2009

# Algorithms for efficient phylogenetic tree construction

Mukul Subodh Bansal
*Iowa State University*

**Algorithms for efficient phylogenetic tree construction**

by

Mukul Subodh Bansal

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
David Fernández-Baca, Co-major Professor
Oliver Eulenstein, Co-major Professor
Srinivas Aluru
Maria Axenovich
Giora Slutzki

Iowa State University

Ames, Iowa

2009

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisors David Fernández-Baca and Oliver Eulenstein. This thesis would certainly not have been possible without them. It is difficult to overstate my gratitude towards David. Through his excellent guidance, endless patience, and immeasurable support, he not only taught me how to think and how to do research, but also enabled me to grow as a person. I consider myself very fortunate to have not just one great advisor, but two. Oliver had been like a second advisor to me long before he was formally designated as such. I would certainly have been less productive during my graduate studies had it not been for Oliver sharing with me his invaluable ideas and his boundless enthusiasm for research. I am extremely grateful to him for all his help, support, and guidance.

A very special thanks to Srinivas Aluru and Hridesh Rajan for always taking the time to talk to me and giving invaluable advice and guidance, particularly during the last few months. I thank Pavan Aduri, Maria Axenovich, Soma Chaudhuri, Giora Slutzki, and Wallapak Tavanapong along with David Fernández-Baca, Oliver Eulenstein, and Srinivas Aluru for their extraordinary courses. Many thanks to Mike Sanderson and Gordon Burleigh for introducing me to the "Biology" in "Computational Biology", and for so many interesting and insightful discussions.

I am indebted to my lab mates Wen-Chieh Chang, Duhong Chen, André Wehe, Jianrong Dong, Harris Lin, Ruchi Chaudhary, and Akshay Deepak for being such excellent friends, colleagues, and collaborators, and for providing a fun and stimulating environment in which to learn and grow.

Finally, I thank Linda Dutton, Cindy Marquardt, and Maria-Nera Davis for always being so helpful, approachable, and friendly.

# CHAPTER 1. General Introduction

The fact that all life on Earth is genetically related is one of the most profound scientific observations of all time. Scientists have observed that these genealogical relationships among living things can (generally) be represented by a vast evolutionary tree, and constructing this tree of all life is a fundamental scientific problem facing human-kind today. Trees that depict evolutionary relationships between species, or other entities, are called phylogenetic trees or phylogenies.

The rapidly increasing amount of available genomic sequence data provides an abundance of potential information for phylogenetic analyses. Many models and methods have been developed to build evolutionary trees based on this information [27]. A common feature of most of these models is that they start out with fragments of the genome, called genes. Depending on the genes and species, and the methods used to perform the phylogenetic analyses, one typically ends up with a large number of phylogenetic trees which may not agree with one another. Simply put, the problem now is the following: Given several discordant phylogenetic trees as input, infer the (presumably) correct phylogeny. This thesis comprises of three new papers that address some of the methodological and algorithmic challenges posed by this problem. The first and second papers (Chapters 2 and 3) are both related to inferring phylogenetic trees in the presence of gene duplication. The third paper (Chapter 4) discusses a new distance measure for comparing phylogenetic trees.

## 1.1 Gene Duplication

Gene duplication is an evolutionary phenomenon in which one or more genes in an organism are duplicated. Both copies of the duplicated genes then evolve independently. Gene

duplication is known to have played a major role in the evolution of almost all life on Earth. Typically, to build a phylogenetic tree for a set of species, one constructs a phylogenetic tree from genes taken from those species. Such trees are called gene trees. The implicit assumption is that the evolution of the chosen genes mimics the evolution of the species themselves. However, due to complex evolutionary processes such as gene duplication and loss, recombination, and horizontal gene transfer, trees constructed on genes do not always accurately represent the evolutionary history of the corresponding species.

The gene duplication model [29] provides a framework for inferring species phylogenies from a collection of gene trees that are confounded by complex histories of gene duplication events. This model proposes that the true species tree can be inferred by solving one of the following two optimization problems: The gene-duplication problem, and the duplication-loss problem. Unfortunately, both of these problems are known to be NP-hard [33]. Therefore, in practice, local search based heuristic strategies are used to approach the gene-duplication and duplication-loss problems.

Even though both the gene-duplication and duplication-loss approaches have been shown to work well in practice, their utility has been limited due to the high time complexity of the existing heuristics. Chapters 2 and 3 introduce algorithms that greatly speed-up these heuristics.

## 1.2 Comparing Phylogenetic Trees

Most algorithms for constructing phylogenetic trees solve some (typically NP-hard) optimization problem. Solving different optimization problems, even on the same input data, can lead to different phylogenetic trees for the same set of species. In fact, even the same optimization problem, on the same input data, might produce multiple equally optimal phylogenies. Thus, it becomes important to be able to compare and quantify the difference between different phylogenetic trees. In Chapter 4 we introduce a new distance measure for comparing phylogenetic trees on the same leaf set. The distinguishing feature of our distance measure relative to existing distance measures for evolutionary trees is its ability to deal cleanly with

the presence of unresolved nodes, also called polytomies.

## 1.3 Thesis Organization

This thesis consists of five chapters. The next chapter, i.e. chapter 2, describes an algorithm which greatly speeds-up existing heuristics for solving the NP-hard gene-duplication problem. A preliminary version of the paper presented in this chapter appeared in [4] and was also included in my Masters Thesis [5]. Here we present the full and completely re-written version of this paper. In a similar spirit, Chapter 3 describes an algorithm that greatly speeds-up existing heuristics for the NP-hard duplication-loss problem. This work is new and unpublished. In Chapter 4 we introduce a new distance measure for comparing phylogenetic trees on the same leaf set, and give algorithms that can be used to compute this distance efficiently. A preliminary version of this work appeared in [6]. Chapter 4 is an extract from the full and re-written version of this paper. Concluding remarks appear in Chapter 5.

### 1.3.1 Authors' Contributions

*Chapter 2:* MSB contributed to algorithm design, wrote major parts of the manuscript, and contributed to program implementation; OE contributed to algorithm design and to the writing of the manuscript; AW contributed to algorithm design and to program implementation. *Chapter 3:* MSB was responsible for algorithm design, program implementation, and writing major parts of the manuscript; OE contributed to algorithm design and to the writing of the manuscript. *Chapter 4:* MSB was responsible for algorithm design, and wrote major parts of the manuscript; DFB contributed to algorithm design and to the writing of the manuscript.

# CHAPTER 2. SPR-Based Local Searches for the Gene-Duplication Problem

A paper to be submitted to IEEE/ACM Transactions on Computational Biology and Bioinformatics

Mukul S. Bansal, Oliver Eulenstein, and André Wehe

## Abstract

The gene-duplication problem is to infer a species supertree from a collection of gene trees that are confounded by complex histories of gene duplications. This problem is NP-hard and thus requires efficient and effective heuristics. Existing heuristics perform a stepwise search of the tree space, where each step is guided by an exact solution to an instance of a local search problem. These local search problems are often defined based on the classical SPR tree edit operation. We improve on the best-known running time of the SPR based local search problem by a factor of $n$, where $n$ is the number of species in the resulting supertree solution. This makes the gene-duplication problem more tractable for large-scale phylogenetic analyses. We verify the exceptional performance of our solution in a comparison study using sets of large randomly generated gene trees.

## 2.1 Introduction

The rapidly increasing amount of available genomic sequence data provides an abundance of potential information for phylogenetic analyses. Most phylogenetic analyses combine genes from presumably orthologous loci, or loci whose homology is the result of speciation. These

analyses largely neglect the vast amounts of sequence data from gene families, in which complex evolutionary processes such as gene duplication and loss, recombination, and horizontal transfer generate gene trees that differ from species trees. One approach to utilize the data from gene families in phylogenetics is to reconcile their gene trees with species trees based on an optimality criterion, such as the gene-duplication model introduced by Goodman et al. [29]. This problem is a type of *supertree problem*, that is, assembling from a set of input gene trees a species supertree that contains all species found in at least one of the input trees. The decision version of the gene-duplication problem is NP-complete [33]. Other approaches make use of sequence similarity to reconstruct the underlying evolutionary history of genes (see, for example, [51, 52]). Probabilistic models for gene/species tree reconciliation as well as gene sequence evolution have also been developed [2, 3].

Existing heuristics aimed at solving the gene-duplication problem search the space of all possible supertrees guided by a series of exact solutions to instances of a local search problem [38, 35]. The gene-duplication problem has shown much potential for building phylogenetic trees for snakes [45], vertebrates [39, 40], *Drosophila* [20], and plants [43]. Yet, the run time performance of existing heuristics has limited the size of such studies. We improve on the best existing solution for the local search problem asymptotically by a factor of $n$, where $n$ is the number of species from which sequences in the gene trees were sampled (that is the number of nodes in a resulting supertree). To show the applicability of our improved solution for the local search problem, we implemented it as part of standard heuristics for the gene-duplication problem. We demonstrate that the implementation of our method greatly improves the speed of standard heuristics for the gene-duplication problem and makes it possible to infer large supertrees that were previously difficult, if not impossible, to compute.

