neither u nor v have reticulations below them, thus any reduction may proceed. In fact, the reduction on $\mathcal{N}_u$ and $\mathcal{N}_v$ must be complete to obtain an isomorphic network since there is no shared leaf below one child of u and one child of v, thus that child must be removed to reach any MACRS-Simple($\mathcal{N}_u$, $\mathcal{N}_v$), requiring a cherry on u and v. Luckily, there is a leaf shared below one child of u and one child of v and so a single-leaf isomorphic network is possible, so this case is correct.

Case: u and v are trivial and $R(u) \cup R(v) \neq \emptyset$, and both of the following are true, ($X(u_1) \cap X(v_1) = \emptyset$ or $X(u_2) \cap X(v_2) = \emptyset$) and ($X(u_1) \cap X(v_2) = \emptyset$ or $X(u_2) \cap X(v_1) = \emptyset$). In this case line 2 resolves to false so we calculate $M[u, v]$ on line 6. We find that $M[u, v] = -\infty$ since $R(u) \cup R(v) \neq \emptyset$. A complete reduction is required to reach the required isomorphic single-leaf in this condition, but the presence of a reticulation prevents this.

Case: u is trivial and v is not trivial or v is trivial and u is not trivial. In this case

we always resolve $M[u, v] = -\infty$ by line 8 and 9. This is indeed the correct case, the presence of a reticulation in $\mathcal{N}_v$ and not in $\mathcal{N}_u$ (or in $\mathcal{N}_u$ and not $\mathcal{N}_v$) makes an isomorphic network unreachable by simple reductions alone.

Case: u and v are trivial, both are not leaves, and at least one of the following is true, ($X(u_1) \cap X(v_1) \neq \emptyset$ and $X(u_2) \cap X(v_2) \neq \emptyset$) or ($X(u_1) \cap X(v_2) \neq \emptyset$ and $X(u_2) \cap X(v_1) \neq \emptyset$). In this case we calculate $M[u, v]$ on line 6. Regardless of any reticulations that may be below u or v, we put $M[u, v]$ as the maximum between $M[u_1, v_1] + M[u_2, v_2]$ and $M[u, v] = M[u_1, v_2] + M[u_2, v_1]$. We can assume, by the inductive hypothesis, that $M[u_1, v_1] = |L(\mathcal{N}^*_{u_1,v_1})|$ and $M[u_2, v_2] = |L(\mathcal{N}^*_{u_2,v_2})|$ (likewise $M[u_1, v_2] = |L(\mathcal{N}^*_{u_1,v_2})|$ and $M[u_2, v_1] = |L(\mathcal{N}^*_{u_2,v_1})|$). It is not difficult to see that if $\mathcal{N}^*_{u,v}$, a MACRS-Simple of $\mathcal{N}_u$ and $\mathcal{N}_v$, exists, then it can be obtained by joining a MACRS-Simple of $\mathcal{N}_{u_1}, \mathcal{N}_{v_1}$ with a MACRS-Simple of $\mathcal{N}_{u_2}, \mathcal{N}_{v_2}$ under a common parent, or by joining a MACRS-Simple of $\mathcal{N}_{u_1}, \mathcal{N}_{v_2}$ with a MACRS-Simple of $\mathcal{N}_{u_2}, \mathcal{N}_{v_1}$ under a common parent. In the current case, $M_1 = M[u_1, v_1] + M[u_2, v_2]$ and $M_2 = M[u_1, v_2] + M[u_2, v_1]$ correspond to constructing these two possible networks, and since they contain the correct values by induction, $M = \max(M_1, M_2)$ is correct.

Case: u and v are both non-trivial and $M[r_1, r_2] \neq -\infty$ for reticulation r_1, r_2 in u's, v's biconnected components respectively, and $M[h(p^1_i), h(p^2_i)] \neq -\infty$ for all i in any $p^1 \in \pi^1_l, \pi^1_r$ or $p^2 \in \pi^2_l, \pi^2_r$. In this case we resolve

$$M[u, v] = M[r_1, r_2] + \sum_{i=1}^{i=|\pi^1_l|} M[h(p^1_{l,i}), h(p^2_{l,i})] + \sum_{i=1}^{i=|\pi^1_r|} M[h(p^1_{r,i}), h(p^2_{r,i})]$$

or

$$M[u, v] = M[r_1, r_2] + \sum_{i=1}^{i=|\pi^1_l|} M[h(p^1_{l,i}), h(p^2_{r,i})] + \sum_{i=1}^{i=|\pi^1_r|} M[h(p^1_{r,i}), h(p^2_{l,i})]$$

.

Each table reference in this summation returns a value that is correct for that subnetwork, by the induction hypothesis, as every proper subnetwork $\tilde{\mathcal{N}}$ of $\mathcal{N}_u$ and $\mathcal{N}_v$ is smaller. The summation itself is also correct. This is evident by noting that the biconnected components on u and v only contain vertices along π_l and π_r [57], so all vertices in the components are accounted for. Furthermore, all bridges must lead to disjoint networks. Finally, by Corollary 2 the only possible networks are constructed by joining up child networks that pair vertices in the order evident by the component paths and their independent subnetwork children/siblings. Since we consider the maximum among all such possible networks, the solution is maximal. It is for this same reason that when u and v are non-trivial but the conditions are such that $M[u, v] = -\infty$ by line 19 or by $M_1 = -\infty$ and $M_2 = -\infty$, that $M[u, v] = -\infty$ is correctly found.

□

We have finally gathered all the necessary ingredients to derive our main result.

Theorem 17. *Let $\mathcal{N}_1, \mathcal{N}_2$ be two networks, let $n = \max(|V(\mathcal{N}_1)|, |V(\mathcal{N}_2)|)$, and $r = |R(\mathcal{N}_1)| + |R(\mathcal{N}_2)|$. Then the* MACRS *problem can be solved in time $O(3^r n^3)$.*

Proof. By Theorem 14, Algorithm 2 can enumerate all reticulation-trimmed subnetworks of $\mathcal{N}_1$ and $\mathcal{N}_2$ in total time $O(3^{|R(\mathcal{N}_1)|}n + 3^{|R(\mathcal{N}_2)|}n) = O(3^r n)$. The number of pairs of such networks for which we compute a MACRS-Simple is $O(3^r)$, each of which can be handled in time $O(n^3)$ by Theorem 15. The total running time is thus $O(3^r n + 3^r n^3) = O(3^r n^3)$.

□

6.5 Counting the number of reticulation-trimmed subnetworks

One immediate optimization is to only calculate MACRS-Simple on networks satisfying $|R(\mathcal{N}_1)| = |R(\mathcal{N}_2)|$, as otherwise no solution exists. One further improvement would be to only enumerate pairs of networks $\mathcal{N}_1$ and $\mathcal{N}_2$ that have the same relative hierarchy of reticulations, something that Corollary 2 implies should also be shared between networks for a solution to MACRS-Simple to exist. Although these filters should be used in practice, including them in our analyses does not improve the theoretical complexity of our algorithms.

However, 3^r is a pessimistic bound on the number of reticulation-trimmed subnetworks that we must enumerate. This bound can be reached in some cases, but does not reflect that the hierarchical reticulation relationship affects the number of possible reticulation-trimmed subnetworks. With this in mind, we introduce a new graph structure to better understand this relationship. These can be useful in practice to estimate in advance the time that Algorithm 1 will take on a given pair of networks.

First, let $\mathcal{N}$ be a network. We define a new graph, later shown to be a tree, that represents a compressed version of $\mathcal{N}$ and serves to describe the ancestry relationships between reticulations in $\mathcal{N}$. Specifically, the *dependency tree* T of a network $\mathcal{N}$ is, in essence, the Hasse diagram of the partially ordered set on $R(\mathcal{N}) \cup \{\rho(\mathcal{N})\}$ defined by the ancestry relation. Formally, the dependency tree T of $\mathcal{N}$ is as follows:

- $V(T) = R(\mathcal{N}) \cup \{\rho(\mathcal{N})\}$
- for distinct u, v in $V(T)$, $(u, v) \in E(T)$ if and only if $v \in \mathit{reach}(u, \mathcal{N})$ and there is no $x \in R(\mathcal{N}) \setminus \{u, v\}$ such that $x \in \mathit{reach}(u, \mathcal{N})$ and $v \in \mathit{reach}(x, \mathcal{N})$. In other words, edge (u, v) exists in T when there are no other reticulations on any directed path between u and v.

Note how for a network without reticulations, the resulting dependency tree consists of only one vertex, $\rho(\mathcal{N}) = \rho(T)$. We call such a dependency tree a *singleton tree*. At a high level, the fact that T is a tree on level-1 networks can be seen as

follows. By taking the set of non-trivial biconnected components of a network and contracting them, one obtains a tree. Since these components correspond to reticulations in a level-1 network, T simply depicts the ancestry relationships of reticulations. For completeness, we provide a detailed proof below.

Proposition 1. *The dependency tree T of any level-1 network $\mathcal{N}$ is a tree rooted on $\rho(\mathcal{N})$.*

Proof. In the case that $|V(T)| \leq 2$, the claim holds, so assume that $|V(T)| > 2$. First, observe that in $\mathcal{N}$, each vertex of $R(\mathcal{N})$ has $\rho(\mathcal{N})$ as an ancestor. So, for any vertex $y \neq \rho(\mathcal{N})$ in T, if there is no edge (x, y) for any vertex $x \neq \rho(\mathcal{N})$, then there is at least edge $(\rho(N), y)$. Therefore, every vertex except $\rho(\mathcal{N})$ has at least one in-neighbor in T. In particular, the underlying undirected graph of T is connected and its only vertex of in-degree 0 is $\rho(\mathcal{N})$.

Note that $(u, v) \in E(T)$ implies a directed path in $\mathcal{N}$ from u to v. This implies, in particular, that there is no directed cycle in T since it would imply the existence of a directed cycle in $\mathcal{N}$, forbidden by our definition of a network.

Next, we show that every vertex in T except $\rho(T)$ has *exactly* one in-neighbour. Combined with the facts that the root has in-degree 0 and that T has no directed cycle, this will show that T must be a tree whose edges are all directed away from the root. Towards a contradiction, assume there is a vertex $z \in V(T)$ such that z has in-neighbours x and y, where $x \neq y$ since there are no parallel edges in T by definition. In $\mathcal{N}$ there are the paths $x \to z$ and $y \to z$. Note that z is the only vertex that occurs on both paths, since if there was some other vertex a on both paths, then a would be a reticulation itself, which contradicts the definition of T. Next, we observe that x and y have some *least common ancestor* in $\mathcal{N}$, so let w be any vertex such that $w \in reach^-(x, \mathcal{N}) \cap reach^-(y, \mathcal{N})$ and such that no other vertex in $reach(w, \mathcal{N})$ has this property. We have now established that in $\mathcal{N}$, there is the undirected cycle $(w...y, z, x, ...w)$. Since x, y, z are in T, they are in $R(\mathcal{N}) \cup \{\rho(\mathcal{N})\}$, and at least two of them are reticulations on the same undirected cycle in $\mathcal{N}$. This contradicts the assumption that $\mathcal{N}$ is level-1. □

We believe this proof holds for level-2 networks and provide an intuitive explanation here. Of the three vertices shared by T and the cycle in $\mathcal{N}$ mentioned in the above proof, one can show through case analysis that none is $\rho(\mathcal{N}) = \rho(T)$, and therefore that $\mathcal{N}$ must be at least level-3.

A *subdependency tree* T' of a dependency tree T on a network $\mathcal{N}$ is any subtree of T that contains $\rho(T)$.

Observation 1. *Let $\mathcal{N}$ be a network with dependency tree T. For each subdependency tree T' of T, there is some reticulation-trimmed subnetwork $\mathcal{N}'$ of $\mathcal{N}$ whose dependency tree is T'. Conversely, the dependency tree of any reticulation-trimmed subnetwork $\mathcal{N}'$ of $\mathcal{N}$ is a subdependency tree of T.*

The first direction of this observation is evident when we consider that cherry reductions are performed in a bottom-up fashion where no reticulation can be removed before a complete reduction is performed below it, and a subdependency tree is connected and contains $\rho(T)$ by definition. In the converse, any dependency tree of a reticulation-trimmed subnetwork $\mathcal{N}'$ of $\mathcal{N}$ can only contain vertices that are some subset of the vertices of T. Also, since any dependency tree is a connected tree containing $\rho(\mathcal{N})$ by definition, there is some subdependency tree of T that serves as the dependency tree of $\mathcal{N}'$.

With these definitions in place, we show how they provide an intuitive bound on the size of the Reticulation-Trimmed Enumeration set and therefore to the runtime of a specific input network to MACRS.

We demonstrate this in two phases. Let $\mathcal{N}$ be a network with dependency tree T. Then we can ask,

1. How many unique dependency trees are encountered while enumerating all reticulation-trimmed subnetworks? In light of Observation 1, this is akin to asking how many subdependency trees there are on T.

2. Given a subdependency tree T' of T, how many reticulation-trimmed subnetworks of $\mathcal{N}$ have T' as a dependency tree?

We answer each of these two questions in the remainder of this section, starting with Question 1. Though there are existing works which list and count subtrees of a tree [114], which enumerate through subtrees of size k [115], or which find "interesting" subtrees [116], we have been unable to find an expression for the number of subtrees of a tree that contain the root (note that here, we treat all vertices as labelled, so that isomorphic subtrees with different sets of vertices are both included in the count).

Therefore, to answer Question 1, first let T_u, for $u \in V(T)$, denote the subtree of tree T that is rooted on vertex u, i.e. the subgraph of T induced by $reach(u, T)$. Let $c(u, T)$ denote the set of children of vertex u in tree T.

Next, define $q(T_u)$ as number of subtrees of T_u that either contain u, or contain no vertices at all. To answer Question 1, simply notice that the number of subdependency trees of a dependency tree T is $q(T_{\rho(T)}) - 1$ (subtracting one since the empty tree is not a valid subdependency tree while all subtrees containing $\rho(T)$ are).

Proposition 2. *For a dependency tree T and $u \in V(T)$, it holds that*

$$q(T_u) = \begin{cases} 2 & \text{if } reach(u, T) = \{u\} \\ 1 + \Pi_{v \in c(u, T_u)} q(T_v) & \text{otherwise} \end{cases}$$

Proof. We proceed by induction on $|V(T_u)|$. In the base case $|V(T_u)| = 1$. In this case, the number of subtrees of T_u that contain u or no vertex is 2, since our only

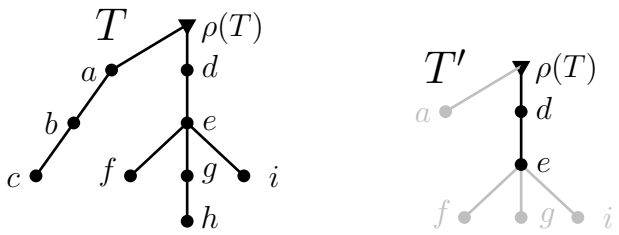

Figure 6.4: An example of a dependency tree T of some network $\mathcal{N}$ (not shown) and a subdependency tree (only black vertices/edges) T' of T. The grey vertices in T' denote the vertices that would be included in the set R for the calculation of $r(T', \mathcal{N})$. Note how R includes vertices that are the "absent" children of any and all vertices of T', but not more.

options are to keep u or not. Then, $q(T_u) = 2$ since a singleton tree T_u has, for u, $reach(u, T) = \{u\}$. So the proposition holds in the base case.