### 2.1.1 Previous Results

The gene-duplication problem is based on the Gene Duplication model from Goodman et al. [29]. In the following, we (i) describe the Gene Duplication model, (ii) formulate the gene-duplication problem, and (iii) describe a heuristic approach of choice [38, 35] to solve the

gene-duplication problem.

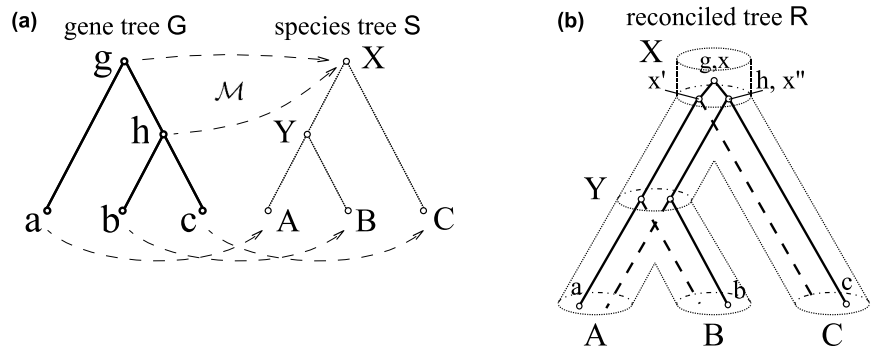


Figure 2.1 (a) Gene trees $G$ and species tree $S$ are comparable, as the leaf-mapping from $G$ to $S$ indicates. $\mathcal{M}$ is the lca-mapping from $G$ to $S$. (b) $R$ is the reconciled tree for $G$ and $S$. In species $X$ of $R$ gene $x$ duplicates into the genes $x'$ and $x''$. The solid lines in $R$ represent the embedding of $G$ into $R$.

#### 2.1.1.1 Gene Duplication model

The Gene Duplication model is well studied [37, 31, 36, 23, 54, 18, 10, 30] and explains incompatibilities between a pair of "comparable" gene and species trees through gene duplications. A gene and a species tree are *comparable*, if a *leaf-mapping* exists that provides a leaf to leaf mapping that maps every leaf node in the gene tree to a leaf node in the species tree. Biologically speaking, the leaves in the gene tree represent genes and the leaves in the species tree represent species, and the leaf-mapping essentially maps each gene to the species from which it was sampled. Consider the example shown in Figure 2.1, taken from [4]. The leaf to leaf mapping from the gene tree $G$ to the species tree $S$ is the leaf-mapping. However, both trees describe incompatible evolutionary histories. The Gene Duplication model explains such incompatibilities by reconciling the gene tree with postulated gene duplications. For example, in Figure 2.1 a reconciled gene tree $R$ can be theoretically inferred from the species tree $S$ by duplicating a gene $x$ in species $X$ into the copies $x'$ and $x''$ and letting both copies speciate according to the topology of $S$. In this case, the gene tree can be embedded into the reconciled tree. Thus, the gene tree can be reconciled by using the duplication of gene $x$ to explain the incompatibility. The minimum number of gene duplications that are necessary under the

Gene Duplication model to explain the incompatibilities can be inferred from the mapping $\mathcal{M}$, which is an extension of the given leaf-mapping. $\mathcal{M}$ maps every gene in the gene tree to the most recent species in the species tree that could have contained the gene. More precisely, $\mathcal{M}$ maps each gene to the least common ancestor of the species from which the leaves (genes) of the subtree rooted at the gene were sampled (given by the leaf-mapping). A gene in the gene tree is a *(gene) duplication* if it has a child with the same $\mathcal{M}$ mapping [37, 23]. The *reconciliation cost* for a gene tree and a comparable species tree is measured in the number of gene duplications in the gene tree induced by the species tree.[1] The *reconciliation cost* for a given collection of gene trees and a species tree is the sum of the reconciliation costs for each gene tree in the collection and the species tree. The mapping function is linear time computable on a PRAM [54] through a reduction from the least common ancestor problem [9]. Hence, the reconciliation cost for a collection of gene trees and a species tree is computable in linear time.

#### 2.1.1.2 Gene-duplication problem and heuristics

The *gene-duplication problem* is to find, for a given collection of gene trees, a comparable species tree with minimum reconciliation cost. This approach has been successfully applied to phylogenetic inference in snakes [45], vertebrates [39, 40], *Drosophila* [20], and plants [43] among others. The decision variant of this problem and some of its characterizations are NP-complete [33, 26] while some parameterizations are fixed parameter tractable [49, 32]. Therefore, in practice, heuristics (e.g. [38, 35]) are commonly used for the gene-duplication problem, even though they are unable to guarantee an optimal solution. These heuristics are based on local search and, consequently, involve repeatedly solving a local search problem. Such a heuristic starts with some species tree comparable with the input gene trees and finds a minimum reconciliation cost tree in its neighborhood. This constitutes one local search step. The new tree thus found then becomes the starting point for the next local search step, and so on, until a local minima is reached. Thus, at each local search step we must solve the local

[1] Alternatively, the reconciliation cost could be defined in the number of gene duplications and losses. However, it is often problematic to accurately infer gene losses if there is missing data. Thus, for this study we only consider gene duplications.

search problem. The time complexity of the local search problem depends on the tree edit operation used to define the neighborhood.

An edit operation of interest is the rooted subtree pruning and regrafting (SPR) operation [1, 11]. Given a tree $S$, an SPR operation can be performed in three steps: (i) prune some subtree $P$ from $S$, (ii) add a root edge to the remaining tree $S$, (iii) regraft $P$ into an edge of the remaining tree $S$. The resulting tree graph is connected and every node has a degree of $\Theta(n^2)$, where $n$ is the size of a species tree comparable to the given gene trees [46]. Assuming, for convenience, similar gene tree and species tree sizes, the local search problem for the SPR edit operation can be solved naively in $\Theta(n^3)$ time per gene tree. If there are $k$ gene trees then this gives a total time bound of $O(kn^3)$. This is the best-known algorithm to solve the local search problem for SPR operations. In practice, the cubic run time typically allows only the computation of smaller supertrees [38, 35]. We show how to solve the local search problem for the SPR edit operations within $O(kn^2)$ time.

**Contribution of the Manuscript:** We introduce an algorithm that, irrespective of the sizes of the gene trees, improves the run time of the best known solution by $\Theta(n)$, where $n$ is the size of the resulting species supertree. To support typical input gene trees, our algorithm also allows multiple leaf-genes from the same gene tree to map to a single leaf-species. We implemented our algorithm as part of a standard heuristic for the gene-duplication problem, and we compared the run times of our implementation and the program GeneTree, which can infer species trees using the same local search heuristic. Our experiments demonstrate the great improvement in runtime offered by our algorithm over current approaches.

## 2.2 Basic Notation and Preliminaries

By and large, we follow the basic definitions and notation from [8]. Given a tree $T$, let $V(T)$ and $E(T)$ denote the node and edge sets of $T$ respectively. $T$ is *rooted* if it has exactly one distinguished node called the *root* which we denote by $rt(T)$. We define $\leq_T$ to be the partial order on $V(T)$ where $x \leq_T y$ if $y$ is a node on the path between $rt(T)$ and $x$. The set of minima under $\leq_T$ is denoted by $rt(T)$ and its elements are called *leaves*. If $\{x, y\} \in E(T)$ and

$x \leq_T y$ then we call $y$ the *parent* of $x$ denoted by $pa_T(x)$ and we call $x$ a *child* of $y$. The set of all children of $y$ is denoted by $Ch_T(y)$. If two nodes in $T$ have the same parent, they are called *siblings*. The *least common ancestor* of a non-empty subset $L \subseteq \mathcal{V}(T)$, denoted as $\text{lca}(L)$, is the unique smallest upper bound of $L$ under $\leq_T$. A *subtree* of $T$ rooted at node $y \in V(T)$, denoted by $T_y$, is the tree induced by $\{x \in V(T) : x \leq_T y\}$. $T$ is *fully binary* if every node has either zero or two children. Throughout this paper, the term tree refers to a rooted fully binary tree.

Next, we introduce the definitions necessary for stating the gene-duplication problem.

### 2.2.1 The Gene-Duplication Problem

A *species tree* is a tree that depicts the evolutionary relationships of a set of species. Given a gene (or gene family) for a set of species, a *gene tree* is a tree that depicts the evolutionary relationships among the sequences encoding only that gene (or gene family) in the given species.[2] Thus, the nodes in a gene tree represent genes from some species. In this work, we shall assume that each leaf of the gene trees is labeled with the species from which that gene was sampled. Thus, unlike species trees, a gene tree might have several leaves with the same label.

**Definition 2.2.1** (Mapping)**.** *The* leaf-mapping $\mathcal{L}_{G,S}\colon Le(G) \to Le(S)$ *maps a leaf node* $g \in Le(G)$ *to that unique leaf node* $s \in Le(S)$ *which has the same label as* $g$*. The extension* $\mathcal{M}_{G,S}\colon V(G) \to V(S)$ *of* $\mathcal{L}_{G,S}$ *is the* mapping *defined by* $\mathcal{M}_{G,S}(g) = \text{lca}(\mathcal{L}_{G,S}(Le(G_g)))$.

Note: For any node $s \in V(S)$, $\mathcal{M}^{-1}_{G,S}(s)$ denotes the set of nodes in $G$ that map to node $s \in V(S)$ under the mapping $\mathcal{M}_{G,S}$.

**Definition 2.2.2** (Comparability)**.** *Given trees* $G$ *and* $S$*, we say that* $G$ *is* comparable *to* $S$ *if, for each* $g \in Le(G)$*, the leaf-mapping* $\mathcal{L}_{G,S}(g)$ *is well defined. A set of gene trees* $\mathcal{G}$ *is* comparable *to* $S$ *if each gene tree in* $\mathcal{G}$ *is comparable to* $S$*.*

Throughout this paper we use the following terminology: $\mathcal{G}$ is a set of gene trees that is comparable to a species tree $S$, and $G \in \mathcal{G}$.

[2] A *gene family* is a set of homologous genes assumed to have shared ancestry.

**Definition 2.2.3** (Duplication)**.** *A node $g \in V(G)$ is a* (gene) duplication *if $\mathcal{M}_{G,S}(g) \in \mathcal{M}_{G,S}(Ch(g))$ and we define $Dup(G,S) = \{g \in V(G) \colon$ $g$ is a duplication$\}$.*

**Definition 2.2.4** (Reconciliation cost)**.** *We define reconciliation costs for gene and species trees as follows:*

1. *$\Delta(G,S) = |\,Dup(G,S)|$ is the* reconciliation cost from $G$ to $S$.
2. *$\Delta(\mathcal{G},S) = \sum_{G\in\mathcal{G}} \Delta(G,S)$ is the* reconciliation cost *from $\mathcal{G}$ to $S$.*
3. *Let $\mathcal{T}$ be the set of species trees to which $\mathcal{G}$ is comparable. We define $\Delta(\mathcal{G}) = \min_{S\in\mathcal{T}} \Delta(\mathcal{G},S)$ to be the* reconciliation cost of $\mathcal{G}$.

The gene-duplication problem is the problem of finding a species tree that requires the minimum number of postulated gene duplications. More formally:

**Problem 1** (Duplication)**.**

Instance: *A set $\mathcal{G}$ of gene trees.*

Find: *A species tree $S^*$ to which $\mathcal{G}$ is comparable, such that $\Delta(\mathcal{G},S^*) = \Delta(\mathcal{G})$.*

### 2.2.2 Local Search Problems

Here, we first define the SPR [11] edit operation and then formulate the related local search problems that were motivated in the Introduction.

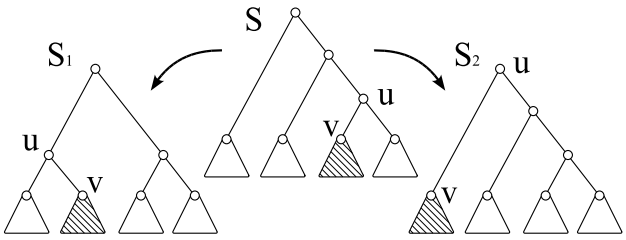


Figure 2.2 $S_1$ and $S_2$ are obtained from $S$ by pruning the subtree rooted at $v$ and regrafting it into the remaining tree $S$.

**Definition 2.2.5** (SPR operation)**.** *(See Fig. 2.2) For technical reasons we first define for a tree $T$ the* planted tree $\Phi(T)$ *that is the tree obtained by adding an additional edge, called* root edge*, $\{p, rt(T)\}$ to $T$.*

*Let $T$ be a tree, $e = (u, v) \in E(T)$ and $X, Y$ be the connected components that are obtained by removing edge $e$ from $T$ such that $v \in X$ and $u \in Y$. We define $\mathsf{SPR}_T(v, y)$ for $y \in Y$ to be the tree that is obtained from $\Phi(T)$ by first removing edge $e$, and then adjoining a new edge $f$ between $v$ and $Y$ as follows:*

1. *Create a new node $y'$ that subdivides the edge $(pa(y), y)$.*
2. *Add edge $f$ between nodes $v$ and $y'$.*
3. *Suppress the node $u$, and rename $y'$ as $u$.*
4. *Contract the root edge.*

*We say that the tree $\mathsf{SPR}_T(v, y)$ is obtained from $T$ by a* subtree prune and regraft (SPR) *operation that* prunes *subtree $T_v$ and* regrafts *it above node $y$.*

**Notation.** We define the following:

1. $\mathsf{SPR}_T(v) = \bigcup_{y \in Y} \{\mathsf{SPR}_T(v, y)\}$
2. $\mathsf{SPR}_T = \bigcup_{(u,v) \in E(T)} \mathsf{SPR}_T(v)$

We now define the relevant local search problems based on the SPR operation.

**Problem 2** (SPR-Scoring (SPR-S))**.**

Instance: *A set $\mathcal{G}$ of gene trees, and a species tree $S$ such that $\bigcup_{G \in \mathcal{G}} \bigcup_{g \in Le(G)} \mathcal{L}_{G,S}(g) = Le(S)$.*

Find: *A tree $T^* \in \mathsf{SPR}_S$ such that $\Delta(\mathcal{G}, T^*) = \min_{T \in \mathsf{SPR}_S} \Delta(\mathcal{G}, T)$.*

**Problem 3** (SPR-Restricted Scoring (SPR-RS))**.**

Instance: *A set $\mathcal{G}$ of gene trees, a species tree $S$ such that $\bigcup_{G \in \mathcal{G}} \bigcup_{g \in Le(G)} \mathcal{L}_{G,S}(g) = Le(S)$, and a non-root node $v$ in $V(S)$.*

Find: *A tree $T^* \in \mathsf{SPR}_S(v)$ such that $\Delta(\mathcal{G}, T^*) = \min_{T \in \mathsf{SPR}_S(v)} \Delta(\mathcal{G}, T)$.*

Throughout the remainder of this manuscript, $S$ denotes a species tree such that $Le(S) = \bigcup_{G \in \mathcal{G}} \bigcup_{g \in Le(G)} \mathcal{L}_{G,S}(g)$, $n$ is the number of leaves in $S$, and $v$ is a non-root node in $V(S)$. The following observation follows from Definition 2.2.5.

**Observation 2.2.1.** *The SPR-S problem on instance $\langle \mathcal{G}, S \rangle$ can be solved by solving the SPR-RS problem $|V(S)| - 1$ times.*

We show how to solve the SPR-RS problem in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time. This immediately implies an $O(\sum_{G \in \mathcal{G}} |V(G)| \cdot n)$ time algorithm for the SPR-S problem (see Observation 2.2.1).

In the next section we study structural properties of the SPR-RS problem and in Section 2.4 we develop our algorithm for the SPR-S problem. Experimental results are discussed in Section 2.5 and concluding remarks appear in Section 2.6.

## 2.3 Solving the SPR-RS problem

Throughout this section, we shall limit our attention to just one gene tree $G \in \mathcal{G}$; in particular, we show how to solve the SPR-RS problem for the instance $\langle \{G\}, S, v \rangle$ in $O(|V(G)| + n)$ time. Our algorithm extends trivially to solve the SPR-RS problem on the instance $\langle \mathcal{G}, T, v \rangle$ in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time.

**Notation.** We define a boolean function $f_T \colon V(G) \to \{0, 1\}$ such that $f_T(g) = 1$ if node $g \in V(G)$ is a duplication w.r.t. tree $T$, and $f_T(g) = 0$ otherwise.

Consider the tree $N^{S,v} = \mathsf{SPR}_S(v, rt(S))$. Observe that, since $\mathsf{SPR}_{N^{S,v}}(v) = \mathsf{SPR}_S(v)$, solving the SPR-RS problem on instance $\langle \{G\}, S, v \rangle$ is equivalent to solving it on the instance $\langle \{G\}, N^{S,v}, v \rangle$. Thus, in the remainder of this section, we will work with tree $N^{S,v}$ instead of tree $S$; the motivation for doing so will become apparent in light of Lemma 2.3.2.

Since $S$ and $v$ are fixed in the current context, we will, in the interest of clarity, abbreviate $N^{S,v}$ simply to $N$.

### 2.3.1 Structural Properties

To solve the SPR-RS problem on instance $\langle \{G\}, N, v \rangle$, we rely on a strong characterization which lets us efficiently infer the value of $f_{S'}(g)$ for any $S' \in \mathsf{SPR}_N(v)$ and any $g \in V(G)$. This characterization is developed in the following six lemmas.