Next assume that $|V(T_u)| > 1$ and that the statement is true for any $v \in V(T)$ such that $|V(T_v)| < |V(T_u)|$. It is clear that u is present on all subtrees of T_u rooted on u and calculating the number of such subtrees amounts to taking the combination of, for each child v of u, all subtrees of T_v that either contain v or no vertex. Note that, for each $v \in c(u, T_u)$, $|V(T_v)| < |V(T_u)|$, so $q(T_v)$ correctly returns the size of the set containing the empty tree and subtrees of T_v rooted on v. After taking the described combination, we must add one to the size of the set to account for the empty tree. Thus we correctly find $q(T_u) = 1 + \Pi_{v \in c(u,T_u)} q(T_v)$ and the proposition holds. □

Next, we address Question 2. For a network $\mathcal{N}$ with dependency tree T, and for a subdependency tree T' of T, define $r(T', \mathcal{N})$ as the number of reticulation-trimmed subnetworks of $\mathcal{N}$ whose dependency tree is T'. Although the exact value depends on the structure of $\mathcal{N}$, we may give an upper bound that is likely to be tight in most cases. See Figure 6.4 for a visual representation of the definition of R defined in the following proposition.

Proposition 3. *Let T' be a subdependency tree T of network $\mathcal{N}$. Then*

$$r(T', \mathcal{N}) \leq 2^{|R|} \text{ where } R = \bigcup_{v \in V(T')} (c(v, T)) \setminus V(T')$$

Proof. Each $u \in R$ is absent from T', so it is set to be reduced on $\mathcal{N}$ in order to reach a reticulation-trimmed subnetwork with dependency tree T'. The reduction of u can proceed in up to two ways, by one reticulated cherry on each of the reticulation edges leading into u. Each of these options could lead to a distinct network. Since these two options are present for each $u \in R$, there are up to $2^{|R|}$ combinations of choices for these reticulations.

Furthermore, for any reticulation $v \in reach(u, \mathcal{N})$ (which is also not present in T'), the choice of reducing by either reticulation edge leading in to v is irrelevant for

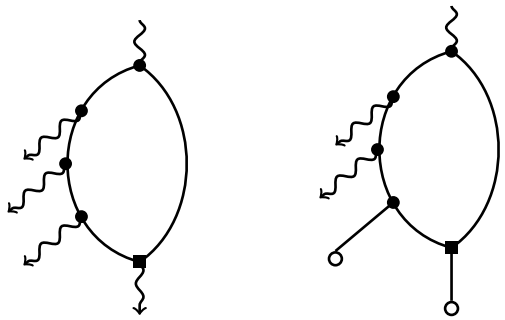

Figure 6.5: An example biconnected component of a network (left) and the same biconnected component with the only possible reticulated cherry present (right). We see in this example how some reticulations confer only one choice in how they are reduced, thus $r(T', \mathcal{N})$ is not bounded from below by the formula given in Proposition 3.

the reduction of u since the subnetwork below u must undergo a complete reduction for u to be in a reticulate cherry. Thus, only the reduction choice on the direct children of vertices of T' need be accounted for. Therefore the size of the set R as defined is the correct exponential factor. □

To demonstrate how $r(T', \mathcal{N})$ does not provide a lower bound on the number of reticulation-trimmed subnetworks of $\mathcal{N}$ that have subdependency tree T' as a dependency tree, we include Figure 6.5 for an example of a case where there is only one choice of reticulated cherry reduction for a given reticulation set to be reduced. It appears plausible that this is the only situation in which our upper bound is not achieved.

We have shown how, for both questions, the size of the set Reticulation-Trimmed Enumeration is bounded by the number and degree of high-degree vertices. In both questions, we see that high-degree node v in a subdependency tree T' of a dependency tree T may produce exponentially more "options" for unique reticulation-trimmed subnetworks. This intuition is confirmed when looking at two extreme cases, as illustrated in Figure 6.6. Consider a network $\mathcal{N}$ that has r independent reticulations, i.e. such that its dependency tree T is a star with r leaves. In this case, Proposition 2 shows that T has 2^r subdependency trees, one for each possible subset of leaves of T to remove. Then Proposition 3 shows that for each subdependency tree with $k \in \{0, 1, \ldots, r\}$ leaves removed, there are up to 2^k subnetworks to enumerate (which can be argued to be the exact bound in this case). The number of subnetworks to enumerate is then $\sum_{k=0}^{r} \binom{r}{k} 2^k = 3^r$, as expected from our complexity analysis.

At the other extreme, suppose that $\mathcal{N}$ consists of r reticulations that are "chained", so that T is a path of $r+1$ vertices. In this case, Proposition 2 gives $r+1$ subdependency trees. By Proposition 3, each of those can be achieved by $2^{|R|} = 2$ reticulation-trimmed subnetworks for the case where $|R| = 1$ or $2^{|R|} = 1$ reticulation-trimmed

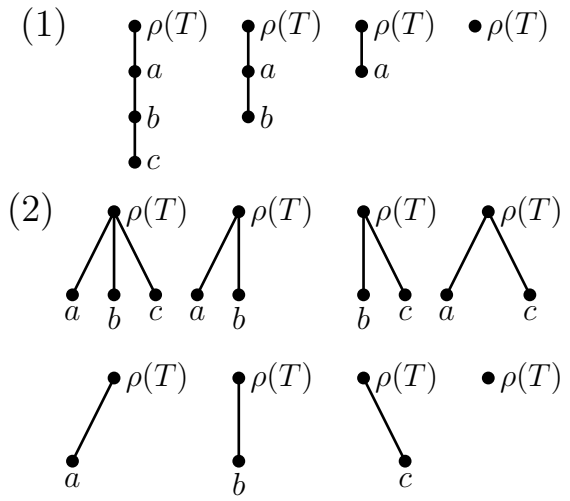

Figure 6.6: (1) On the left, we have the dependency tree of a network whose three reticulations are all of a descendant/ancestor relationship with each other. The four graphs from left to right include all subdependency trees of the left dependency tree.
(2) On the upper left, we have the dependency tree of a network in which none of the three reticulations they represent is an ancestor or descendant of another. The eight graphs from upper left to bottom right are all subdependency trees of the upper left dependency tree.

subnetwork for the case where $|R| = 0$ (the subdependency tree is the dependency tree). This means that in total, the number of subnetworks to enumerate is only $2r + 1$. This demonstrates how much the structure of the network can impact the complexity of our algorithm.

6.6 Concluding remarks

6.6.1 Counting dependency trees

It is known that biconnected components of a network can be contracted to form a tree, and sometimes dynamic programming can be performed on such a structure. With this fact in the background of our work, it is a small step to our formulation of dependency trees, which we use in this work to refine our analysis of runtime. Preferred would be to have found a closed-form equation that counts subdependency trees, but lacking that our investigations proved to be fruitful in another way.

Immediately at start of Section 6.5 on counting the number of reticulation-trimmed subnetworks, we identify how the hierarchy of reticulations can be used as a prefilter to MACRS-Simple, a fact we use to our advantage in the next Chapter. To do so, there are a few pieces to connect. Namely, we start with the observation that there is an edge-preserving bijective function between vertices above all reticulations between two networks, if there is an MACRS-Simple between them (Corollary 2). During research and development, we called this structure the *forbidden subnetwork*, since it is forbidden to reduce. If we capture each possible forbidden subnetwork with each pair of isomorphic subdependency trees, what remains is how we enumerate reticulation-trimmed subnetworks (from the dependency tree) that supplies all the remaining possible options for MACRS-Simple. This, and more, is explored in the next work.

Chapter 7

Project 3: Optimization and experimentation

7.1 Front matter

7.1.1 Abstract

Phylogenetic networks are increasingly being used to represent more complex evolutionary relationships, such as hybridization and horizontal gene transfer. Calculating distances between networks measures the discrepancies between networks built using different methodologies or reference networks, in the evaluation of construction methods. Here we are interested in the cherry distance, a network distance based on cherry operations. We describe refinements made to a cherry distance algorithm design that takes advantage of a network abstraction that maps reticulated elements between two inputs and performs a preprocessing step. We experimentally show, on a newly implemented and publicly available Rust package, both the improvements this design provides, as well as an exploration of when such an improvement is most effective vis-a-vis the input network topology. Next we present a heuristic strategy to calculate the cherry distance in a non-exact way, and experimentally show how it maintains a very high degree of accuracy while still providing large gains in the runtime efficiency. Finally, we explore particular characteristics of the cherry distance through experiments on real data from the Rose family using another rearrangement operation (rNNI). Specifically, we show how the cherry distance excels at reflecting how many taxa are impacted by changes in the network. We also show a higher degree of sensitivity in cherry distance when compared to a network adaptation of the ubiquitous RF distance on trees, the soft RF distance.

7.1.2 Dissemination

This work was first presented at the 22nd satellite conference on comparative genomics (RECOMB-CG), colocated with the international conference Research in Computational Molecular Biology (RECOMB), delivered in person in April 2025 in Seoul, Korea. As of the time of this writing, the conference proceedings have not been published.

[117] K. Landry and O. Tremblay-Savard, "Fast calculation of cherry distance on level-1 orchard networks: optimization, heuristic and implementation," Presented at the 22nd RECOMB-CG, Seoul, South Korea, 2025

Source code is found at this GitHub repository: `https://github.com/KaariL/cherry_dist`.

7.1.3 Contribution to authorship

I realized the majority of this work, with oversight and consultation from my coauthor. I am the sole author of the code repository and performed all experimentation under the guidance of my coauthor. It was my coauthor who provided the critical suggestion of the strategy that became the main heuristic. I was responsible for the majority of writing and all diagrams and formatting.

During the writing of this manuscript, the work of Olivier Tremblay-Savard was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) Discovery program grant number RGPIN-2016-06051.

7.1.4 Manuscript modifications

Modifications bring notation in line with the standards of this document.

7.2 Introduction

7.2.1 Implementation details

Before the main work of this project began, I invested considerable time and energy in implementing the algorithm design detailed in the previous project. This major undertaking was a critical prerequisite to the main objective, making a practical and efficient software release for real use.

Rust, a general purpose programming language popular for systems programming, was selected for its emphasis on performance and its aggressive memory safety. Rust is also known for being quite user-friendly with its verbose and specific errors and excellent documentation. The Rust "trait" system was well-used when constructing

this program. Often compared to Java Interfaces, Rust traits are inspired by type classes in Haskell. Using traits allowed for generic typing (specifically in the data stored in network nodes) ensuring that functionality was guaranteed while leaving the real type flexible. Some of these traits were user-defined, allowing me to better handle how data was stored in structures (specifically the ability to optionally store data in a node at a certain id/index in the network).

The main data structures used to model the phylogenetic networks are the `Node` which contains its adjacencies, labels, and cluster, and the `Network` which contains all of the nodes and some helpful data such as a map identifying non-trivial biconnected components and the nodes therein. The node clusters are often compared, and so are stored as data structures supporting set operations. Calculations are often performed recursively on the children of a node, so to prevent repetition when reticulation nodes are encountered, the `Network` stores one reticulation edge, for each reticulation, as an "additional edge" i.e. the network is stored as a tree with additional edges. In this way, recursive calculations can be done on an embedded tree of the network while preserving the real network adjacencies that exist. This characterization is possible due to the fact that every level-1 network is a TCN, which are the class in which every embedded tree is a base tree (so which reticulate edge is "additional" can be selected arbitrarily). This is purely for computational reasons, the "additional edges" here do not represent HGT or have any other biological interpretation.

7.2.2 Random network generator initial design

Next, I will speak more deeply on the random network generator since it went through an important design change that helped build out a section in the following work. The philosophy of the base random network generator was to build a random tree, then add additional edges to reach a desired number of reticulations while maintaining an appropriate network structure. The characterization of an orchard as a tree with additional edges [50] inspires this.

We start in building a random tree by, at each internal vertex, deciding what fraction of the remaining children will be descendants of the left child, with the remainder descendants of its right. When only 1 descendant is available, that vertex becomes a leaf as a base case and a (string) label is applied to it either randomly generated or randomly selected from a predefined set. Once this tree is complete, the next stage is to attach an edge to the tree in order to create a network, accomplished by selecting two edges to bisect and connect. We can avoid creating cycles in a simple way. We check if a cycle is created, and if so then we reverse the direction of the additional edge to remove it. This is possible because we are assured that an edge added in this way spans distinct biconnected components, as described next.

Of course, we must not increase the level of the network beyond 1, so addition of network edges after the first one requires care. In particular, we observe that descendants of two distinct vertices in the same biconnected component would simply

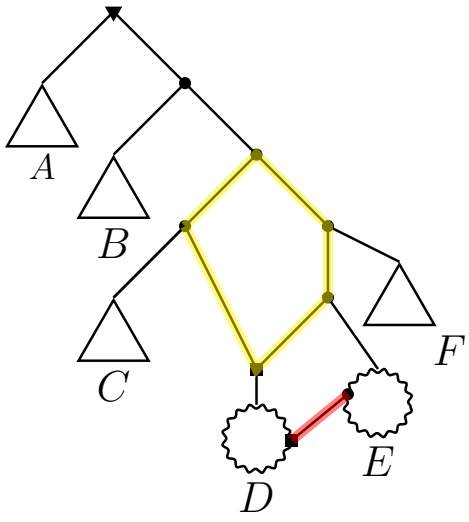

Figure 7.1: Subtrees with no reticulations are depicted with a large open triangle, subnetworks with one reticulation are depicted with cloud shapes. Highlighted edges, as well as some edges in D and E are not eligible to be joined. If an edge like the red edge were inserted, the level of the network would be increased to 4. All edges from the root down to the depicted nontrivial biconnected component, and down through all of A and B, constitute one group while all the edges in C and all the edges in F constitute the other two eligible groups, there may be other groups in D and E that are not shown.

expand this component if joined, and therefore increase the level of the network. To counter this, edges that are eligible to be joined are not already in a biconnected component and are grouped according to the closest ancestor in a non-trivial biconnected component. There is an additional group of vertices for which no such ancestor exists. Then, selection of two edges to join to create a new nontrivial biconnected component must be intragroup. See Figure 7.1. Because of this constraint, it is possible that a reticulation cannot be added to a network because no group has at least two edges. In this case, there are two options; we can choose to return the network as-is with less than the user-specified number of reticulations, or we can destroy the network and start the process again in order to achieve the exact amount of desired reticulations. A user-defined flag determines which behaviour happens.

Initial experiments proceeded with this base network generator, which compared the number of pairs produced for input to MACRS-Simple between our naive enumeration and our improved enumeration on the dependency trees (found post-hoc, after the random network is constructed). Our hypothesis, as per the analysis in Section 6.5, was that some cases would have a minimal (or no) effect, while others would see a grand effect. What we saw was, in all our tests, a nearly imperceptible effect. The number of pairs being produced by the different enumeration schemes was nearly identical, a far cry from what our analysis would suggest. It was at this point we categorized the topology of dependency trees being produced and noticed

a critical issue. Networks were being produced nearly always with the dependency tree topology of depth 1 (a star graph when undirected, then rooted on the internal vertex).