Let $u$ denote the sibling of $v$ in $N$. We color the nodes of $N$ as follows: (i) All nodes in the subtree $N_v$ are colored red, (ii) the root node of $N$ is colored blue, and (iii) all the remaining nodes, i.e. all nodes in $N_u$, are colored green. Correspondingly, we color the nodes of $G$ by assigning to each $g \in V(G)$ the color of the node $\mathcal{M}_{G,N}(g)$.

Then, we have the following lemma.

**Lemma 2.3.1.** *Given $G$ and $N$, if $g \in V(G)$ is either red or green, then $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,N}(g)$ for all $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* If $g$ is red, then so are all its descendants. Now, since the subtree $N_v$ is identical in all trees in $\mathsf{SPR}_N(v)$, $g$ and all its descendants must map to the same nodes under the mappings $\mathcal{M}_{G,N}$ and $\mathcal{M}_{G,S'}$, for any $S' \in \mathsf{SPR}_N(v)$.

Similarly, if $g$ is green, then so are all its descendants. Thus, the mappings from $g$ and its descendants depend only on the green nodes in any $S' \in \mathsf{SPR}_N(v)$ and are immune to the placement of the red nodes. Since the subtree $N_v$ is identical in all trees in $\mathsf{SPR}_N(v)$, except for the addition of the red nodes, $g$ and its descendants must map to the same nodes under the mappings $\mathcal{M}_{G,N}$ and $\mathcal{M}_{G,S'}$ for any $S' \in \mathsf{SPR}_N(v)$. □

**Lemma 2.3.2.** *Given $G$ and $N$, if $g \in V(G)$ is either red or green, then $f_{S'}(g) = f_N(g)$ for all $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* This follows immediately from Lemma 2.3.1. □

Thus, only the blue gene tree nodes, i.e. those gene tree nodes that map to the root of $N$, are responsible for any difference in the reconciliation costs $\Delta(G, N)$ and $\Delta(G, S')$ for any $S' \in \mathsf{SPR}_N(v)$.

Consider the version of $G$ obtained by removing all red nodes from it. The leftover, which we shall call $\Gamma$, must also be a tree. See Figure 2.3 for an example. The significance of $\Gamma$ stems from the following Lemma.

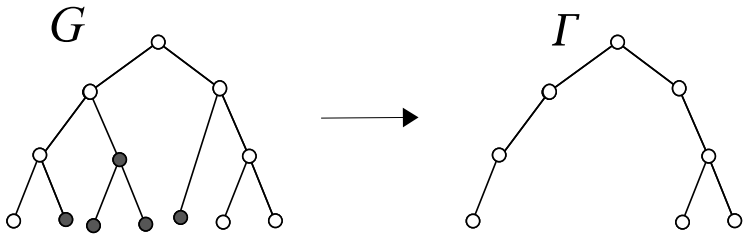


Figure 2.3 The tree $\Gamma$ is obtained from $G$ by removing all the red (shaded) nodes.

**Lemma 2.3.3.** *Given $G$ and $N$, if $g \in V(G)$ is a blue node, then $\mathcal{M}_{G,S'}(g) = \text{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g))$ for any $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* Let $R(g)$ and $G(g)$ denote the set of red and green descendants of $g$ respectively. Note that since $g$ is blue, neither $B(g)$ nor $R(g)$ may be empty, and, moreover, $Le(G_g) = G(g) \cup R(g)$. Thus, for any $S' \in \mathsf{SPR}_N(v)$, we must have $\mathcal{M}_{G,S'}(g) = \text{lca}_{S'}(\mathcal{M}_{G,S'}(G(g)) \cup \mathcal{M}_{G,S'}(R(g)))$.

From the definition of the tree $\Gamma$, it follows that $\text{lca}_{S'}(\mathcal{M}_{S'}(G(g))) = \mathcal{M}_{\Gamma,S'}(g)$, and consequently, by Lemma 2.3.1, $\text{lca}_{S'}(\mathcal{M}_{S'}(G(g))) = \mathcal{M}_{\Gamma,N}(g)$. Thus, $\mathcal{M}_{G,S'}(g) = \text{lca}_{S'}((\mathcal{M}_{\Gamma,N}(g)), \mathcal{M}_{G,S'}(R(g)))$.

Now observe that the subtree $S'_v$ contains precisely all the red nodes in $S'$. Thus, the least common ancestor, in $S'$, of $\mathcal{M}_{\Gamma,N}(g)$ and $\mathcal{M}_{G,S'}(R(g))$, must be simply $\text{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g))$, yielding the lemma. □

Lemma 2.3.3 characterizes the behavior of the mapping from any given blue node in $G$ when the tree $N$ is modified into some other tree $S' \in \mathsf{SPR}_N(v)$. This characterization will be used to prove Lemmas 2.3.4 through 2.3.7.

Note that any blue node $g \in V(G)$ must belong to one of the following four categories: (i) $g$ has two blue children, (ii) $g$ has one blue child and one red child, (iii) $g$ has one blue child and one green child, or (iv) $g$ has one red child and one green child. We analyze each of these four cases separately.

**Lemma 2.3.4.** *If $g \in V(G)$ is blue, and $g$ has two blue children, then $f_{S'}(g) = f_N(g)$ for all $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* We will show that $f_{S'}(g) = 1$ for all $S' \in \mathsf{SPR}_N(v)$.

Let $s$ denote the node $\mathcal{M}_{\Gamma,N}(g)$, $\{s', s''\} = Ch_N(s)$, and $\{g', g''\} = Ch_G(g)$. If either $\mathcal{M}_{\Gamma,N}(g')$ or $\mathcal{M}_{\Gamma,N}(g'')$ is the same as $s$, then it follows from Lemma 2.3.3 that $f_{S'}(g) = 1$ for all $S' \in \mathsf{SPR}_N(v)$. Therefore, let us assume, without any loss of generality, that $\mathcal{M}_{\Gamma,N}(g') \in N_{s'}$ and $\mathcal{M}_{\Gamma,N}(g'') \in N_{s''}$. Let $S' = \mathsf{SPR}_N(v, y)$ for some $y \in V(N_u)$. There are now three possible cases.

$y \in N_{s'}$**:** In this case, by Lemma 2.3.3, we must have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g'') = s$. Therefore, $f_{S'}(g) = 1$.

$y \in N_{s''}$**:** In this case, by Lemma 2.3.3, we must have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = s$. Therefore, $f_{S'}(g) = 1$.

**All other** $y$**:** Here, we must have $\text{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g)) = \text{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g')) = \text{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g''))$. Consequently, by Lemma 2.3.3, $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = \mathcal{M}_{G,S'}(g'')$, and therefore, $f_{S'}(g) = 1$.

Thus, since $\mathsf{SPR}_N(v) = \bigcup_{y \in N_u} \mathsf{SPR}_N(v, y)$, we have $f_{S'}(g) = 1$ for all $S' \in \mathsf{SPR}_N(v)$. □

**Lemma 2.3.5.** *If $g \in V(G)$ is blue, and $g$ has one blue and one red child, then $f_{S'}(g) = f_N(g)$ for all $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* We will show that $f_{S'}(g) = 1$ for all $S' \in \mathsf{SPR}_N(v)$. Let $g'$ and $g''$ denote the red and blue child of $g$ respectively. By definition, since all nodes in the subtree $G_{g'}$ must be red, they can not appear in the tree $\Gamma$. Thus, in $\Gamma$, the node $g$ has only one child, $g''$. This implies that $\mathcal{M}_{\Gamma,N}(g) = \mathcal{M}_{\Gamma,N}(g'')$. Consequently, by Lemma 2.3.3, we must have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g'')$, i.e. $f_{S'}(g) = 1$, for all $S' \in \mathsf{SPR}_N(v)$. □

**Lemma 2.3.6.** *Let $g \in V(G)$ be a blue node and let $s$ denote the node $\mathcal{M}_{\Gamma,N}(g)$. Let $\{s', s''\} = Ch_N(s)$, and $S'$ be a tree in $\mathsf{SPR}_N(v)$. If $g$ has one blue and one green child, denoted $g'$ and $g''$ respectively, then $f_{S'}(g) \neq f_N(g)$ if and only if $\mathcal{M}_{\Gamma,N}(g') \in V(N_{s'})$, $\mathcal{M}_{\Gamma,N}(g'') \in V(N_{s''})$, and $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_{s'})$.*

*Proof.* We will show that $f_{S'}(g) = 0$ if and only if $\mathcal{M}_{\Gamma,N}(g') \in V(N_{s'})$, $\mathcal{M}_{\Gamma,N}(g'') \in V(N_{s''})$, and $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_{s'})$.

Suppose $\mathcal{M}_{\Gamma,N}(g') = s$. Then, by Lemma 2.3.3, we must have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g')$, and, consequently, $f_{S'}(g) = 1$, for any $S' \in \mathsf{SPR}_N(v)$.

Similarly, suppose $\mathcal{M}_{\Gamma,N}(g'') = s$. We have two possible cases: (i) $y \in V(N_s) \setminus \{s\}$, or (ii) $y \notin V(N_s \setminus \{s\})$. In case (i), Lemma 2.3.3 implies that $\mathcal{M}_{G,S'}(g) = s$, i.e. $\mathcal{M}_{G,S'}(g) =$

$\mathcal{M}_{G,S'}(g')$. Consequently, $f_{S'}(g) = 1$ in this case. In case (ii), Lemma 2.3.3 implies that $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g')$ and hence, $f_{S'}(g) = 1$.

Thus, if $f_{S'}(g) = 0$ for some $S' = \mathsf{SPR}_N(v, y)$, then there must exist children $s', s''$ of $s$ such that $\mathcal{M}_{\Gamma,N}(g') \in N_{s'}$ and $\mathcal{M}_{\Gamma,N}(g'') \in N_{s''}$. There are now three possibilities for $y$:

$y \in N_{s'}$**:** In this case, by Lemma 2.3.3, we have $\mathcal{M}_{G,S'}(g) = s$. Now since $\mathcal{M}_{\Gamma,N}(g'') \neq s$, by Lemma 2.3.1, we know that $\mathcal{M}_{\Gamma,S'}(g'') \neq s$. And, since $\mathcal{M}_{\Gamma,N}(g') \in N_{s'}$, we know, by Lemma 2.3.3, that $\mathcal{M}_{\Gamma,S'}(g') \neq s$ in this case. Thus, $f_{S'}(g) = 0$.

$y \in N_{s''}$**:** In this case, by Lemma 2.3.3, we have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = s$. Thus, $f_{S'}(g) = 1$.

**All other** $y \in V(N_u)$**:** In this case, Lemma 2.3.3 implies that we must have $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g')$. Thus, $f_{S'}(g) = 1$.

The lemma follows. □

**Lemma 2.3.7.** *Let $g \in V(G)$ be a blue node and let $s$ denote the node $\mathcal{M}_{\Gamma,N}(g)$. Let $S'$ be a tree in $\mathsf{SPR}_N(v)$. If $g$ has one red and one green child, then $f_{S'}(g) \neq f_N(g)$ if and only if $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$.*

*Proof.* Note that $f_N(g) = 0$. We will show that $f_{S'}(g) = 1$ if and only if $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$.

Let $g'$ and $g''$ denote the red and green child of $g$ respectively. By definition, since all nodes in the subtree $G_{g'}$ must be red, they can not appear in the tree $\Gamma$. Thus, in $\Gamma$, the node $g$ has only one child, $g''$. This implies that $\mathcal{M}_{\Gamma,N}(g) = \mathcal{M}_{\Gamma,N}(g'') = s$.

Observe that $\mathcal{M}_{G,N}(g'') = \mathcal{M}_{\Gamma,N}(g'')$, and consequently, by Lemma 2.3.1, we must have $\mathcal{M}_{G,S'}(g'') = s$ for all $S' \in \mathsf{SPR}_N(v)$. Now, Lemma 2.1 implies that $\mathcal{M}_{G,S'}(g) = s$ if and only if $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$. Thus, it follows immediately that $f_{S'}(g) = 1$ if and only if $S' = \mathsf{SPR}_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$. □

## 2.4 The Algorithm

For any $s \in V(N_u)$, let $A(s)$ denote the cardinality of the set $\{g \in V(G)\colon f_{S'}(g) = 0, \text{but } f_N(g) = 1\}$, and $B(x)$ the cardinality of the set $\{g \in V(G)\colon f_{S'}(g) = 1, \text{but } f_N(g) = 0\}$, where $S' = \mathsf{SPR}_N(v, x)$. Observe that $\mathsf{SPR}_N(v) = \bigcup_{y \in N_u} \mathsf{SPR}_N(v, y)$, and therefore, to solve the SPR-RS problem we must find a node $s \in V(N_u)$ for which $|A(s)| - |B(s)|$ is maximized. Our algorithm computes, at each node $s \in V(N_u)$, the values $A(s)$ and $B(s)$.

In a preprocessing step, our algorithm converts the given tree $S$ into tree $N$, computes the mapping $\mathcal{M}_{G,N}$, colors the nodes in $G$, obtains the tree $\Gamma$, and computes the mapping $\mathcal{M}_{\Gamma,N}$. It also creates and initializes (to 0) two counters $\alpha(s)$ and $\beta(s)$ at each node $s \in V(N_u)$. This takes $O(n)$ time.

The algorithm then considers each node $g \in V(G)$. There are four possible cases:

1. If $g$ is either green or red, then, by Lemma 2.3.2, we must have $f_{S'}(g) = f_N(g)$ for all $S' \in \mathsf{SPR}_N(v)$. Consequently, we do nothing in this case.

2. If $g$ blue and satisfies the precondition of either Lemma 2.3.4 or Lemma 2.3.5, then we must have $f_{S'}(g) = f_N(g)$ for all $S' \in \mathsf{SPR}_N(v)$. Consequently, we do nothing in this case.

3. If $g$ satisfies the precondition of Lemma 2.3.6, then we increment the value of $\alpha(y)$ at each node $y \in V(N_{s'})$ (where $s'$ is as in the statement of Lemma 2.3.6). To do this efficiently we can simply increment a counter at node $s'$ such that, after all $g \in V(G)$ have been considered, a single post-order traversal of $N_u$ can be used to compute the correct values of $\alpha(y)$ at each $y \in V(N_u)$. See Algorithm 1 for a more detailed description.

4. If $g$ satisfies the precondition of Lemma 2.3.7, then we increment the value of $\beta(y)$ at each node $y \in V(N_s) \setminus \{s\}$ (where $s$ is as in the statement of Lemma 2.3.7). Again, to do this efficiently, we can simply increment a counter at node $s$ such that, after all $g \in V(G)$ have been considered, a single post-order traversal of $N_u$ can be used to compute the correct values of $\beta(y)$ at each $y \in V(N_u)$. See Algorithm 1 for a more detailed description.

When the algorithm terminates, the values $\alpha(x)$ and $\beta(x)$ at each node $x \in V(T_v)$ must be the values $A(x)$ and $B(x)$.

A formal description of our algorithm for the SPR-RS problem appears in Procedure-SPR-RS (see Algorithm 1).

**Algorithm 1** PROCEDURE-SPR-RS

**Input:** $\mathcal{G}$, $S$, $v$
Construct the tree $N$ from $S$.
Create and initialize to zero two counters $\alpha(t)$, and $\beta(t)$ at each node $t$ in $N_u$.
**for all** each $G \in \mathcal{G}$ **do**
    Construct the mapping $\mathcal{M}_{G,N}$, color the nodes of $G$ as described in Section 2.3.1, and construct the mapping $\mathcal{M}_{\Gamma,N}$.
**for all** each blue node $g \in \bigcup_{G\in\mathcal{G}} V(G)$ **do**
    Let $s$ denote the node $\mathcal{M}_{\Gamma,N}(g)$, and let $\{s', s''\} = Ch_N(s)$.
    **if** $g$ has a red child and a green child **then**
        Increment $\beta(s')$ and $\beta(s'')$ by one each.
    **if** $g$ has children $g'$ and $g''$ such that $g'$ is blue and $g''$ is green, and $\mathcal{M}_{\Gamma,N}(g') \in V(N_{s'})$ and $\mathcal{M}_{\Gamma,N}(g'') \in V(N_{s''})$ **then**
        Increment $\alpha(s')$ by one.
**for** each node $t$ in a preorder traversal of $N_u$ **do**
    **if** $t \neq u$ **then**
        $\alpha(t) = \alpha(pa(t)) + \alpha(t)$, and $\beta(t) = \beta(pa(t)) + \beta(t)$
A tree with lowest reconciliation cost in $\mathsf{SPR}_S(v)$ is given by $\mathsf{SPR}_S(v,t)$, where $t \in V(N_u)$ is a node that maximizes $\alpha(t) - \beta(t)$. The reconciliation cost of this tree is given by $\Delta(\mathcal{G}, N) - (\alpha(t) - \beta(t))$.

**Theorem 2.4.1.** *The SPR-RS problem on instance $\langle \mathcal{G}, S, v\rangle$ can be solved in $O(\sum_{G\in\mathcal{G}} |V(G)|)$ time.*

*Proof.* We will show that Procedure-SPR-RS solves the SPR-RS problem on instance $\langle \mathcal{G}, S, v\rangle$ in $O(\sum_{G\in\mathcal{G}} |V(G)|)$ time.