The random network generator thus required redesign. In its new iteration, the first step was to generate all possible dependency tree topologies on a given number of reticulations, then to choose randomly from among them to use as a base for the network construction. The new design is described in the paper. You will also see in Section 7.4.1 details of the effects of dependency tree topology on the running of the software, much of the content is inspired by this design experience. This design lets us employ a "worst case" test to accompany the experimental results.

7.2.3 Current landscape of distances

This section was included in the introduction of the original publication. While most of this content has been covered in Section 3.1, it serves as a succinct summary of the state of research on network distances, so it is reproduced here.

Network distance quantifies the difference between networks and are dominated by two general flavours, rearrangement-based and cluster-based metrics (Robinson-Foulds (RF) and related metrics). Some well-established rearrangement operations on trees that also define a distance metric include rooted subtree pruning and regrafting (rSPR) [91], tree bisection and reconnection (TBR) [92], and nearest-neighbour interchange operation (NNI) [89, 90]. All three of these rearrangements have fixed-parameter tractable (FPT) algorithms on trees parameterized by the distance themselves [92, 68], but none are considered fast enough for practical use. Each of these operations and metrics have been adapted to networks, the subnet prune and regraft (SNPR) on rooted networks [95, 94], the TBR on unrooted networks [94], the NNI for general unrooted networks [96], the rooted NNI (rNNI) on rooted networks [97] (no complexity changing moves) and local subnetwork transfer (LST) which defines an rNNI move on relevant classes of networks and two associated complexity-changing moves [3]. The RF metric on trees [78], the symmetric difference on leaf sets descending vertices between two input trees, by contrast, is computed more efficiently (linear time [78, 79]) and is therefore ubiquitous in practical use. Unfortunately, RF has a well-known limitation: it is overly sensitive to small changes, leading to large distances between relatively similar trees. There are many adaptations of RF to networks, however early adaptation was only shown to be a metric for certain restricted classes of networks [66, 80, 81]. Another variation of the RF for networks, the *soft RF distance*, takes the symmetric difference of leaf sets of descending vertices on trees embedded on the network. The soft RF stands apart from RF as being experimentally shown to be more fine-grained that standard RF [82].

7.2.4 Outline

In this work, we employ various practical strategies to improve on the runtime of an algorithm that calculates MACRS, and therefore cherry distance. The strategy of the optimization is to observe that the runtime of the algorithm is dominated by a subroutine that is called many times, so we aim to reduce the number of calls to this subroutine. In Section 7.3 of this work, after a high-level review of the algorithm design, we outline a method of data processing that acts as a filter to reduce unnecessary calls to the time-consuming subroutine. We then describe a heuristic strategy to further reduce subroutine calls based on a threshold number of its runs we are willing to consider, with accuracy relying on a priority ranking. To end this section, we describe how random networks are produced for the experimentation that follows in Section 7.4.1. The experimentation starts with a comparison of our new optimization with a naive method, then with a comparison of runtime. Experiments proceed with further runtime tests on our heuristic. In Section 7.4.2 we end with a demonstration, based on real data, of how the cherry distance, unlike other well-known rearrangement distances, is well-behaved in a few critical aspects; there is a strong correlation between it and how many taxa are affected by changes in the network topology, it is not overly sensitive to a rearrangement move, and gives well distributed distances between random networks.

Not many network distances have been implemented to our knowledge, so this software package and accompanying analysis are a meaningful contribution. Here we focus on level-1 networks. Many difficult problems become tractable on level-1 networks [2] because reticulations are isolated from each other. For example, we see results on statistical identifiably of phylogenetic networks, the ability to uniquely determine a network topology based on (for example) sequence data measured at the leaves, have focused on level-1 networks [118]. Moreover, it is worth noting that many existing network inference methods strictly consider the level-1 class of networks, such as PhyloNetworks [119, 60] and NANUQ [120].

7.3 Methodology

In this section we describe the existing MACRS algorithm, then we elaborate on how the required data enumeration (preprocessing) is improved. We further describe a heuristic strategy that ranks inputs on their promise to deliver a good final result. Finally, with some relevant definitions established, we describe how initial random networks are generated for the experimentation presented in the following section. Pseudocode of the complete algorithm with all elements described here is found in Appendix 9.1.1.

7.3.1 Exact algorithm design overview

This section provides a high level summary of the algorithm design from [112]. This work also has a corresponding conference proceeding [111], and open-source version [121]. The original algorithm design summarized here has a time complexity of $O(3^r n^3)$ where r is the combined number of reticulations in each network, and n is the maximum number of vertices in an input network.

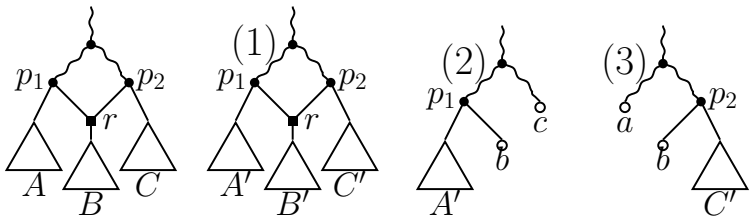

Figure 7.2: Showing the three possible options for a reticulation edge in a cherry-reduced network. Let r be a reticulation vertex in this network and r's parents are p_1 and p_2 where p_1 is neither the parent nor the child of p_2 (there would be two choices in that case). (1), (2), (3) depict the three choices for how r may appear on the way to an MACRS (i.e. at some point in the reduction). We show some minimum requirement in the case of (2), (3) or the final MACRS in the case of (1). To simplify the example, subtrees are represented by triangles to emphasize that we assume there are no reticulations in A, B, or C. The prime ($'$) indicates that subtree may or may not be reduced. Small open circles indicate leaf vertices that were required in order to make a cherry on r, assuming $a \in A$, $b \in B$, $c \in C$ in the original network.

The complexity of calculating cherry distance is in the reticulations. So, our strategy is to enumerate through all of the three possible outcomes that each reticulation of the input could take in a calculated MACRS (see Figure 7.2). So for an input network $\mathcal{N}$, for every specified set of possible outcomes on $R(\mathcal{N})$ (fixed by choosing a disjoint subset of reticulation edges to be reduced), we produce the set of associated minimally cherry-reduced subnetworks of $\mathcal{N}$ that accommodates the specification. More formally, we call a subnetwork of $\mathcal{N}$, $\mathcal{N}'$, a *reticulation-trimmed subnetwork* of $\mathcal{N}$ with respect to $F \subset E_R(\mathcal{N})$ when there exists a sequence of cherry reductions on $\mathcal{N}$ that produce $\mathcal{N}'$ such that all edges of F are removed, that all edges $E_R(\mathcal{N}) \setminus F$ remain, and only the minimum reduction was performed to satisfy this constraint. Refer to Figure 7.3 and see [112, Lem. 1] for more details on why these are appropriate structures to produce. We equally refer to such an $\mathcal{N}'$ as simply a reticulation-trimmed subnetwork of $\mathcal{N}$, and we can also refer to the set of all reticulation-trimmed subnetworks of $\mathcal{N}$, a set whose members are defined by all appropriate $F \subset E_R(\mathcal{N})$.

The next part of the algorithm involves a new sub-formulation of the MACRS problem, the *simple maximum agreement cherry-reduced subnetwork*, MACRS-Simple, that also asks for the largest common cherry-reduced subnetwork of the input net-

works, but for one found with only simple cherry reductions. For two input networks $\mathcal{N}_1$ and $\mathcal{N}_2$ we call the solution (if there is one) an MACRS-Simple of $\mathcal{N}_1$ and $\mathcal{N}_2$ or simply an MACRS-Simple.

The algorithm to calculate MACRS consists of two main steps, that is proved to output a correct result in [112, Thm. 2]:

1. enumerate each set of reticulated reduction configurations, resulting in a set of reticulation-trimmed subnetworks for each input network,

2. calculate the MACRS-Simple for each such pair of reduced networks, taking the largest from among them.

The first step can be done in $O(3^r n)$ time, producing a number of pairs reaching $O(3^r)$. Each of those pairs can be handled in the second step in $O(n^3)$ time. Much of our investigation into runtime improvements of this algorithm concerns reducing the number of pairs produced in the first step, since these serve as the inputs to the time-consuming dynamic programming in step 2.

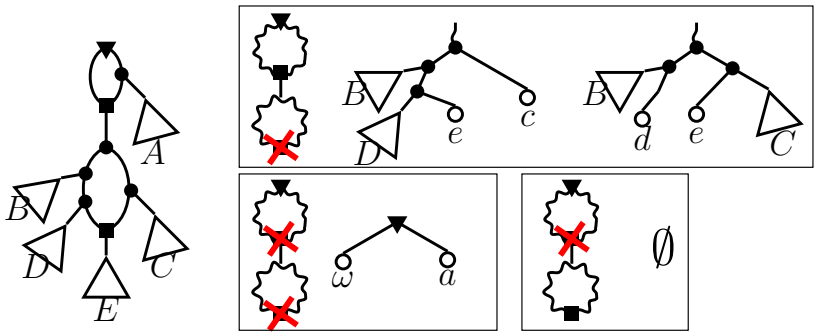

Figure 7.3: Showing a set of reticulation-trimmed subnetworks. On the left, we represent a network with two reticulations, one an ancestor of the other. Call the ancestor reticulation r_1 and the descendant reticulation r_2. Subtrees labelled A, B, C, D, and E are just that, subtrees with no reticulations. Let $c \in C$, $d \in D$, $e \in E$ and $\omega \in B \cup C \cup D \cup E$. In each box we show a simplified network representation to the left where the selection of a reticulation edge vr or ur for reduction is denoted with a red 'X' over r. In the right of each box is the resulting combined set of reticulation-trimmed subnetworks, or just the relevant portion in the top case (the upper non-trivial biconnected component does not change). Notice how on the bottom left example, whether we reduce the "left" or "right" reticulation edge of r_2 is irrelevant: Not every reticulation edge set choice produces a unique reticulation-trimmed subnetwork set. Notice how in the bottom right example, we produce the empty set: the scenario in which only r_1 is slated for reduction cannot produce a reticulation-trimmed subnetwork since r_1 cannot be in any cherry while r_2 is unreduced.

See Figure 7.3 for a simple description of how not every naive choice of reticulated reductions produce a valid or a unique reticulation-trimmed subnetwork, both due to the bottom-up nature of cherry reductions. This fact, along with an observation

regarding a required isomorphism between inputs (described next), is what inspires our optimization: smart enumeration.

7.3.2 Enumeration of reticulation-trimmed subnetworks

Generating reticulation-trimmed subnetwork pairs can be done, for input networks $\mathcal{N}_1$ and $\mathcal{N}_2$, by taking the product of the two respective sets of their every reticulation-trimmed subnetwork. We call this the naive method of data enumeration (or *naive enumeration*). We compare this method experimentally with the following, improved method in Section 7.4.

For any reticulation r, if we decide that it should remain in the final MACRS i.e. that it will not be reduced to construct a reticulation-trimmed subnetwork, it defines a subnetwork (which we call the *forbidden subnetwork*) above it that can never be a part of a cherry, nor can it be rearranged by any reduction. Similarly, we speak of a forbidden subnetwork defined by a set of reticulations ($R \subseteq R(\mathcal{N})$) that are not reduced. We have previously shown [112, Cor. 1] if there is a solution to MACRS-Simple, then the forbidden subnetwork must form part of an isomorphism (an edge-preserving bijection, since there are no labelled vertices on the forbidden subnetwork) between the two input networks. In this way, we can consider the topological relationship between fixed in-place reticulations as a pre-filter to producing pairs for the MACRS-Simple procedure. We achieve this filter by manipulating a tree structure that we originally defined in [112] for the purpose of counting reticulation-trimmed subnetworks.

For some network $\mathcal{N}$, we define a new graph T, a *dependency tree* of $\mathcal{N}$ (or simply a dependency tree), that describes the hierarchy among reticulations of $\mathcal{N}$. Essentially, T is the Hasse diagram on the partially ordered set $R(\mathcal{N}) \cup \rho(\mathcal{N})$ defined by the ancestry relation in $\mathcal{N}$. In other words, $V(T)$ is named by $R(\mathcal{N}) \cup \{\rho(\mathcal{N})\}$, and the edge uv exists if u covers v. We have previously shown that, by this definition, a dependency tree forms a tree for all level-1 orchard networks [112, Prop. 1]. See Appendix 9.1.2. The isomorphism of two dependency trees is edge-preserving as usual, and does **not** preserve labels: labels are network specific, and we seek isomorphisms between dependency trees of different networks. Let T', a subtree of a dependency tree T, be called a *subdependency tree* of T (or simply a subdependency tree when the context is clear) when it contains the root of T. Note that in this way, a subdependency tree of T is a dependency tree of a network reachable by cherry reductions on the network corresponding to T. In particular, we show that a subdependency tree T' of $\mathcal{N}$ leads us to a set of reticulation-trimmed subnetworks on $\mathcal{N}$ that have T' as their dependency tree, and critically, that all reticulation-trimmed subnetworks of $\mathcal{N}$ can be found in this way [112, Obs. 1].

The final step that we take here is connecting the concept of the subdependency tree generating reticulation-trimmed subnetworks and that of the isomorphism between particular forbidden subnetworks of each input. We assert that there is no

MACRS-Simple solution of input networks if there is no isomorphism between the their dependency trees. Observe how the forbidden subnetwork defined by a set of reticulations that remain, R, and a given reticulation-trimmed subnetwork defined strictly by reticulation edges without an endpoint in R (i.e. in which $R(\mathcal{N}) \setminus R$ is reduced) both have the same dependency tree. As we enumerate all reticulation-trimmed subnetwork pairs produced by isomorphic (sub)dependency trees, we are only discarding pairs that do not have a possible forbidden subnetwork isomorphism. Put differently, non-isomorphic (sub)dependency trees cannot produce reticulation-trimmed subnetwork pairs that lead to a valid MACRS-Simple solution. It is easy to see that subdependency trees that have a different number of nodes (reticulations) cannot yield a valid MACRS-Simple. In the same way, subdependency trees with the same number of nodes, but with a different topology will produce different forbidden subnetworks that cannot result in a valid MACRS-Simple.

In practice, we implement this optimization as follows. First, for the two input networks $\mathcal{N}_1$ and $\mathcal{N}_2$, we build their corresponding dependency trees T_1 and T_2. Then, we build each possible isomorphic pair between subdependency trees of T_1 and T_2. Next, for two isomorphic subdependency trees T_1' and T_2', we take the product of the two sets of reticulation-trimmed subnetworks corresponding to them. It is these pairs that are the ultimate inputs to the MACRS-Simple algorithm. We call this strategy of generating paired input of reticulation-trimmed subnetworks *smart enumeration*.

Where we expect the smart enumeration strategy to show its strength is when the dependency trees between two input networks are highly divergent in topology, thus reducing the number of possible isomorphisms between them. Importantly, the possibility of seeing such divergence increases with reticulation number, so we expect to get better efficiency gains when they are most appreciated, when network complexity increases. We show this in Section 7.4. However, there are a few angles that show a diminished benefit of the smart enumeration strategy. An interesting but not particularly consequential aspect is that the vertices of a (sub)dependency tree are not completely independent as their treatment would suggest. Siblings in a (sub)dependency tree could be the result of a network topology in which both endpoints of a reticulation edge are the ancestor of distinct reticulations, so they both must be removed for its reduction (Appendix 9.1.2). We expect this arrangement is uncommon or rare. More significant is the positive association between the number of isomorphisms and the out-degree of dependency tree vertices, which is explored experimentally in Section 7.4.

7.3.3 Dependency tree ranking heuristic

We acknowledge that the runtime hinges on how many pairs from the algorithm step 1 we check for an MACRS-Simple in algorithm step 2. Here we depart from a guarantee of accuracy in order to make exceptional gains in runtime. As we will see, we do not need to sacrifice much accuracy to achieve this. We do so by prioritizing which

reticulation-trimmed subnetwork pairs are the most promising in terms of returning an MACRS-Simple that ends up being best from among them all, and discarding tests on those we deem least promising.

Recall how the smart enumeration scheme pre-processes the dependency trees of the input network by finding all isomorphic subdependency tree pairs, and call this set T. In these terms "promise" is measured simply by how large the subdependency tree is, thus the heuristic is quite simply defined. We rank the pairs generated by the smart enumeration scheme by size, largest first/highest priority. Given a threshold $0 \leq t \leq 1$, where $t = c/|T|$, c is defined as follows. When running the MACRS-Simple one-by-one on pairs of reticulation-trimmed subnetworks generated from subdependency trees (where T is sorted), we consider only the first c valid MACRS-Simple solution networks, taking our solution to best from among them as usual.

7.3.4 Generating and modifying a network

In acknowledgement of the importance to our experiments of the shape of the dependency tree of a network, we wish to control the dependency tree in our network generator. Therefore, we start with the dependency tree as a base in the following way. First, given a specified number of reticulations r, we generate the set of all isomorphism classes of rooted trees with $r + 1$ vertices, representing the possible dependency trees. Sampling equally across the set (except in one experimental case described later), we choose a dependency tree T. Then, we build a minimally-sized base network that contains the appropriate number of non-trivial biconnected components connected in a way that is true to T. We then "grow" a desired number of leaves from this base by randomly choosing edges to be bisected with a pendant leaf.

It is worth noting that when building a base, two sibling vertices in a dependency tree could be the result of many different topological relationships between non-trivial biconnected components in a network. For example, the root of each component could be a sibling, or one component could be descending a non-reticulate vertex in the other. This only gets more complicated with more siblings. For the sake of simplicity, we chose a consistent way to resolve this. We make sibling vertices descendent a common (trivial) parent structure. Using other types of topological relationships between siblings is not expected to alter the speed or accuracy of our approaches in any significant way.

Once the network $\mathcal{N}$ has been constructed, we generate another network $\mathcal{N}'$ from it by applying d cherry operations on it so that we have a cherry distance of d where d is known. We take half d in cherry reductions, then the remainder in *cherry expansions* (which is the inverse operation to the cherry reduction) with both the simple and reticulated version of the operation. We endeavour to maintain the complexity and apply, if possible, as many reticulated cherry expansions as there were reticulated reductions. Level of the network is maintained by ensuring existing biconnected components are not connected on reticulated cherry expansions. It is not always

possible to add a reticulation and preserve the level, so the modified network does not have any guarantee on the total number of reticulations. When we describe experiments on one network in Section 7.4, assume we obtain a pair of inputs by this process.

Also, it is worth noting there is the possible situation in which a reticulated cherry is reduced, then later, before the two edges/leaves are expanded by any other cherries, a reticulated cherry is added back in the same position. These two moves are undetectable, reducing the real distance by two. This may happen in multiple positions, causing the total difference between the requested and real distance to be up to $2r$. While this situation is observed on extremely small networks, we confirmed experimentally that an inaccuracy is exceedingly rare on even small networks, such that it is a non-issue. We avoid this issue entirely for simple cherry expansions by introducing new leaf labels.

7.4 Results and discussion

All experiments are conducted on Apple M1 chip with 8GB memory. The software package is implemented in Rust. Networks are input and output in extended Newick format [122]. Our software is fully implemented and available on GitHub.

7.4.1 Experiments on simulated data

Comparing enumeration schemes

We hypothesized that the topology of the dependency trees will affect the effectiveness of smart enumeration. This is where we begin our experimentation.

We define the *out-degree* of a dependency tree as the maximum out-degree among all its vertices. We define the *height* of a dependency tree as the number of edges on the longest path from the root to a leaf.

We generate the set of all six-vertex tree isomorphism classes (of which there are 20) representing the possible dependency trees of a network $\mathcal{N}$ with $|R(\mathcal{N})| = 5$. In our tests we individually vary both the out-degree and the height, then we show the linear progression between the extrema of these variables, varying both. See Figure 7.4 for the results of this experimentation. Additionally, a figure is available in Appendix 9.1.3 which plots each possible combination of out-degree and height (with 6 vertices), and in which the three experimental tests are highlighted.

Examining Figure 7.4, left, we see how indeed there is a much larger percent of pairs that can be ruled out in the smart enumeration method when the dependency tree is more caterpillar-like while the percent of pairs that can be filtered out dramatically lowers as out-degree increases, we attribute this to the fact that there are many more possible isomorphisms between (unordered) sibling sets than on (ordered)

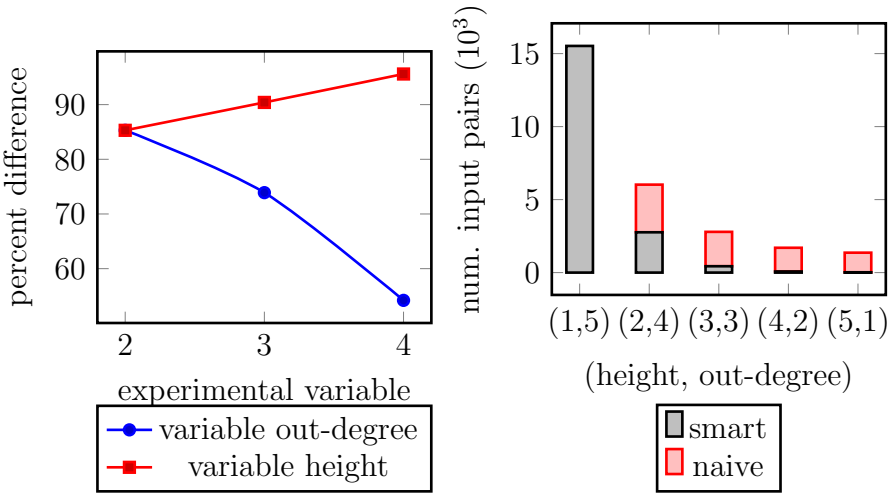

Figure 7.4: Comparing the number of dependency tree pairs requiring testing based on dependency tree topology. Both experimental scenarios are averaged over $n = 100$ replicates of 500 leaf networks with distance of 0. (Left) When out-degree is variable, height is set at 2, and vice versa. The y-axis gives the percent (of the naive enumeration) difference of input pairs that is produced by the smart enumeration method i.e. the percent of pairs that is filtered out by smart enumeration. The higher the number, the better. (Right) The pink area of the bar represents how much is filtered out by smart enumeration (bars are overlapping, not stacked).

ancestral relations. Right, we see the impact in definite terms. For each variable (that is not (1,5)), a not dissimilar absolute number of pairs is filtered out, though what percent of the whole that constitutes varies dramatically. It is obvious to see that the worst case scenario is the *star graph* (height of 1 with the out-degree of 5), which we isolate in the next test.

Next we explore how runtime is affected by other variables of the input networks, number of leaves, of reticulations, and cherry distance. In particular we compare the naive enumeration method with two different scenarios that utilize the smart enumeration scheme, a worst case (a star graph dependency tree) and an average case (any dependency tree). Note that the dependency tree shape may differ between the two input networks after modification by a given cherry distance, especially in the case of larger distances.

See Figure 7.5 for results. Overall the smart enumeration scheme may run on par with the naive enumeration scheme when there are less non-isomorphic pairs to rule out, however we still see an improvement in the average case. Indeed, the overbearing source of complexity is the number of reticulations as we see a dramatic rise in runtime as it increases. The size of the input networks also affects runtime, but in a more linear manner. Interestingly, there is an inverse relationship between the runtime and the cherry distance. We attribute this to the functioning of the dynamic

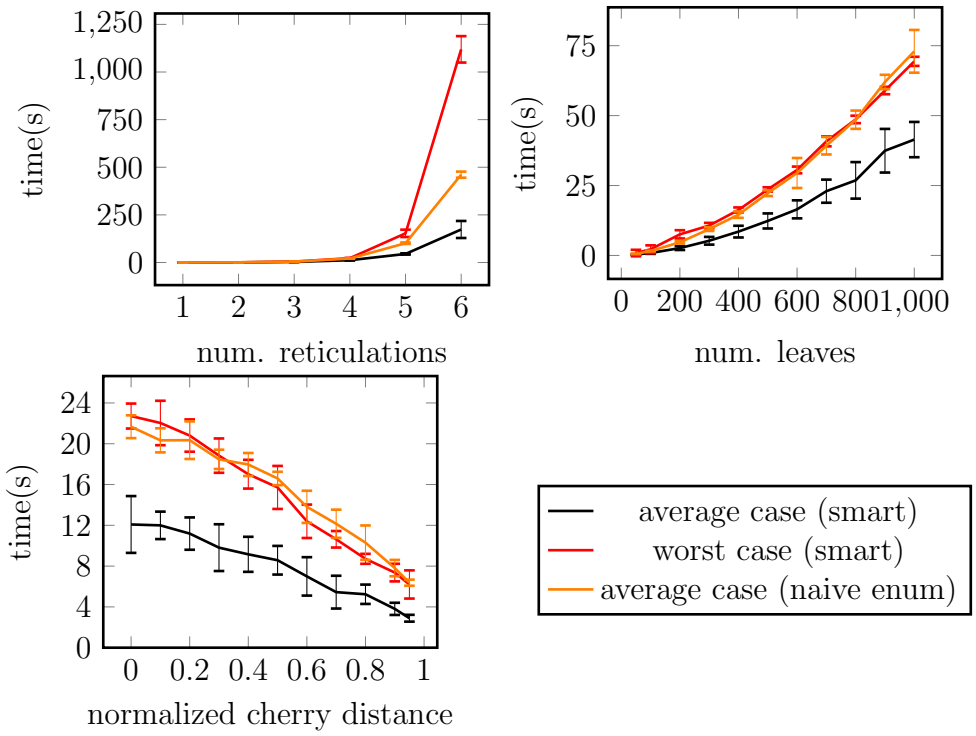

Figure 7.5: Showing the runtime under variable network parameters for both smart and naive enumeration methods. In all cases, results are averaged over $n = 100$ replicates and standard error is shown. Number of reticulations step by 1, number of leaves by 100 and normalized cherry distance by 0.1, with the exception of the maximum normalized cherry distance measure being 0.950 as a cherry distance of 1 is trivial to compute. Unless otherwise stated, the default size of the networks is 500 leaves and with 4 reticulations. The distance is 0.05 when experimenting on the number of reticulations, while it is 0 when the number of leaves is varied so as to ensure the number of reticulations is unchanged.

programming algorithm, which builds the MACRS-Simple between pairs, a more timely task when the MACRS-Simple is larger as in the case of a lower distance. We return to this fact when analyzing the results of the ranking heuristic.

Dependency tree ranking heuristic

We use cherry distance and number of reticulations as the experimental variables. We also show the average case (smart enumeration), as reported in Figure 7.5, for comparison. See Figure 7.6 for results.

Looking at the $c = 1$ scenario, while it is the fastest scenario (left), the accuracy is not sufficient (bottom right). Interestingly, there is a dip in the accuracy before it rises again, mirrored in the inaccuracy of the worst result (dotted) which rises and then falls towards higher accuracy at high cherry distance. This can be explained

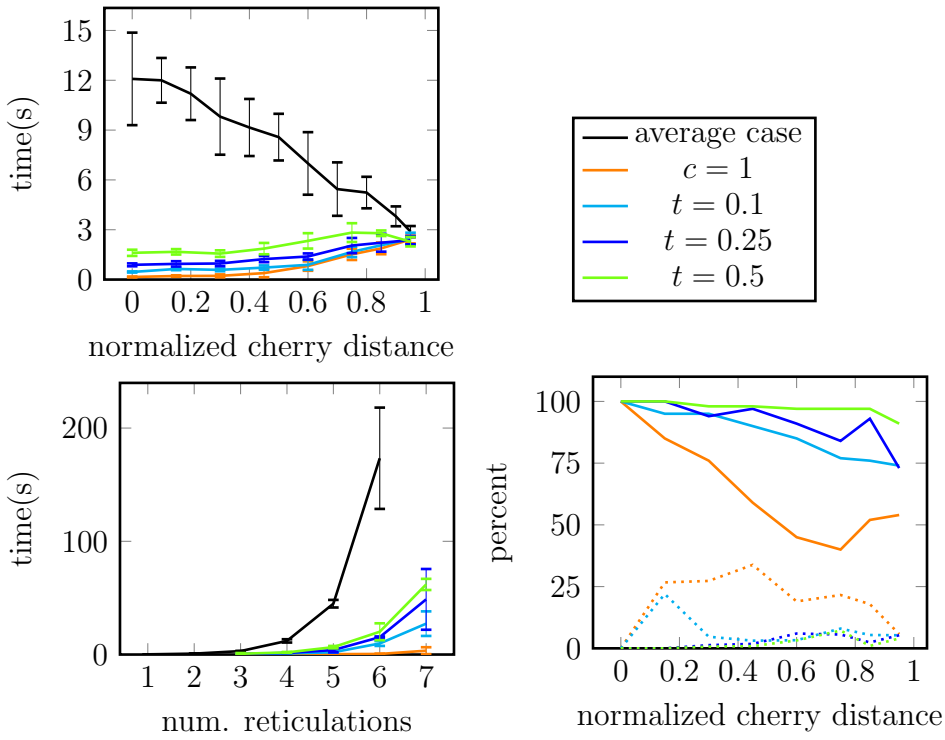

Figure 7.6: Showing runtime and accuracy on ranking heuristic method of calculating inexact cherry distance. (Left Upper and Lower) Showing runtime, in both cases results are averaged over $n = 100$ replicates on input networks with 500 leaves and 4 reticulations, standard error is shown. (Bottom right) Solid lines represent the proportion of the correct results in percent (that is, the number of times the distance is correctly returned in 100 replicates). The dotted lines represents the percent difference of the worst solution returned, that is to say it is the percent difference between the correct distance and the largest returned incorrect distance. In an exact program this is always 0. Recall that a threshold t sets the threshold count c, in the case of $c = 1$ we bypass t and simply take the first valid solution to MACRS-Simple.

by noting that the closer the real distance is to the maximum, the less the heuristic can overshoot the distance. More significant are the results in the $t \geq 0.1$ scenarios, where we see massive increases in accuracy (bottom right) without such dramatic increases in runtime (left). Critically, we see that taking $t = 0.5$ results in nearly perfect accuracy (deviating no more than 10% in both accuracy and in inaccuracy of the worst result) all while maintaining runtime much closer to the $c = 1$ (especially when distance is low) case than to the average case without the ranking heuristic.

Furthermore, we posit that the effectiveness of the ranking heuristic strategy depends on two factors. First, though there are examples to the contrary, we expect the largest MACRS-Simple solution is more likely to have larger (sub)dependency trees. What we see is working in synergy with this factor is that the dynamic programming algorithm in step two runs slower when cherry distance is low, as previously illuminated, because it builds the entire MACRS-Simple. Taken together, we put forward

that the small subdependency tree pairs (of which there are many more possible than large ones) end up having a complete MACRS-Simple being built, which ends up dominating the runtime of the exact method.