*Correctness:* Since $\mathsf{SPR}_N(v) = \bigcup_{y\in N_u} \mathsf{SPR}_N(v, y)$, it is sufficient to show that the values $A(t)$ and $B(t)$ are computed correctly at each node $t \in V(N_u)$, where $A(t) = |\{g \in \bigcup_{G\in\mathcal{G}} V(G) \colon f_{S'}(g) = 0, \text{but } f_N(g) = 1\}|$ and $B(t) = |\{g \in \bigcup_{G\in\mathcal{G}} V(G) \colon f_{S'}(g) = 1, \text{but } f_N(g) = 0\}|$. Since the values $A(t)$ and $B(t)$, for each $t \in V(N_u)$, are computed according to Lemmas 2.3.4 through 2.3.7, the correctness of Procedure-SPR-RS follows.

*Complexity:* Let us analyze Procedure-SPR-RS step-by-step. Steps 2 and 3 take $O(n)$ time.

The 'for' loop of Step 4 can be executed in $O(\sum_{G\in\mathcal{G}}|V(G)|+n)$ time as follows: During a preprocessing step, with-in $O(n)$ time, we can process the tree $N$ so that lca queries on any two nodes in $V(N)$ can be answered in $O(1)$ time; see [9] for details on how to do this. Subsequently, the task of constructing the mapping $\mathcal{M}_{G,N}$ only takes $O(|V(G)|)$ time. Coloring the nodes of $G$ and constructing the mapping $\mathcal{M}_{\Gamma,N}$ also take $O(|V(G)|)$ time. Thus, the total time complexity of this 'for' loop is $O(\sum_{G\in\mathcal{G}}|V(G)|+n)$, which is, since $\sum_{G\in\mathcal{G}}|V(G)|\geq n$, simply $O(\sum_{G\in\mathcal{G}}|V(G)|)$.

The 'for' loop of Step 6 involves considering $O(\sum_{G\in\mathcal{G}}|V(G)|)$ gene tree nodes and performing some processing at each node. We claim that the algorithm can be executed so as to spend only $O(1)$ time at each node. Observe that the 'if' block of Step 8 can be trivially executed in $O(1)$ time. However, to claim an $O(1)$ time complexity for the 'if' block of Step 10, we must show how to check the conditions $\mathcal{M}_{\Gamma,N}(g')\in V(N_{s'})$ and $\mathcal{M}_{\Gamma,N}(g'')\in V(N_{s''})$ in $O(1)$ time. This is done as follows: In a preprocessing step, with-in $O(n)$ time, we can perform an in-order traversal of the tree $N$ and label the nodes with increasing integer values in the order in which they are traversed. Based on the resulting order we can check whether a given node is in $V(N_t)$ for any $t\in V(N)$ in $O(1)$ time. Thus, the 'if' block of Step 10 can be executed in $O(1)$ time as well, yielding a time complexity of $O(\sum_{G\in\mathcal{G}}|V(G)|)$ for the 'for' loop of Step 6.

And lastly, the 'for' loop of Step 12 involves traversing through the nodes in the subtree $N_u$ and spending $O(1)$ time at each node. Therefore, this 'for' loop requires $O(n)$ time.

The total time complexity of Procedure-SPR-RS is thus $O(\sum_{G\in\mathcal{G}}|V(G)|)$. □

**Theorem 2.4.2.** *The* SPR*-S problem on instance* $\langle\mathcal{G},S\rangle$ *can be solved in* $O(\sum_{G\in\mathcal{G}}|V(G)|\cdot n)$ *time.*

*Proof.* This follows immediately from Theorem 2.4.1 and Observation 2.2.1 □

The best known (naïve) approach to solve the SPR-S problem involves computing the reconciliation cost for each of the $\Theta(n^2)$ trees in the SPR neighborhood of $S$ separately. This requires $\Theta(\sum_{G\in\mathcal{G}}|V(G)|\cdot n^2)$ time. Our algorithm thus improves on the best known solution for the SPR-S problem by a factor of $n$.

## 2.5 Experimental Analysis

In order to study the performance of our algorithm we implemented it as part of a standard local search heuristic for the gene-duplication problem; our program is called DupTree. We compared the run time performance of DupTree against the program GeneTree [38].[3] We measured the run time of each program to compute its final species supertree for the same set of input gene trees and the same randomly generated starting species tree. The input gene trees for each run consisted of a set of 20 randomly generated gene trees, all with the same set of taxa. We conducted six such runs, each with a different number of taxa (50, 100, 200, 400, 1000, and 2000) in the input trees. All analyses were performed on a 3 Ghz Intel Pentium 4 CPU based PC with Windows XP operating system. The results of these experiments are shown in Table 2.1. DupTree shows a vast improvement in run time and scalability compared to GeneTree. Consequently, DupTree can compute much larger supertrees within a reasonable time. This also allows our algorithm to be used with more thorough versions of the heuristic to obtain supertrees with lower reconciliation costs. We could not run GeneTree on input trees with more than 200 taxa. Also, the memory consumption of DupTree was less than the memory consumption of GeneTree.

Note that even though both DupTree and GeneTree implement the same local search heuristic, they may produce different supertrees which may also have different reconciliation costs. This happens because during a local search step, more than one neighboring node may have the smallest reconciliation cost. In this case the node to follow is chosen arbitrarily among such nodes, and this may cause the programs to follow different paths in the search space. In practice we noticed little or no difference in the final reconciliation costs. In fact, during the experiments, DupTree inferred supertrees with smaller reconciliation cost more often than GeneTree.

[3]The programs Mesquite [35] and GeneTree [38] both implement similar brute-force algorithms for SPR local search under the gene duplication model.

Table 2.1 GeneTree vs. DupTree

| Taxa size | GeneTree | DupTree |
|---|---|---|
| 50 | 9m:23s | 1s |
| 100 | 3h:25m | 6s |
| 200 | 108h:33m | 58s |
| 400 | – | 9m:19s |
| 1000 | – | 3h:20m |
| 2000 | – | 38h:25m |

## 2.6 Conclusion

Despite the inherent complexity of the gene-duplication problem, it has been an effective approach for incorporating data from gene families into a phylogenetic inference [45, 39, 40, 20]. Yet, existing local search heuristics for the problem are slow and thus cannot utilize the vast quantities of newly available genomic sequence data. We introduced an algorithm that speeds up the stepwise search procedure of local search heuristics for the gene-duplication problem. Our algorithm eliminates redundant calculations in computing the reconciliation cost for all trees resulting from pruning a given subtree and regrafting it to all possible positions.

Since the publication of a preliminary version of this manuscript in [4], efficient algorithms have also been presented for TBR [8] and NNI [7] based local searches for the gene-duplication problem. In particular, the solution for TBR based local searches depends crucially on our efficient algorithm for the SPR local search problem. We have also implemented our SPR algorithm, as well as various enhancements, in the software package DupTree [53].

# CHAPTER 3. Algorithms for Gene Tree Parsimony under Duplication-Loss

A paper to be submitted to IEEE/ACM Transactions on Computational Biology and Bioinformatics

Mukul S. Bansal and Oliver Eulenstein

## Abstract

The duplication-loss problem is to infer a species supertree from a collection of gene trees that are confounded by complex histories of gene duplication and loss events. The utility of this NP-hard problem for large-scale phylogenetic analyses has been largely limited by the high time complexity of its existing heuristics. These heuristics perform a stepwise search of the tree space, where each step is guided by an exact solution to an instance of a local search problem. We present new algorithms for these local search problems that improve on the time complexity of the best known solutions by a factor of $n$ (the number of taxa in the species supertree). This makes the duplication-loss problem much more tractable for rigorous large-scale phylogenetic analyses. We verify the performance of our algorithms in practice by a comparison study using sets of large randomly generated gene trees.

## 3.1 Introduction

Large-scale phylogenetic analysis is of fundamental importance to comparative genomics and ubiquitous in evolutionary biology. *Phylogenetics* is the study of the evolutionary development or history of a species. Most phylogenetic analyses combine genomic sequences, from

presumably orthologous loci, or in other words, loci whose homology (similarity) is the result of a speciation event, into gene trees. Typically, it can be assumed that such gene trees are similar to the actual species trees. However, these analyses have to neglect the vast amounts of available sequence information in which gene duplication and gene loss cause gene trees to differ from the species tree. Phylogenetic information from such gene trees can be utilized by reconciling the gene trees with a species tree based on the duplication-loss cost (also known as the mutation cost) [29]. The duplication-loss cost captures the minimum number of gene duplications and gene losses that are necessary to reconcile the inconsistencies of the gene trees with the species tree. The corresponding optimization problem, the *duplication-loss* problem [31], is to find a species tree with the minimum duplication-loss cost for a given collection of gene trees. The decision variant of the duplication-loss problem is NP-complete [33]. Other approaches make use of sequence similarity to reconstruct the underlying evolutionary history of genes (see, for example, [51, 52]). Probabilistic models for gene/species tree reconciliation as well as gene sequence evolution have also been developed [2, 3].

Existing heuristics aimed at solving the duplication-loss problem have shown much potential for building accurate species trees (see, for example, [45, 39, 40, 20, 43]). These heuristics search the space of all possible species trees guided by a series of exact solutions to instances of a local search problem [38, 35]. The local search problem is to find an optimal species tree under the duplication-loss cost in the neighborhood of a given tree. The neighborhood is the set of all phylogenetic trees into which the given species tree can be transformed by applying a tree edit operation. The rapidly increasing availability of whole genome data has lent the duplication-loss problem especially desirable for performing phylogenetic analyses. Yet, the high time complexity of duplication-loss local search problems has largely limited its applicability for large-scale phylogenetic analyses.

The duplication-loss problem is closely related to the gene-duplication problem, in which the cost is defined strictly in the number of gene duplications. Recently, efficient solutions were given for the standard SPR [4] and TBR [8] local search problems for the gene-duplication problem. However, due to the added complexity of computing losses, it has remained unclear

whether the SPR and TBR local search problems associated with the duplication-loss problem could be significantly speeded-up as well. In this work we answer this question in the affirmative by providing algorithms that improve on the complexity of the best known solutions for both the SPR and TBR local search problems by a factor of $n$, where $n$ is the size of the resulting species supertree.

### 3.1.1 Previous Results

The duplication-loss problem is based on the Gene Duplication model from Goodman et al. [29]. In the following, we (i) describe the Gene Duplication model, (ii) formulate the duplication-loss problem, and (iii) describe the standard local search heuristic [38, 35] used to solve the duplication-loss problem.

#### 3.1.1.1 Gene Duplication Model

The Gene Duplication model is well studied [37, 31, 36, 54, 18, 10, 30] and explains incompatibilities between a pair of "comparable" gene and species trees through gene duplications. A gene and a species tree are *comparable*, if a *leaf-mapping* exists that provides a leaf to leaf mapping that maps every gene to the species from which it was sampled. The minimum number of gene duplications and gene losses that are necessary under the Gene Duplication model to explain the incompatibilities can be inferred from the mapping $\mathcal{M}$, which is an extension of the given leaf-mapping. $\mathcal{M}$ maps every gene in the gene tree to the most recent species in the species tree that could have contained the gene. More precisely, $\mathcal{M}$ maps each gene to the least common ancestor of the species from which the leaves (genes) of the subtree rooted at the gene were sampled (given by the leaf-mapping). An ancestral gene[1] in the gene tree is a *gene duplication* if it has a child with the same $\mathcal{M}$ mapping. Similarly, the number of *losses* associated with an ancestral gene is roughly equal to the number of ancient species between the mapping of the gene and the mappings of its children. A more precise definition of losses appears in Section 3.2.2.

[1] An ancestral gene is a gene that corresponds to some internal node in the gene tree.

The duplication-loss cost (also known as the mutation cost) for a gene tree and a comparable species tree is measured in the number of gene duplications and the number of gene losses. The duplication-loss cost for a given collection of gene trees and a species tree is the sum of the duplication-loss costs for each gene tree in the collection and the species tree. The mapping function is linear time computable [54] through a reduction from the least common ancestor problem [9]. Consequently, the duplication-loss cost for a collection of gene trees and a species tree is computable in linear time.

#### 3.1.1.2 Duplication-Loss Problem and Heuristics

The *duplication-loss problem* is to find, for a given set of gene trees, a comparable species tree with minimum duplication-loss cost. This approach has been successfully applied to phylogenetic inference in snakes [45], vertebrates [39, 40], *Drosophila* [20], and plants [43] among others. However, the decision variant of this problem and some of its characterizations are NP-complete [33, 26]. The problem is also known to be fixed parameter tractable [32] for a particular parameterization. In practice, heuristics (e.g. [38, 35]) are commonly used for the duplication-loss problem, even though they are unable to guarantee an optimal solution. In these heuristics, a *tree graph* (see [1, 44]) is defined for the given set of gene trees and some fixed tree edit operation. Each node in the tree graph represents a unique species tree comparable with the given gene trees. An edge is drawn between two nodes exactly if the corresponding trees can be transformed into each other by one tree edit operation. The duplication-loss cost of a node in the graph is the duplication-loss cost of the species tree represented by that node and the given gene trees. Given an initial node in the tree graph, the heuristic's task is to find a maximal-length path of steepest descent in the duplication-loss cost of its nodes and to return the last node on such a path. This path is found by solving the local search problem for every node along the path. The local search problem is to find a node with the minimum duplication-loss cost in the neighborhood of a given node. The time complexity of the local search problem depends on the tree edit operation used. Edit operations of interest are rooted subtree pruning and regrafting (SPR) [11] and rooted tree bisection and reconnection (TBR) [17].

For convenience, assume that the size of the $k$ given gene trees differs by a constant factor from the size of the resulting species tree, which we denote by $n$. The best known (naive) solutions for the SPR and TBR local search problems require $\Theta(kn^3)$ and $\Theta(kn^4)$ time respectively, where $k$ is the number of input gene trees.

### 3.1.2 Contribution of this Work

We introduce efficient algorithms for local search heuristics based on SPR and TBR neighborhoods. Our algorithms solve the SPR and TBR local search problems in $O(kn^2)$ and $O(kn^3)$ time respectively. Consequently, our algorithms provide a speedup of $\Theta(n)$ over the best known algorithms for both of these local search problems. The exceptional speed-ups achieved make the duplication-loss problem much more tractable for large-scale phylogenetic analyses.

Recently, in [16], Chauve et al. introduced a Loss-only variant of the duplication-loss problem. This problem seeks to find, for the given set of gene trees, a species supertree that minimizes the loss cost. Our efficient local search algorithms for SPR and TBR also work under this Loss-only optimization setting. Chauve et al. [16] also presented an algorithm which, for a given gene tree, heuristically constructs a species tree minimizing the total loss cost against the gene tree. This heuristic could possibly be used to obtain good starting species trees for our local search algorithms.

We implemented our algorithm for the SPR local search and demonstrate the improvement it offers over the best current solutions by applying it to several large simulated datasets.

## 3.2 Basic Notation and Preliminaries

In this section we first introduce basic definitions and notation, and then the necessary preliminaries required for this work. For the most part, we follow the basic definitions, notation, and preliminaries from Chapter 2.

### 3.2.1 Basic Definitions and Notation

A *tree* $T$ is a connected graph with no cycles, consisting of a node set $V(T)$ and an edge set $E(T)$. $T$ is *rooted* if it has exactly one distinguished node called the *root* which we denote by $rt(T)$. Let $T$ be a rooted tree. We define $\leq_T$ to be the partial order on $V(T)$ where $x \leq_T y$ if $y$ is a node on the path between $rt(T)$ and $x$. The set of minima under $\leq_T$ is denoted by $Le(T)$ and its elements are called *leaves*. The set of *internal nodes* of $T$, denoted $I(T)$, is defined to be $V(T) \setminus Le(T)$. If $\{x, y\} \in E(T)$ and $x \leq_T y$ then we call $y$ the *parent* of $x$ denoted by $pa_T(x)$ and we call $x$ a *child* of $y$. The set of all children of $y$ is denoted by $Ch_T(y)$. If two nodes in $T$ have the same parent, they are called *siblings*. The *least common ancestor* of a non-empty subset $L \subseteq \mathcal{V}(T)$ in tree $T$, denoted as $\text{lca}_T(L)$, is the unique smallest upper bound of $L$ under $\leq_T$. A *subtree* of $T$ rooted at node $y \in V(T)$, denoted by $T_y$, is the tree induced by $\{x \in V(T) : x \leq y\}$. Given $x, y \in V(T)$, $x \rightarrow_T y$ denotes the unique path from $x$ to $y$ in $T$. We denote by $d_T(x, y)$ the number of edges on the path $x \rightarrow_T y$. $T$ is fully *binary* if every node has either zero or two children. Throughout this paper, the term tree refers to a rooted fully binary tree.

Given $T$ and a set $L \subseteq Le(T)$, let $T'$ be the minimal rooted subtree of $T$ with leaf set $L$. We define the leaf induced subtree $T[L]$ of $T$ on leaf set $L$ to be the tree obtained from $T'$ by successively removing each non-root node of degree two and adjoining its two neighbors.

### 3.2.2 The Duplication-Loss Problem

We now introduce necessary definitions to state the gene-duplication problem. A *species tree* is a tree that depicts the evolutionary relationships of a set of species. Given a gene family for a set of species, a *gene tree* is a tree that depicts the evolutionary relationships among the sequences encoding only that gene family in the given species. Thus, the nodes in a gene tree represent genes. We shall assume that each leaf of the gene trees is labeled with the species from which that gene was sampled. In order to compare a gene tree $G$ with a species tree $S$ a mapping from each gene $g \in V(G)$ to the most recent species in $S$ that could have contained $g$ is required.