7.4.2 Experiments based on real data

Now we show properties of the cherry distance through experiments on real data. Specifically, we obtain six networks on a 27 taxa sample of the apple tribe, Maleae, of the Rosaceae (Rose) family [123]. This sample include genera containing apple, crabapple, pear, quince, hawthorn, and others. The six species networks have each a number of reticulations between 0 and 5, and are inferred from Species Networks applying Quartets (SNaQ) [119, 60] network analysis.

First, we verify the accuracy of the ranking heuristics on this real data. In nearly all cases, across a variety of number of reticulations and of cherry distances tested, the resulting accuracy is 100% with $t = 0.5$. In many cases that deviate from this perfect accuracy, it still remains above 95%. There is one case in which accuracy drops to 87%, however in this case the distance calculated was only one "step" away from the real distance. See Appendix 9.1.4 for tabular results.

Next, we single out rearrangement operation that is an earlier adaptation of the rNNI operation which, together with two complexity changing moves, constitutes the rLST operation [3]. This version is selected as in its definition level is maintained at 1. In short, the rNNI operation selects two edges who exchange a child subnetwork, with specifications as to which edges may be appropriately picked so that level is maintained. An rNNI move is illustrated in Appendix 9.1.5.

In Figure 7.7, left, the experiment performs a single rNNI move, doing so 1000 times for each Maleae network for 6000 total replicates. We measure both the resulting cherry distance, as well as the number of leaves below any vertex affected by the move. Noticing the gap in results on the real data, we also perform the test for simulated data of the same size, 27 leaves and with a random number of reticulations between 0 and 5, we generate 1000 networks in this way, performing the same experiment (Appendix 9.1.6). We confirm the gap results from the specific network topology.

In analyzing the results, first note how, in each test, a single rNNI move results in a fairly uniform distribution of resulting cherry distances measured, which indicates that the cherry distance is valuable in differentiating the changed networks. Then, note how there is a strong correlation between the cherry distance and the proportion of leaves being affected. This is a strength of the cherry distance, it is accentuated by how many taxa have a lineage change by an incongruence. In this way, superficial changes to the network make a weak contribution to cherry distance while changes closer to the root make a stronger contribution.

Let us contrast this with the *soft RF* distance on networks which more or less (aside from rare special cases) corresponds directly to the number of rNNI moves, regardless of where that rearrangement takes place, confirmed through testing whose

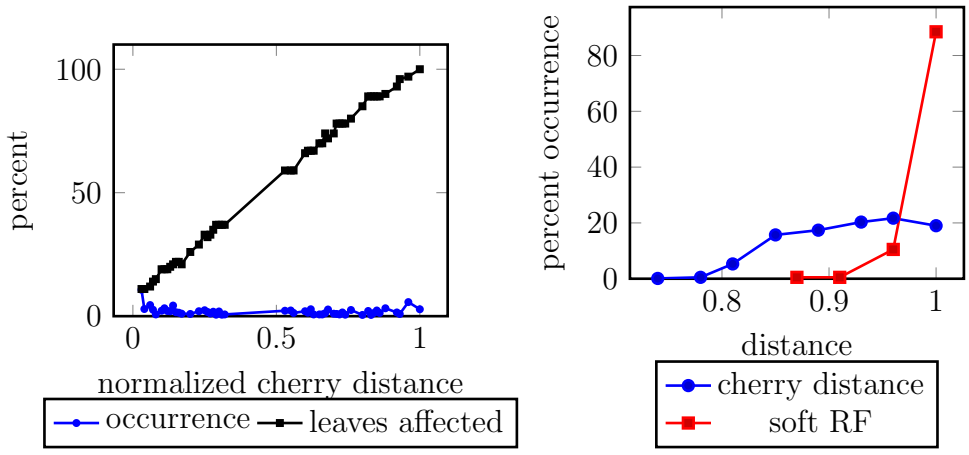

Figure 7.7: (Left) After one rNNI move, we compare the resulting cherry distance and the number of leaves affected by the move, measured across 6000 replicates. Concerning the y-axis, the legend displays what percent means in each case. (Right) This plot shows the percent occurrence that each distance is measured on a pair that consists of a base network from the real, Malus, data set and on a random network of the same leaf label set and number of reticulations. The original networks include a random removal of two leaves to keep a consistent leaf set size of 25. Tests include an equal mix of network pairs on a number of reticulations $r = \{0, 1, 2, 3\}$. Cherry distance is normalized by the diameter while soft RF is normalized by the maximum result, as a formal diameter on level-1 orchards does not yet exist to our knowledge.

results are not shown here. We contend that this fact favours cherry distance as a more accurate reflection of the extent that a change propagates through lineages in the network. We choose the soft RF as comparison since it is (to our knowledge) one of the very few real implementations of a network distance, in the publicly available PhyloNetwork software [82]. Despite being able to handle large numbers of reticulations, PhyloNetwork does not handle large leaf sets well, its runtime being exponential in the number of leaves. Due to this, the experiment using PhyloNetwork (right in Figure 7.7) has a reduced number of replicates. A network is randomly generated on the same leaf set and number of reticulations as a real network from the Malus data set, then the distance (soft RF and cherry) is taken. We see how soft RF does not have the sensitivity of cherry distance. Due to these differences in efficiency and characteristics, our definition and implementation of the cherry distance offer an interesting avenue for advancing the study and practice of network comparison.

7.5 Concluding remarks

7.5.1 Threshold value t

In Section 7.3.3 we present how the threshold value t, provided by the user, is used to limit the number of pairs of MACRS-Simple networks that are considered by way of the formula $t = c/|T|$. The number of MACRS-Simples considered, c, is presented as if it were a portion of T when we know that a subdependency tree pair in T could produce one, some, many, or no reticulation-trimmed subnetwork pairs. Ideally, it would be made more clear to the user the meaning and application of the value they are providing. Also, using the ranking heuristic as currently implemented will always provide a solution, because we do not decrement c unless an MACRS-Simple is found, and we set $c = 1$ at minimum. On the other hand, what value of t is maximum depends on cherry distance, reticulation number, and topology of input. So, when it comes to setting a maximum value to find an exact solution, the current option is to input a sufficiently large value for t (> 1.0). We could additionally implement a message to the user which indicates all possible comparisons have been made. We don't consider this option to be intuitive or satisfying.

There are a few alternatives of how to handle these two complicating matters. We could have the user set c directly. In this way the user understands they are directly setting the number of MACRS-Simples that are considered. We can also relax the condition that we decrement only for an MACRS-Simple, and instead decide to only test a certain number of reticulation-trimmed subnetwork pairs. Here, the user again specifies c with a direct understanding of what they are setting, but we still cannot set a maximum value in an intuitive way. Finally, there is an option that maintains the user's intuition, while making a clearly defined maximum value. In this option, the user sets t, and we still use the formula $t = c/|T|$, but this time c is decremented for each subdependency tree pair. Thus, t truly represents taking a portion of T, and if no solution is found we warn the user to increase the value of t.

If we choose to redefine t in this way, it should be made clear to the user. For example, we could explain this in the repository README, or in a journal version of the paper where we would delineate between the definition used for the experimentation in the paper, and the definition in the available repository. It may also be wise to repeat the experiments with the new definition of t.

Chapter 8

Conclusion

8.1 Summary

What I brought to this program of study in 2019 was a blossoming familiarity with various phylogenetic graph models, with cherry picking in the context of network hybridization, and with proof techniques using set characteristics of trees and networks as it relates to cherry picking. Around the same time, Janssen and Murakami began to disseminate their work on network containment and isomorphism. Inspired by the continued success of cherry picking in the manipulation of phylogenetic networks, and by a persistent gap in the comparison of phylogenetic networks, my advisor, Dr. Olivier Tremblay-Savard and a collaborator Dr. Manuel Lafond, consulted, and a plan for my program of study was crystallized: phylogenetic network distances based on cherry operations.

First we formulated some cherry-based distances, showing equality between some of them (Section 5.3) as well as some other basic properties of them. Then we gave a dynamic programming algorithm to calculate this distance between trees (Section 5.4), and finally we showed that once we introduce a network as input, these distances are NP-hard (Section 5.5). Some of the distances introduced here, which we collectively dub as cherry distance, could be characterized another way, by a shared subnetwork between the two inputs that can be found by cherry picking, the MACRS.

The obvious next step was the formulation of an algorithm to calculate the cherry distance through MACRS. Because of the hardness of the problem, at least on certain levels of networks (the NP-hard proof of Chapter 5 constructs higher-level networks), we had the impetus to design an FPT algorithm with a good parameter. Despite not strictly showing that lower network levels are indeed NP-hard, to facilitate an accessible algorithm design progression that can be generalized to higher level values, we chose to restrict our inputs to cases in which the reticulations are uncomplicated, in the sense that they can be isolated and processed separately. In this way, we presented

an algorithm that finds MACRS on binary, level-1 orchards in Section 6.3. The proposed approach starts by enumerating all reticulation-trimmed subnetworks for both input networks, and then compares all the possible pairs produced for each input network using a dynamic programming algorithm for the MACRS-Simple problem. The enumeration step presented is exponential in the sum of reticulation numbers of both input networks, and the MACRS-Simple algorithm takes cubic time in the maximum number of vertices contained in the input networks. These two distinct subproblems are elaborated on separately in Section 6.4. To finish off this body of work, Section 6.5 provides a framework for bounding the runtime of MACRS by bounding the size of the Reticulation-Trimmed Enumeration set. This is done though an abstraction of the relationship of reticulations in a level-1 network, dependency trees. From the hierarchy of reticulations we can calculate the minimal size of Reticulation-Trimmed Enumeration needed to be processed in the MACRS-Simple step based on the specific input networks' topologies.

For the practical success of any software to calculate cherry distance, it was known from the start that we would need to make optimizations, heuristics, or approximations. Development of speed up strategies were always being considered in the background. As alluded to at the end of Chapter 6, a great starting place was in the isomorphism of dependency trees. Thus, we continued on with this enumeration strategy while going through the process of implementing the code bank. Meanwhile, a heuristic remained elusive. Strategies operating on cherry distance itself was fraught due to the multiplicity of CSs. An approximation was explored that involved representing a network as the trees it displays, and then taking the cherry distance between the trees as a bounds, but it remained unconvincing. Disagreements near the root of a network, with many descendant leaves, increase the distance in a way that is difficult to account for without directly identifying the disagreement. With the software up and running, we noticed a dramatic increase in the runtime of the algorithm when cherry distances were low and we hypothesized there were many unnecessary calculations of MACRS-Simple. The new ranking heuristic strategy was mocked up, and left to run over a holiday, good results streaming in as a gift. It was then we saw our final project to be a worthwhile contribution and set to turn it around to a publication in great time. The initial algorithm design, both improvements, and a technical description of our random network generator are found in Section 7.3. Experimental results and discussion, including a foray into the properties of cherry distance when compared to other network distances, appears in Section 7.4.

What will follow for the remainder of this chapter is an extended discussion of the future work, delineating when it is of particular promise or urgency.

8.2 Limitations

8.2.1 Usability of the software package

A common limiting factor in operation-based network distance is that it is NP-hard to compute, cherry distance included. Even with a well-parameterized algorithm, the software is bound to be exponential if it is exact. Before discussing algorithm design directly, I examine other ways that the existing software package can be made better for practitioners; implementation optimizations and user experience.

Speed is of the utmost importance for a software that works on networks, so it is important to identify optimizations that may eke out precious efficiency gains. This can include profiling the efficiency of libraries and data structures that are used. Though, what I expect to be more significant is the introduction of concurrency to the software. The design of the algorithm itself emphasizes the parallelizability of the recursive calls, there is no dependency between the calls on each vertex's child, or between the subnetworks pendant a biconnected component. What invigorates this prospect even further is Rust's insistence on concurrency support. Threaded computation in this algorithm is therefore expected to be relatively straightforward to implement and be beneficial. As an aside, concurrency itself can be further optimized for example by preferring worst-case data structures over amortized ones since, in an amortized structure, threads left with the harder tasks may cause other threads to wait. Memory efficiency has not been addressed in this work. In fact, in Chapter 6, we opt for a less memory-efficient table-filling method for our MACRS-Simple algorithm, despite that each table entry is only consulted one time, so making a divide-and-conquer approach (without a table) also viable. The dynamic programming method was selected due to the comparative simplicity of its presentation and implementation.

The user experience of the software package should not be taken for granted as a factor in choice. The software should be easy to find and while the source is available on Github, it should also be distributed as a *crate*, the standard packaging environment for Rust. Then, the software must be reliable, in the sense that it has support through language updates. Since version 1.00, Rust has taken stability seriously. Namely, Rust editions remove the issue of syntax changes, since an edition must be specified for every crate. Efficiency hardly matters if the operation of the software is not understandable and clear to the user. A README will provide clear instructions not only how to compile and run the program, but will also include examples of such, including sample inputs that the user can employ to verify correct operation. Previously, the focus was on releasing an operational version of the program in step with the publication, however now a higher priority will be placed on the quality issues outlined here.

8.2.2 Experimentation

The depth and breadth of experimentation was limited by time. This was principally true for the experimentation of Section 7.4.2 that sought to compare cherry distance with other distances to elucidate its advantages and disadvantages. Comparing against soft RF was selected due to the PhyloNetwork program being implemented and available, though the operation of cluster-based distances such as RF and variants is fundamentally different from cherry distance. Because RF necessitates so many cluster comparisons, it is limited by the number of leaves present in the network, rather than the number of reticulations. In fact, the soft RF program, in the publication wherein it was introduced, is tested on networks of up to 36 reticulations, but only as many as 24 leaves.

This difference in and of itself is not the main issue with these tests, what complicated matters the most was the final design of our random network generator (Section 7.3.4). For each reticulation, the first step taken by the generator is to build a minimal biconnected substructure, which is then attached to the overall network. In the building and attaching of these components, vertices are necessarily added to the network, which in turn defines a certain minimum number of leaves on the network. The design did not prioritize minimizing vertices, especially in the attachment step, so this minimal number of leaves quickly climbs towards a number that cannot be sustained by PhyloNetwork. Thus, we settled on testing up to 3 reticulations on 25 leaves. This cluster size was at the limits of PhyloNetwork's capabilities, and so we were constrained in terms of what we could accomplish with these tests in a timely manner. In the future, it would be very interesting to compare a wider range of both reticulate complexity and cluster size of networks that underscore the strengths of each distance. To do so, the random network generator may need to be redesigned.

In evaluating properties of cherry distance, we also performed an experiment that made one rNNI move, then measured the resulting cherry distance. In practice, there are many other rearrangements to choose from to test its effect on the cherry distance. This includes moving one or more leaves, swapping leaves, moving random subtrees/subnetworks, or some version of SPR or TBR. Multiple rearrangements can also be done: In measuring cherry distance after each rearrangement, we would expect it to increase, though not necessarily with exact correlation since some moves would undo previous ones. For our experiment, we implemented the rNNI move ourselves, a time-consuming process which limited our ability to conduct further tests. A deeper analysis of cherry distance will potentially benefit from using existing network manipulation software to expedite this process. It is not clear that such software exists yet, for networks, in as much breadth and depth as it does for trees.

Another interesting potential analysis of cherry distance is the 'bullseye' test as per [77]. This test naturally assumes a good distance can distinguish between a tree/network that is inferred from better data than from worse data. In this test, a tree/network is inferred from some set of data as a reference. The data is then de-

graded in some way, either in size (data is removed) or quality (data is modified), and another tree/network is inferred. This is done successively, with the assumption that a good distance will give a higher distance between the reference and trees/networks inferred with worse data. In this way the 'bullseye' test gauges the performance of a distance under a simulated real use case.

Also, it is well known that most distance measures are maximized between random trees/networks, we show in Section 7.4.2 that we have less such saturation than soft RF, but it would be interesting to go further in testing saturation for different size and complexity of networks. Forms of the RF distance are so pervasive in real practice, it is wise to investigate what might set cherry distance apart. Still in terms of setting cherry distance apart, we can examine the use of cherry operations when it comes to searching the space of networks. Recall that the use case here is as a secondary refinement step to better fit an objective function when constructing a network. In this final refinement step, we want to avoid increasing the complexity of the network (see Section 3.1.2), for which there is potentially the application of simple cherry operations. Because we rely on rearrangements to perform a search like this, we required both cherry expansions and reductions. This is akin to the mixed distance metric described in Chapter 5 (not the cherry distance) modified to the *simple mixed distance*. This could also be done with the mixed distance constrained in another way, wherein a reticulated expansion can only be applied sometime after a reticulated reduction such that reticulations cannot be arbitrarily added in the refinement step. The unique "bottom-up" processing of cherry picking here means the technique would be best when we have a higher confidence of the inner, closer to the root inferences.

8.2.3 The question of real data

We restrict our work to level-1, binary networks mainly for computational reasons, with justify that it is common to network construction techniques. However, a deeper investigation into how common these networks are in real practice is worthwhile. Not only does it have the potential to strengthen the utility of this work, it may also inform future work. If we find that low-level and having binary vertices is common in constructed networks, then the design focus of algorithms can tend towards optimization rather than the complexity of extending to lower parameters or higher complexity networks. In contrast, should real data prove to require increased complexity, it will not be sufficient to stop at the more mathematically simple network classes.

Next, our main optimization is only relevant for the specific situation in which reticulations have an ancestor/descendant relationship. The optimization contributes minimally in the case that reticulations are unordered by ancestry. It is worth asking how often we encounter something like the "worst case" in real data. Is there a "naturally occurring" dependency tree shape or shapes? Answering this question would provide insight on the real efficacy of our optimization.

Thus, a review of evolutionary networks created by practitioners would be a ben-

eficial supplement for the bioinformatician. Not only in facilitating the comparison of networks, but in fulfilling the needs of the construction of networks. In performing such a review we cannot just consider the constraints of the software currently being used to construct networks. That is, it is not enough to evaluate only the capabilities of the currently available tools, rather the emphasis should be placed on the actual needs and use cases of the practitioner.

8.3 Future work

8.3.1 Other parameterizations of cherry distance

The complexity of our algorithm is exponential in the sum of the number of reticulations in both input networks, due to the data enumeration requirement of the design. While this parameterization is a vast improvement in comparison to input size, we can still go further. Other candidate parameters I will discuss here are level, *treewidth*, and *scanwidth*. Let level of network $\mathcal{N}$ be denoted $l(\mathcal{N})$, treewidth $tw(\mathcal{N})$, scanwidth $sw(\mathcal{N})$. Then $r(\mathcal{N}) \geq sw(\mathcal{N}) \geq l(\mathcal{N}) \geq tw(\mathcal{N})$ (though $sw(\mathcal{N}) \leq l(\mathcal{N}) + 1$) [124]. First, I will define and show their relationship before discussing strengths and weaknesses of adapting each of them to cherry distance. We have already defined level as the maximum number of reticulations in any single biconnected component of the network. Extending the algorithm to operate in time parameterized by network level is a non-issue, as we can observe how biconnected components truly have no dependency based on ancestry, as previously assumed. Details are not presented here as they are the subject of a manuscript in preparation, though in the next section I do discuss a simple design modification to operate on level-k networks. Instead, here I focus on treewidth and scanwidth. Note that treewidth is found on an undirected graph, so assume we are doing so on the underlying graph of a directed network.

Treewidth, due to [125, 126, 127, 128], is said to represent how close a graph is to being tree-like [129] for a tree, and only a tree, has a treewidth of one. Treewidth is defined as the least width of any *tree decomposition*, a tree decomposition being a tree that has subsets of vertices of the original graph (a *bag*) mapped to its own in such a way that certain adjacency properties among vertices in the bag are maintained. Subsets of vertices grouped together are then considered together in calculations, giving us algorithms bounded by the size of the largest subset, i.e. the treewidth. Finding an optimal tree decomposition is such a hard problem as to be impractical to compute for general graphs [130]. That said, it is likely easy to get a reasonable one for the simple networks we have been working on here, we can trivially construct a tree decomposition whose width is bounded by level. Though, this might not suit for high level and nonbinary networks.

There are some existing applications of treewidth as it pertains to phylogenetic networks. For example, the binary tree containment problem, whether a tree can be

embedded in a network, is solved with an algorithm parameterized by treewidth [131]. It was also applied to the *small parsimony problem*, an important subtask in some construction algorithms [132]. Notably, the latter work reformulates the tree decomposition in terms more familiar to those working with phylogenetic models, as a set of "agreeing" trees, in an effort to make this parameter more accessible to those working in this domain.

Treewidth is also related to the *graph minors theorem*, a theorem which implies that every family of graphs that is closed under taking minors (a *minor* of a graph can be found by taking only edge contractions or edge/vertex deletions of the graph) can be defined by a finite set of forbidden minors [133], and this applies to *any* graph property. This theorem, also named the Robertson-Seymour theorem, is said to "dwarf any other result in graph theory" and treewidth plays a crucial role in its proof [129]. Relevant here is a new expression of the cherry reduction operation such that, for a network $\mathcal{N}$ and a cherry on it c, $\mathcal{N}\langle c\rangle$ is a graph minor of $\mathcal{N}$. Consider the cherry to be consisting of the two leaves of the cherry and all vertices between them (the parent in the simple case, and the two endpoints of the reticulation edge in the reticulated case), and let the root of the cherry be the highest ancestor vertex in the cherry. Then, a cherry reduction (simple or reticulated) consists of deleting all edges and leaves below the cherry root, noting that the remaining root of the cherry becomes a new leaf, as does the former reticulation in the reticulate case. This new definition will likely become the new standard because of this connection to such a powerful existing theorem. The fact that treewidth is closed under the graph minor relation is central to the proof of the result presented earlier, in Section 3.2.3, characterizing reducing CSs as linear extensions in TCNs [104].

Unfortunately, the undirected nature of treewidth clashes with the directed nature of phylogenetic networks when it comes to formulating algorithms. Scanwidth, in a promising response, is a generalization of cutwidth that integrates the ordering imposed by directionality of network arcs, specifically by respecting the maximality of leaves in the ancestry order. Scanwidth, in particular, can be seen as a form of treewidth that takes the directed nature of networks into account. The introduction of scanwidth was recent, so there are still many open applications for its use. Of note, it is suggested that use of scanwidth may lend itself to more intuitive formulations in dynamic programming when compared to treewidth [124, 132]. So while treewidth is the smaller parameter in a technical sense, scanwidth holds promise for its use as a more intuitive parameter in the case of phylogenetic trees and networks, and may be a more approachable starting point. As we have observed, cherry picking is bottom up, so the application of accounting for edge direction is clear when it comes to cherry picking.

8.3.2 Extensions of cherry distance

A very important extension of cherry distance is to level-k networks, which will be covered in this section after a discussion of extensions to non-binary orchards and non-orchard phylogenetic networks.

We constrained all of our work to binary orchards. Recall in Section 3.2.2 there was a discussion on the other reconstructible classes delineated by Janssen and Murakami, these are classes of orchard networks that can be unambiguously reconstructed from the CS that reduced it. Binary orchards are one of the four reconstructible classes, and so the distance formulation itself extends to the other three classes being in terms of CSs themselves. That being said, it is not hard to imagine a generic form of cherry distance on non-binary networks by allowing cherry-to-cherry variance in which expansion we use. What about algorithm design? We rely heavily on the fact that cherry reductions on binary orchards remove exactly two vertices, thus we can formulate the distance in terms of the size of the MACRS of the two inputs. This is not true for other reconstructible classes as reducing a cherry won't necessarily reduce a multifurcating vertex. However, we are sure that the cherry will reduce either a leaf or a reticulation edge, and this is a reasonable alternative characterization for cherry distance (see Equation 4.1). Where our current algorithm design fails is in handling any non-trivial biconnected component that is not a cycle. Dealing with a more complex biconnected component is the main challenge in an algorithm design for a level-k extension of the algorithm as well. The alternative to working on an MACRS, to calculate cherry distance through CS length directly, is so far impenetrable. If achieved, it would be an easy extension to the other reconstructible classes.

Throughout our work, we required that networks can be reduced completely by cherry reductions, i.e. that they are orchards. Even though the proposed approach applies to orchard networks and not to general networks, the orchard network class actually contains network types that are of interest to the research community, such as tree-child networks [134] and tree-sibling time-consistent networks [84]. The tree-child networks in particular, in addition to having been studied extensively in the literature, are biologically relevant, since all ancestral species (internal vertices) have a path that can go to a leaf using only tree vertices. This reflects the idea that ancestral species have descendants that will perdure through mutation and speciation events, and that hybridization events are not as common as speciation events [2]. Even so, our published work on cherry distance has alluded to a leaf-attaching operation, that transforms general networks into orchards [75], as a possible avenue for the adaptation of cherry distance to non-orchards.

Unfortunately, there are some critical difficulties to adapting this technique and in how leaf adding would be handled. In the original context, the leaf attaching operation is performed on trees, the input set when constructing a network. It was noted at the time that it is hard to decide where to add the auxiliary leaf to the input trees to ensure the final network is an orchard. If we are in the process of

reducing a network and find that there is no cherry present, it is a trivial choice on where to add a leaf to make a cherry. The issue lies in the fact that the choice of cherry to create will have an effect on the resulting distance since we can create the cherry closer or further from a disagreement. Technically speaking, our algorithm design does not perform any actual cherry reductions while enumerating reticulation-trimmed subnetworks, nor in the calculation of MACRS-Simple. So it's possible we don't need to perform any actual leaf additions, but rather just proceed as if we are operating on an orchard. Of course, we still must deal with the impact on the distance itself, there is no ideal solution to these problems.

In fact, we can take inspiration from MACRS-Simple for another simple extension of cherry distance to non-orchard networks. In MACRS-Simple, we live with the fact that there may not be a simple cherry in the network, and when such a point is reached without a solution, we simply return nothing, i.e. there is no solution in that case. We don't require the network be completely reducible under simple cherry reductions, only the portion of the network that disagrees. For MACRS, once we reach a state in which there is no cherry, and we have not reached a solution, we could choose to return no solution in the same way.

Finally, we turn to arguably the highest priority extension, to level-k networks. We restricted our initial algorithm to level-1 for simplicity's sake, such that we can be assured that reticulations are handled separately. There is a rather straightforward adaptation to our algorithm that allows it to handle level-k networks while maintaining basically the same design. Consider the design of the MACRS-Simple algorithm. The key is to notice that we are finding an isomorphism of vertices in the non-trivial biconnected component. Currently, it is done in an explicit way, as there is only one possible isomorphism for each mapping of the two distinct (non-trivial) component paths. Both of the resulting isomorphisms are checked by making recursive calls on each matched exterior (to the component) subnetwork, the maximum is taken. Instead of the explicit mapping, we can precalculate all possible isomorphisms between two biconnected components, taking the maximum in the same way across subcalls to the paired (by the isomorphism) vertices. We can go even further, throughout this course of research, we place an importance on the ancestral ordering of reticulations in a network, that is, we follow from the assumption that changes lower in a network affect those higher up. While this did lead to a fruitful abstraction (dependency trees leading to a heuristic), our future work is investigating how biconnected components, even in an ancestor/descendant relationship, can be handled independently. The development of this algorithm is ongoing, so the details are outside the scope of this work.

Bibliography

[1] M. Willems, E. Lord, L. Laforest, G. Labelle, F.-J. Lapointe, A. M. Di Sciullo, and V. Makarenkov, "Using hybridization networks to retrace the evolution of Indo-European languages," *BMC evolutionary biology*, vol. 16, no. 1, pp. 1–18, 2016.

[2] S. Kong, J. C. Pons, L. Kubatko, and K. Wicke, "Classes of explicit phylogenetic networks and their biological and mathematical significance," *Journal of Mathematical Biology*, vol. 84, no. 6, p. 47, 2022.

[3] K. T. Huber, S. Linz, V. Moulton, and T. Wu, "Spaces of phylogenetic networks from generalized nearest-neighbor interchange operations," *Journal of Mathematical Biology*, vol. 72, no. 3, pp. 699–725, 2016.

[4] L. Nakhleh, D. Ringe, and T. Warnow, "Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages," *Language*, pp. 382–420, 2005.

[5] L. Gabora, S. Lejinen, T. Veloz, and C. Lipo, "A non-phylogenetic conceptual network architecture for organizing classes of material artifacts into cultural lineages," *Proceedings of the Annual Meeting of the Cognitice Science Society*, vol. 33, 2011.

[6] J.-B. d. M. de Lamarck, *Philosophie zoologique, ou Exposition des considérations relatives à l'histoire naturelle des animaux...*, vol. 1. Dentu, 1809.

[7] C. Darwin, *On the Origin of Species by Means of Natural Selection*. Murray, 1859. subtitle: or the Preservation of Favored Races in the Struggle for Life.

[8] E. Haeckel, *Generelle Morphologie der Organismen. Allgemeine Grundzüge der organischen Formen-Wissenschaft, mechanisch begründet durch die von C. Darwin reformirte Descendenz-Theorie, etc*, vol. 2. 1866.

[9] J. B. Reece, L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, R. B. Jackson, *et al.*, *Campbell biology*, vol. 9. Pearson Boston, 2014.

[10] J. Mallet, N. Besansky, and M. W. Hahn, "How reticulated are species?," *BioEssays*, vol. 38, no. 2, pp. 140–149, 2016.

[11] P. H. Sneath, "Cladistic representation of reticulate evolution," *Systematic Zoology*, vol. 24, no. 3, pp. 360–368, 1975.

[12] D. H. Huson and D. Bryant, "Application of Phylogenetic Networks in Evolutionary Studies," *Molecular Biology and Evolution*, vol. 23, pp. 254–267, 10 2005.

[13] R. Froissart, D. Roze, M. Uzest, L. Galibert, S. Blanc, and Y. Michalakis, "Recombination every day: abundant recombination in a virus during a single multi-cellular host infection," *PLoS Biol*, vol. 3, no. 3, p. e89, 2005.

[14] E. Koonin, K. Makarova, and L. Aravind, "Horizontal gene transfer in prokaryotes: quantification and classification," *Annual Reviews in Microbiology*, vol. 55, no. 1, pp. 709–742, 2001.

[15] C. M. Thomas and K. M. Nielsen, "Mechanisms of, and barriers to, horizontal gene transfer between bacteria," *Nature reviews microbiology*, vol. 3, no. 9, pp. 711–721, 2005.

[16] S. M. Soucy, J. Huang, and J. P. Gogarten, "Horizontal gene transfer: building the web of life," *Nature Reviews Genetics*, vol. 16, no. 8, pp. 472–482, 2015.

[17] P. J. Keeling and J. D. Palmer, "Horizontal gene transfer in eukaryotic evolution," *Nature Reviews Genetics*, vol. 9, no. 8, pp. 605–618, 2008.

[18] J. C. D. Hotopp, "Horizontal gene transfer between bacteria and animals," *Trends in Genetics*, vol. 27, no. 4, pp. 157–163, 2011.

[19] A. P. Hertle, B. Haberl, and R. Bock, "Horizontal genome transfer by cell-to-cell travel of whole organelles," *Science Advances*, vol. 7, no. 1, p. eabd8215, 2021.

[20] V. Kubyshkin, C. G. Acevedo-Rocha, and N. Budisa, "On universal coding events in protein biogenesis," *Biosystems*, vol. 164, pp. 16–25, 2018.

[21] N. C. Ellstrand and K. A. Schierenbeck, "Hybridization as a stimulus for the evolution of invasiveness in plants?," *Proceedings of the National Academy of Sciences*, vol. 97, no. 13, pp. 7043–7050, 2000.

[22] S. I. Warwick and C. Stewart, "Crops come from wild plants: how domestication, transgenes, and linkage together shape ferality," *Crop ferality and volunteerism*, vol. 36, no. 1, pp. 9–30, 2005.

[23] B. E. Goulet, F. Roda, and R. Hopkins, "Hybridization in plants: old ideas, new techniques," *Plant physiology*, vol. 173, no. 1, pp. 65–78, 2017.

[24] J. G. Kölreuter, *Vorläufige Nachricht von einigen das Geschlecht der Pflanzen betreffenden Versuchen und Beobachtungen*. Gleditsch, 1761.

[25] E. Warschefsky, R. V. Penmetsa, D. R. Cook, and E. J. Von Wettberg, "Back to the wilds: tapping evolutionary adaptations for resilient crops through systematic hybridization with crop wild relatives," *American journal of botany*, vol. 101, no. 10, pp. 1791–1800, 2014.

[26] R. G. Harrison and E. L. Larson, "Hybridization, introgression, and the nature of species boundaries," *Journal of Heredity*, vol. 105, no. S1, pp. 795–809, 2014.

[27] T. E. Dawling and C. L. Secor, "The role of hybridization and introgression in the diversification of animals," *Annual review of Ecology and Systematics*, vol. 28, no. 1, pp. 593–619, 1997.

[28] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L. Stougie, *et al.*, "Networks: expanding evolutionary thinking," *Trends in Genetics*, vol. 29, no. 8, pp. 439–441, 2013.

[29] D. H. Huson and C. Scornavacca, "A survey of combinatorial methods for phylogenetic networks," *Genome biology and evolution*, vol. 3, pp. 23–35, 2011.

[30] M. Anisimova, D. A. Liberles, H. Philippe, J. Provan, T. Pupko, and A. von Haeseler, "State-of the art methodologies dictate new standards for phylogenetic analysis," 2013.

[31] R. G. Beiko, "Gene sharing and genome evolution: networks in trees and trees in networks," *Biology & Philosophy*, vol. 25, no. 4, pp. 659–673, 2010.

[32] E. Rosenberg and I. Zilber-Rosenberg, "Microbes drive evolution of animals and plants: the hologenome concept," *MBio*, vol. 7, no. 2, pp. e01395–15, 2016.

[33] D. Gevers, R. Knight, J. F. Petrosino, K. Huang, A. L. McGuire, B. W. Birren, K. E. Nelson, O. White, B. A. Methé, and C. Huttenhower, "The human microbiome project: A community resource for the healthy human microbiome," *PLOS Biology*, vol. 10, pp. 1–5, 08 2012.

[34] M. J. Bull and N. T. Plummer, "Part 1: The human gut microbiome in health and disease," *Integrative medicine (Encinitas, Calif.)*, vol. 13, no. 6, pp. 17–22, 2014.

[35] A. Nandi, G. Singh, A. Tiwari, J. Solanki, M. Bedse, and P. Suravajhala, "Metagenomics and neurodegenerative diseases," in *Metagenomics*, pp. 209–223, Elsevier, 2025.

[36] D. P. Dhotre and B. Karmarkar, "Metagenomics-guided reengineering of the gut microbiome," in *Metagenomics*, pp. 225–264, Elsevier, 2025.

[37] V. R. Prabhu, K. Bhavana, P. N. Deo, A. Suresh, R. Kamalakkannan, and M. Nagarajan, "Metagenomics: Implications in oral health and disease," in *Metagenomics*, pp. 265–287, Elsevier, 2025.

[38] B. T. Grenfell, O. G. Pybus, J. R. Gog, J. L. Wood, J. M. Daly, J. A. Mumford, and E. C. Holmes, "Unifying the epidemiological and evolutionary dynamics of pathogens," *science*, vol. 303, no. 5656, pp. 327–332, 2004.

[39] S. J. Salipante and M. S. Horwitz, "Phylogenetic fate mapping," *Proceedings of the National Academy of Sciences*, vol. 103, no. 14, pp. 5448–5453, 2006.

[40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[41] 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*, vol. 1, no. 1, pp. 13–23, 2004.

[42] L. Nakhleh, *Phylogenetic networks*. The University of Texas at Austin, 2004. PhD Thesis.

[43] G. Cardona, F. Rosselló, and G. Valiente, "Comparison of tree-child phylogenetic networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, no. 4, pp. 552–569, 2009.

[44] 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*, vol. 1, no. 1, pp. 13–23, 2004.

[45] C. R. Linder, B. M. Moret, L. Nakhleh, A. Padolina, J. Sun, A. Tholse, R. Timme, and T. Warnow, "An error metric for phylogenetic networks," *University of New Mexico, Tech. Rep. TR03-26*, 2003.

[46] G. Cardona, J. C. Pons, and C. Scornavacca, "Generation of binary tree-child phylogenetic networks," *PLoS Computational Biology*, vol. 15, no. 10, 2019.

[47] A. R. Francis and M. Steel, "Which phylogenetic networks are merely trees with additional arcs?," *Systematic Biology*, vol. 64, no. 5, pp. 768–777, 2015.

[48] L. Zhang, "On tree-based phylogenetic networks," *Journal of Computational Biology*, vol. 23, no. 7, pp. 553–565, 2016.

[49] C. Semple, "Phylogenetic networks with every embedded phylogenetic tree a base tree," *Bulletin of Mathematical Biology*, vol. 78, no. 1, pp. 132–137, 2016.

[50] L. van Iersel, R. Janssen, M. Jones, and Y. Murakami, "Orchard networks are trees with additional horizontal arcs," *Bulletin of Mathematical Biology*, vol. 84, no. 8, p. 76, 2022.

[51] D. Gusfield, S. Eddhu, and C. Langley, "Efficient reconstruction of phylogenetic networks with constrained recombination," in *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pp. 363–374, IEEE, 2003.

[52] D. H. Huson and T. H. Kloepper, "Beyond galled trees-decomposition and computation of galled networks," in *Annual international conference on research in computational molecular biology*, pp. 211–225, Springer, 2007.

[53] 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*, vol. 246, pp. 62–79, 2018.

[54] 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.

[55] C. Semple and J. Simpson, "When is a phylogenetic network simply an amalgamation of two trees?," *Bulletin of mathematical biology*, vol. 80, no. 9, pp. 2338–2348, 2018.

[56] R. Janssen and Y. Murakami, "On cherry-picking and network containment," *Theoretical Computer Science*, vol. 856, pp. 121–150, 2021.

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

[58] H. J. Park, G. Jin, and L. Nakhleh, "Bootstrap-based support of HGT inferred by maximum parsimony," *BMC Evolutionary Biology*, vol. 10, no. 1, pp. 1–11, 2010.

[59] Q. Nguyen and T. Roos, "Likelihood-based inference of phylogenetic networks from sequence data by phyloDAG," in *Algorithms for Computational Biology: Second International Conference, AlCoB 2015, Mexico City, Mexico, August 4-5, 2015, Proceedings 2*, pp. 126–140, Springer, 2015.

[60] C. Solís-Lemus, P. Bastide, and C. Ané, "PhyloNetworks: a package for phylogenetic networks," *Molecular biology and evolution*, vol. 34, no. 12, pp. 3292–3298, 2017.

[61] D. Wen, Y. Yu, J. Zhu, and L. Nakhleh, "Inferring phylogenetic networks using PhyloNet," *Systematic biology*, vol. 67, no. 4, pp. 735–740, 2018.

[62] M. Tan, H. Long, B. Liao, Z. Cao, D. Yuan, G. Tian, J. Zhuang, and J. Yang, "QS-Net: Reconstructing phylogenetic networks based on quartet and sextet," *Frontiers in Genetics*, vol. 10, p. 607, 2019.

[63] C. Allen-Savietta, *Estimating Phylogenetic Networks from Concatenated Sequence Alignments*. The University of Wisconsin-Madison, 2020. Doctoral dissertation.

[64] S. Lutteropp, C. Scornavacca, A. M. Kozlov, B. Morel, and A. Stamatakis, "NetRAX: accurate and fast maximum likelihood phylogenetic network inference," *Bioinformatics*, vol. 38, no. 15, pp. 3725–3733, 2022.

[65] Z. Yang and B. Rannala, "Molecular phylogenetics: principles and practice," *Nature reviews genetics*, vol. 13, no. 5, pp. 303–314, 2012.

[66] M. Baroni, C. Semple, and M. Steel, "A framework for representing reticulate evolution," *Annals of Combinatorics*, vol. 8, no. 4, pp. 391–408, 2005.

[67] M. Bordewich and C. Semple, "Computing the minimum number of hybridization events for a consistent evolutionary history," *Discrete Applied Mathematics*, vol. 155, no. 8, pp. 914–928, 2007.

[68] M. Bordewich and C. Semple, "Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 3, pp. 458–466, 2007.

[69] L. Van Iersel, R. Janssen, M. Jones, Y. Murakami, and N. Zeh, "Polynomial-time algorithms for phylogenetic inference problems involving duplication and reticulation," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 17, no. 1, pp. 14–26, 2019.

[70] K. T. Huber, S. Linz, and V. Moulton, "The rigid hybrid number for two phylogenetic trees," *Journal of Mathematical Biology*, vol. 82, pp. 1–29, 2021.

[71] K. T. Huber, S. Linz, and V. Moulton, "Cherry picking in forests: A new characterization for the unrooted hybrid number of two phylogenetic trees," *arXiv preprint arXiv:2212.08145*, 2022.

[72] G. Bernardini, L. van Iersel, E. Julien, and L. Stougie, "Reconstructing phylogenetic networks via cherry picking and machine learning," in *WABI 2022-2nd International Workshop on Algorithms in Bioinformatics*, 2022.

[73] C. Whidden, R. G. Beiko, and N. Zeh, "Fixed-parameter algorithms for maximum agreement forests," *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1431–1466, 2013.

[74] P. J. Humphries, S. Linz, and C. Semple, "On the complexity of computing the temporal hybridization number for two phylogenies," *Discrete Applied Mathematics*, vol. 161, no. 7-8, pp. 871–880, 2013.

[75] S. Linz and C. Semple, "Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies," *Advances in Applied Mathematics*, vol. 105, pp. 102–129, 2019.

[76] R. Janssen, M. Jones, and Y. Murakami, "Combining networks using cherry picking sequences," in *International Conference on Algorithms for Computational Biology*, pp. 77–92, Springer, Springer, 2020.

[77] M. K. Kuhner and J. Yamato, "Practical performance of tree comparison metrics," *Systematic Biology*, vol. 64, no. 2, pp. 205–214, 2015.

[78] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[79] W. H. Day, "Optimal algorithms for comparing trees with labeled leaves," *Journal of Classification*, vol. 2, pp. 7–28, 1985.

[80] G. Cardona, F. Rosselló, and G. Valiente, "Tripartitions do not always discriminate phylogenetic networks," *Mathematical Biosciences*, vol. 211, no. 2, pp. 356–370, 2008.

[81] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, "Metrics for phylogenetic networks I: Generalizations of the Robinson-Foulds metric," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, no. 1, pp. 46–61, 2009.

[82] B. Lu, L. Zhang, and H. W. Leong, "A program to compute the soft robinson–foulds distance between phylogenetic networks," *BMC Genomics*, vol. 18, no. 2, 2017.

[83] G. Cardona, M. Llabrés, F. Rosselló, and G. Valiente, "A distance metric for a class of tree-sibling phylogenetic networks," *Bioinformatics*, vol. 24, no. 13, pp. 1481–1488, 2008.

[84] P. L. Erdős, C. Semple, and M. Steel, "A class of phylogenetic networks reconstructable from ancestral profiles," *Mathematical Biosciences*, vol. 313, pp. 33–40, 2019.

[85] A. Bai, P. L. Erdős, C. Semple, and M. Steel, "Defining phylogenetic networks using ancestral profiles," *Mathematical Biosciences*, vol. 332, p. 108537, 2021.

[86] Y. Murakami, *On Phylogenetic Encodings and Orchard Networks*. PhD thesis, Delft University of Technology, 2021.

[87] C. Reichling, L. van Iersel, and Y. Murakami, "Metrics for classes of semi-binary phylogenetic networks using μ-representations," *arXiv preprint arXiv:2412.05107*, 2024.

[88] G. Cardona, J. C. Pons, G. Ribas, and T. M. Coronado, "Comparison of orchard networks using their extended μ-representation," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 21, no. 3, pp. 501–507, 2024.

[89] D. F. Robinson, "Comparison of labeled trees with valency three," *Journal of Combinatorial Theory, Series B*, vol. 11, no. 2, pp. 105–119, 1971.

[90] G. W. Moore, M. Goodman, and J. Barnabas, "An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets," *Journal of Theoretical Biology*, vol. 38, no. 3, pp. 423–457, 1973.

[91] D. Penny and M. Hendy, "The use of tree comparison metrics," *Systematic Zoology*, vol. 34, no. 1, pp. 75–82, 1985.

[92] B. L. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of Combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.

[93] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, "On distances between phylogenetic trees," in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pp. 427–436, Society for Industrial and Applied Mathematics, 1997.

[94] A. Francis, K. T. Huber, V. Moulton, and T. Wu, "Bounds for phylogenetic network space metrics," *Journal of Mathematical Biology*, vol. 76, no. 5, pp. 1229–1248, 2018.

[95] M. Bordewich, S. Linz, and C. Semple, "Lost in space? generalising subtree prune and regraft to spaces of phylogenetic networks," *Journal of Theoretical Biology*, vol. 423, pp. 1–12, 2017.

[96] K. T. Huber, V. Moulton, and T. Wu, "Transforming phylogenetic networks: Moving beyond tree space," *Journal of theoretical biology*, vol. 404, pp. 30–39, 2016.

[97] P. Gambette, L. Van Iersel, M. Jones, M. Lafond, F. Pardi, and C. Scornavacca, "Rearrangement moves on rooted phylogenetic networks," *PLoS Computational Biology*, vol. 13, no. 8, 2017.

[98] 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*, vol. 75, no. 10, pp. 1879–1890, 2013.

[99] 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," *Algorithmica*, vol. 84, no. 4, pp. 917–960, 2022.

[100] L. Van Iersel, S. Kelk, N. Lekic, C. Whidden, and N. Zeh, "Hybridization number on three rooted binary trees is ept," *SIAM Journal on Discrete Mathematics*, vol. 30, no. 3, pp. 1607–1631, 2016.

[101] K. Landry, "Reliability of clustering on binary phylogenetic trees," bachelor's thesis, Dalhousie University, 2016.

[102] R. Janssen and Y. Murakami, "Linear time algorithm for tree-child network containment," in *International Conference on Algorithms for Computational Biology*, pp. 93–107, Springer, 2020.

[103] 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*, vol. 129, p. 102222, 2021.

[104] T. M. Coronado, J. C. Pons, and G. Riera, "Counting cherry reduction sequences is counting linear extensions (in phylogenetic tree-child networks)," *arXiv preprint arXiv:2403.14491*, 2024.

[105] 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*, pp. 57–81, Springer International Publishing, 2021.

[106] K. Landry, A. Teodocio, M. Lafond, and O. Tremblay-Savard, "Defining phylogenetic network distances using cherry operations," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 3, pp. 1654–1666, 2022.

[107] M. Steel and T. Warnow, "Kaikoura tree theorems: Computing the maximum agreement subtree," *Information Processing Letters*, vol. 48, no. 2, pp. 77–82, 1993.

[108] C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung, "Computing the maximum agreement of phylogenetic networks," *Theoretical Computer Science*, vol. 335, no. 1, pp. 93–107, 2005.

[109] C. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.

[110] A. Condon, J. Maňuch, and C. Thachuk, "The complexity of string partitioning," *Journal of Discrete Algorithms*, vol. 32, pp. 24–43, 2015.

[111] K. Landry, O. Tremblay-Savard, and M. Lafond, "Finding agreement cherry-reduced subnetworks in level-1 networks," in *RECOMB International Workshop on Comparative Genomics* (K. Jahn and T. Vinař, eds.), pp. 179–195, Springer Nature Switzerland Cham, Springer Nature Switzerland, 2023.

[112] K. Landry, O. Tremblay-Savard, and M. Lafond, "A fixed-parameter tractable algorithm for finding agreement cherry-reduced subnetworks in level-1 orchard networks," *Journal of Computational Biology*, vol. 31, no. 4, pp. 360–379, 2024.

[113] J. E. Hopcroft and R. E. Tarjan, "Dividing a graph into triconnected components," *SIAM Journal on computing*, vol. 2, no. 3, pp. 135–158, 1973.

[114] F. Ruskey, "Listing and counting subtrees of a tree," *SIAM Journal on Computing*, vol. 10, no. 1, pp. 141–150, 1981.

[115] R. Ferreira, R. Grossi, and R. Rizzi, "Output-sensitive listing of bounded-size trees in undirected graphs," in *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*, pp. 275–286, Springer, 2011.

[116] S. Nijssen, "Tree mining," in *Encyclopedia of Machine Learning* (C. Sammut and G. I. Webb, eds.), (Boston, MA), pp. 991–999, Springer US, 2010.

[117] K. Landry and O. Tremblay-Savard, "Fast calculation of cherry distance on level-1 orchard networks: optimization, heuristic and implementation," Presented at the 22nd RECOMB-CG, Seoul, South Korea, 2025.

[118] C. Solis-Lemus, A. Coen, and C. Ane, "On the identifiability of phylogenetic networks under a pseudolikelihood model," *arXiv preprint arXiv:2010.01758*, 2020.

[119] C. Solís-Lemus and C. Ané, "Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting," *PLoS genetics*, vol. 12, no. 3, p. e1005896, 2016.

[120] E. S. Allman, H. Baños, and J. A. Rhodes, "NANUQ: a method for inferring species networks from gene trees under the coalescent model," *Algorithms for Molecular Biology*, vol. 14, pp. 1–25, 2019.

[121] K. Landry, O. Tremblay-Savard, and M. Lafond, "Finding agreement cherry-reduced subnetworks in level-1 networks," *arXiv preprint arXiv:2305.00033*, 2023.

[122] G. Cardona, F. Rosselló, and G. Valiente, "Extended newick: it is time for a standard representation of phylogenetic networks," *BMC bioinformatics*, vol. 9, pp. 1–8, 2008.

[123] B. Liu, C. Ren, M. Kwak, R. G. Hodel, C. Xu, J. He, W. Zhou, C. Huang, H. Ma, G. Qian, D. Hong, and J. Wen, "Phylogenomic conflict analyses in the apple genus malus s.l. reveal widespread hybridization and allopolyploidy driving diversification, with insights into the complex biogeographic history in the northern hemisphere," *Journal of integrative plant biology*, vol. 64, no. 5, pp. 1020–1043, 2022.

[124] V. Berry, C. Scornavacca, and M. Weller, "Scanning phylogenetic networks is np-hard," in *SOFSEM 2020: Theory and Practice of Computer Science: 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20–24, 2020, Proceedings 46*, pp. 519–530, Springer, 2020.

[125] U. Bertele and F. Brioschi, "On non-serial dynamic programming," *J. Comb. Theory, Ser. A*, vol. 14, no. 2, pp. 137–148, 1973.

[126] R. Halin, "S-functions for graphs," *Journal of geometry*, vol. 8, pp. 171–186, 1976.

[127] N. Robertson and P. D. Seymour, "Graph minors. iii. planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, no. 1, pp. 49–64, 1984.

[128] N. Robertson and P. D. Seymour, "Graph minors. ii. algorithmic aspects of tree-width," *Journal of algorithms*, vol. 7, no. 3, pp. 309–322, 1986.

[129] R. Diestel, *Graph theory*. Springer, 5 ed., 2017.

[130] H. L. Bodlaender, "Discovering treewidth," in *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 1–16, Springer, 2005.

[131] L. Van Iersel, M. Jones, and M. Weller, "Embedding phylogenetic trees in networks of low treewidth," *arXiv preprint arXiv:2207.00574*, 2022.

[132] C. Scornavacca and M. Weller, "Treewidth-based algorithms for the small parsimony problem on networks," *Algorithms for Molecular Biology*, vol. 17, no. 1, p. 15, 2022.

[133] N. Robertson and P. D. Seymour, "Graph minors. xx. wagner's conjecture," *Journal of Combinatorial Theory, Series B*, vol. 92, no. 2, pp. 325–357, 2004.

[134] M. Bordewich and C. Semple, "Determining phylogenetic networks from inter-taxa distances," *Journal of mathematical biology*, vol. 73, no. 2, pp. 283–303, 2016.

Chapter 9

Appendix

9.1 Appendices to Project 3, Chapter 7, page 88

9.1.1 Algorithm pseudocode

The following pseudocode is repeated directly from [112]. This is the naive form of enumeration.

Algorithm 5 MACRS Finder

Input Two networks $\mathcal{N}_1$ and $\mathcal{N}_2$
Output A MACRS of $\mathcal{N}_1$ and $\mathcal{N}_2$

1: $\tilde{\mathcal{N}} \leftarrow$ empty network
2: **for each** reticulation-trimmed subnetwork $\mathcal{N}_1'$ of $\mathcal{N}_1$ **do**
3: **for each** reticulation-trimmed subnetwork $\mathcal{N}_2'$ of $\mathcal{N}_2$ **do**
4: Let $\mathcal{N}'$ be a MACRS-Simple of $\mathcal{N}_1'$ and $\mathcal{N}_2'$
5: **if** $\mathcal{N}'$ exists and $|V(\mathcal{N}')| > |V(\tilde{\mathcal{N}})|$ **then** $\tilde{\mathcal{N}} \leftarrow \mathcal{N}'$
6: **end for**
7: **end for**
8: return $\tilde{\mathcal{N}}$

The next pseudocode incorporates all of the improvements discussed in this work. This includes the smart enumeration in step 1 using subdependency trees, as well as the heuristic ranking that filters out less important tests in step 2. Recall that for the ranking threshold, $t = 1$ in the exact algorithm.

Algorithm 6 MACRS Finder with Smart Enumeration and Ranking Heuristic

Input Two networks $\mathcal{N}_1$ and $\mathcal{N}_2$, and a ranking threshold $0 < t \leq 1$
Output A MACRS of $\mathcal{N}_1$ and $\mathcal{N}_2$

let T_1 be the dependency tree of $\mathcal{N}_1$
let T_2 be the dependency tree of $\mathcal{N}_2$
$P = \{(T_1', T_2') | T_1' \simeq T_2', T_1'$ a subdependency tree of T_1, T_2' a subdependency tree of $T_2\}$
$\mathbf{N} \leftarrow \emptyset$
for each $(T_1', T_2') \in P$ **do**
    $\mathcal{N}_1 \leftarrow$ all reticulation-trimmed subnetworks corresponding to T_1'
    $\mathcal{N}_2 \leftarrow$ all reticulation-trimmed subnetworks corresponding to T_2'
    $\mathbf{N} \leftarrow \mathbf{N} \cup (\mathcal{N}_1 \times \mathcal{N}_2)$
end for
order $\mathbf{N}$ by size
$c \leftarrow |T| * t$
$\tilde{\mathcal{N}} \leftarrow$ empty network
while $c > 0$ **do**
    $(\mathcal{N}_1', \mathcal{N}_2') \leftarrow \mathbf{N}$`.pop()` ▷ Pop the largest list element
    let $\mathcal{N}'$ be a MACRS-Simple of $\mathcal{N}_1'$ and $\mathcal{N}_2'$
    if $\mathcal{N}'$ exists **then**
        if $|V(\mathcal{N}')| > |V(\tilde{\mathcal{N}})|$ **then**
            $\tilde{\mathcal{N}} \leftarrow \mathcal{N}'$
        end if
        c--
    end if
end while
return $\tilde{\mathcal{N}}$

9.1.2 Dependency tree

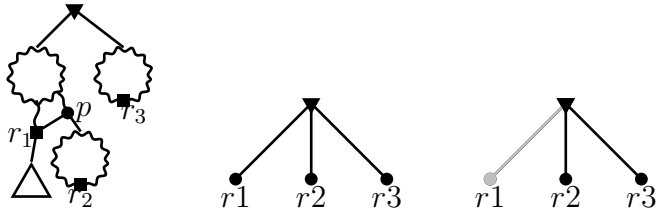

Figure 9.1: (Left) Network with three reticulations r_1, r_2, r_3. Triangles are subtrees (no reticulations). (Centre) Dependency tree of network on left. (Right) Subdependency tree of left, with removed vertices greyed-out. Reticulation r_1 could not be reduced without the cherry reduction of r_2 first, so this subdependency does not produce any reticulation-trimmed subnetwork.

9.1.3 6-vertex dependency trees

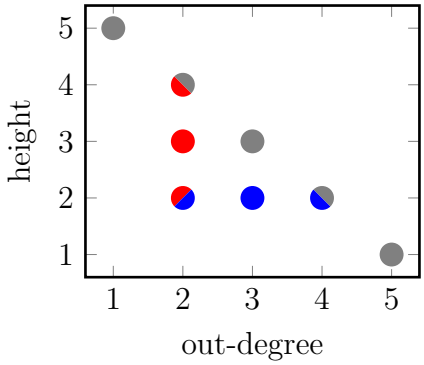

Figure 9.2: Each point represents a cluster of distinct topologies with the same height and out-degree. when the cluster size is > 1, each tree is represented equally in the corresponding results shown in Figure 9.2. The colours correlate to the colours used on the plots, with grey representing the experiment on the right of Figure 9.2. As a matter of interest, the most dense clusters are (2,3) (5 trees) and (2,4) (4 trees) and the clusters with each only 1 tree are (2,2), (1,5), and (5,1).

9.1.4 Ranking heuristic on real data

All tests are on 27-leaf Malus data set, randomly modified by the specified cherry distance. For all tests, $t = 0.5$.

num. reticulations	**0**			
cherry distance	0	15	30	45
time (ms)	0	0	0	0
% accuracy	100	100	100	100
% of worst solution	0	0	0	0
num. reticulations	**1**			
cherry distance	0	15	30	45
time (ms)	2	1	1	1
% accuracy	100	100	100	100
% of worst solution	0	0	0	0
num. reticulations	**2**			
cherry distance	0	15	30	45
time (ms)	6	7	6	5
% accuracy	100	87	99	100
% of worst solution (abs. value)	0	0.13 (2)	0.07 (2)	0
num. reticulations	**3**			
cherry distance	0	15	30	45
time (ms)	31	26	23	20
% accuracy	100	100	100	100
% of worst solution	0	0	0	0
num. reticulations	**4**			
cherry distance	0	15	30	45
time (ms)	100	85	76	65
% accuracy	100	99	100	100
% of worst solution (abs. value)	0	0.13 (2)	0	0
num. reticulations	**5**			
cherry distance	0	15	30	45
time (ms)	166	229	172	110
% accuracy	100	95	94	95
% of worst solution (abs. value)	0	0.13(2)	0.8 (24)	0.2 (10)

9.1.5 rNNI

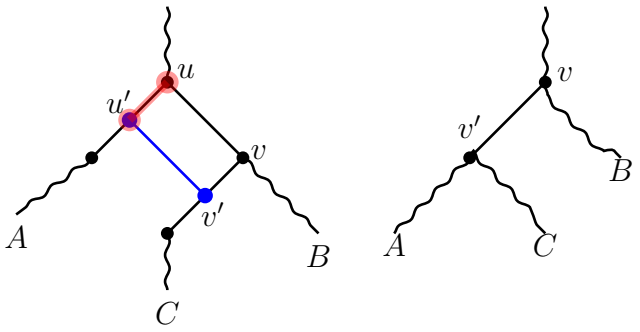

Figure 9.3: An rNNI move on a level-1 network is demonstrated, as specified in [3], where it is shown to always preserve level. (Left) The original network is in black. An edge uv is selected such that it is not in a 3-cycle (a biconnected component made of 3 vertices), the vertices u and v are not in unique nontrivial biconnected components, and that neither u nor v are reticulations. The sibling edge to uv is bisected with new vertex u', and so is one child edge of v with new vertex v'. An edge $v'u'$ is added. The edge uu' is removed, requiring the its endpoints to be with a neighbour to preserve the in order to preserve a proper network definition that precludes a vertex v with $v^- = 1$ and $v^+ = 1$. Additions are represented in blue and removals are highlighted in red. It is not necessarily the case that $A \cap B \cap C = \emptyset$. (Right) The resulting network after the rNNI move is made.

9.1.6 rNNI on simulated networks

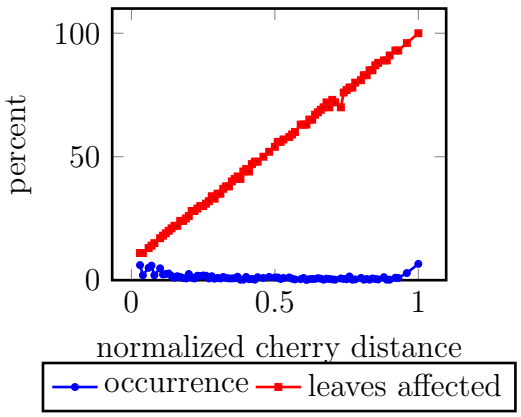

Figure 9.4: Results on a test of $n = 1000$ simulated networks, a pair who differ by 1 rNNI move, each with 27 leaves and an equal number of replicates on each number of reticulations 0-5.