**Definition 3.2.1** (Mapping)**.** *The* leaf-mapping $\mathcal{L}_{G,S}\colon Le(G) \to Le(S)$ *maps a leaf node* $g \in Le(G)$ *to that unique leaf node* $s \in Le(S)$ *which has the same label as* $g$*. The extension* $\mathcal{M}_{G,S}\colon V(G) \to V(S)$ *of* $\mathcal{L}_{G,S}$ *is the* mapping *defined by* $\mathcal{M}_{G,S}(g) = \mathrm{lca}(\mathcal{L}_{G,S}(Le(G_g)))$.

Note: For any node $s \in V(S)$, $\mathcal{M}^{-1}_{G,S}(s)$ denotes the set of nodes in $G$ that map to node $s \in V(S)$ under the mapping $\mathcal{M}_{G,S}$.

**Definition 3.2.2** (Comparability)**.** *Given trees* $G$ *and* $S$*, we say that* $G$ *is* comparable *to* $S$ *if, for each* $g \in Le(G)$*, the leaf-mapping* $\mathcal{L}_{G,S}(g)$ *is well defined. A set of gene trees* $\mathcal{G}$ *is* comparable *to* $S$ *if each gene tree in* $\mathcal{G}$ *is comparable to* $S$*.*

Throughout this paper we use the following terminology: $\mathcal{G}$ is a set of gene trees that is comparable to a species tree $S$, and $G \in \mathcal{G}$.

**Definition 3.2.3** (Duplication)**.** *A node* $g \in V(G)$ *is a* (gene) duplication *if* $\mathcal{M}_{G,S}(g) \in \mathcal{M}_{G,S}(Ch(g))$ *and we define* $Dup(G,S) = \{g \in V(G)\colon\ g \text{ is a duplication}\}$.

Following [32], we define the number of losses as follows.

**Definition 3.2.4** (Losses)**.** *The number of* losses $Loss(G,S,g)$ *at a node* $g \in I(G)$*, is defined to be:*

- $0$*, if* $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g')\ \forall g' \in Ch_G(g)$ *, and*
- $\sum_{g \in Ch_G(g)} |d_{S'}(\mathcal{M}_{G,S'}(g), \mathcal{M}_{G,S'}(g')) - 1|$ *, otherwise;*

*where* $S' = S[Le(G)]$*. We define* $Loss(G,S) = \sum_{g \in I(G)} Loss(G,S,g)$ *to be the number of losses in* $G$*.*

Under the duplication-loss model, the reconciliation cost for $G$ with respect to $S$ is simply the duplication-loss cost; that is, the number of duplications and losses.

**Definition 3.2.5** (Reconciliation cost)**.** *We define reconciliation costs for gene and species trees as follows:*

1. $\Delta(G,S) = |\,Dup(G,S)| + Loss(G,S)$ *is the* reconciliation cost from $G$ to $S$*.*

2. $\Delta(\mathcal{G}, S) = \sum_{G \in \mathcal{G}} \Delta(G, S)$ *is the* reconciliation cost *from* $\mathcal{G}$ *to* $S$.

3. *Let* $\mathcal{T}$ *be the set of species trees that is comparable with* $\mathcal{G}$. *We define* $\Delta(\mathcal{G}) = \min_{S \in \mathcal{T}} \Delta(\mathcal{G}, S)$ *to be the* reconciliation cost of $\mathcal{G}$.

**Problem 4** (Duplication-Loss)**.**

Instance: *A set* $\mathcal{G}$ *of gene trees.*

Find: *A species tree* $S^*$ *comparable with* $\mathcal{G}$, *such that* $\Delta(\mathcal{G}, S^*) = \Delta(\mathcal{G})$.

### 3.2.3 Local Search Problems

Here we first provide the definition of an SPR edit operation [11] and then formulate the related local search problems that were motivated in the Introduction. The definition and associated local search problems for the TBR edit operation are considered later in Section 3.5.

**Definition 3.2.6** (SPR operation)**.** *(See Fig. 3.1) For technical reasons we first define for a tree* $T$ *the* planted tree $\Phi(T)$ *that is the tree obtained by adding an additional edge, called* root edge, $\{p, rt(T)\}$ *to* $T$.

*Let* $T$ *be a tree,* $e = (u, v) \in E(T)$ *and* $X, Y$ *be the connected components that are obtained by removing edge* $e$ *from* $T$ *such that* $v \in X$ *and* $u \in Y$. *We define* $\mathsf{SPR}_T(v, y)$ *for* $y \in Y$ *to be the tree that is obtained from* $\Phi(T)$ *by first removing edge* $e$, *and then adjoining a new edge* $f$ *between* $v$ *and* $Y$ *as follows:*

1. *Create a new node* $y'$ *that subdivides the edge* $(pa(y), y)$.
2. *Add edge* $f$ *between nodes* $v$ *and* $y'$.
3. *Suppress the node* $u$, *and rename* $y'$ *as* $u$.
4. *Contract the root edge.*

*We say that the tree* $\mathsf{SPR}_T(v, y)$ *is obtained from* $T$ *by a* subtree prune and regraft (SPR) *operation that* prunes *subtree* $T_v$ *and* regrafts *it above node* $y$.

**Notation.** We define the following:

1. $\mathsf{SPR}_T(v) = \bigcup_{y \in Y} \{\mathsf{SPR}_T(v, y)\}$
2. $\mathsf{SPR}_T = \bigcup_{(u,v) \in E(T)} \mathsf{SPR}_T(v)$

We now define the relevant local search problems based on the SPR operation.

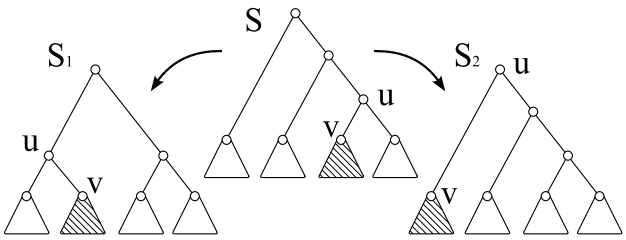


Figure 3.1 $S_1$ and $S_2$ are obtained from $S$ by pruning the subtree rooted at $v$ and regrafting it into the remaining tree $S$.

**Problem 5** (SPR-Scoring (SPR-S))**.**

Instance: *A set $\mathcal{G}$ of gene trees, and a species tree $S$ such that $\bigcup_{G\in\mathcal{G}}\bigcup_{g\in Le(G)}\mathcal{L}_{G,S}(g) = Le(S)$.*

Find: *A tree $T^* \in \mathsf{SPR}_S$ such that $\Delta(\mathcal{G}, T^*) = \min_{T\in\mathsf{SPR}_S}\Delta(\mathcal{G}, T)$.*

Our goal, as seen in the Introduction, is to solve the SPR-S problem efficiently. To that end, we first define a restricted version of the SPR-S problem, called the SPR-*Restricted Scoring* Problem.

**Problem 6** (SPR-Restricted Scoring (SPR-RS))**.**

Instance: *A set $\mathcal{G}$ of gene trees, a species tree $S$ such that $\bigcup_{G\in\mathcal{G}}\bigcup_{g\in Le(G)}\mathcal{L}_{G,S}(g) = Le(S)$, and a non-root node $v$ in $V(S)$.*

Find: *A tree $T^* \in \mathsf{SPR}_S(v)$ such that $\Delta(\mathcal{G}, T^*) = \min_{T\in\mathsf{SPR}_S(v)}\Delta(\mathcal{G}, T)$.*

Throughout the remainder of this manuscript, $S$ denotes a species tree such that $Le(S) = \bigcup_{G\in\mathcal{G}}\bigcup_{g\in Le(G)}\mathcal{L}_{G,S}(g)$, $n$ is the number of leaves in $S$, and $v$ is a non-root node in $V(S)$.

Let $n = |\,Le(S)|$, $m = |\,Le(S)| + |\,Le(G)|$ and $k = |\mathcal{G}|$, and let us assume, for convenience, that all $G \in \mathcal{G}$ have approximately the same size. In the following, we show how to solve the SPR-RS problem in $O(km)$ time. Since $\mathsf{SPR}_S = \bigcup_{\{pa(v),v\}\in E(S)}\mathsf{SPR}_S(v)$ , it is easy to see that the SPR-S problem can be solved by solving the SPR-RS problem $O(n)$ times. This yields an $O(kmn)$ time algorithm for the SPR-S problem. Later, in Section 3.5, we show that the local search problem corresponding to the TBR operation reduces to solving $O(n^2)$ SPR-RS problems; which yields an $O(kmn^2)$ time algorithm for the TBR local search problem.

## 3.3 Solving the SPR-RS Problem

Throughout this section, we shall limit our attention to one gene tree $G$; in particular, we show how to solve the SPR-RS problem for $G$ in $O(m)$ time. Our algorithm extends trivially to solve the SPR-RS problem on the set of gene trees $\mathcal{G}$ in $O(km)$ time. For simplicity, we will assume that $Le(G) = Le(S)$.[2]

In order to solve the SPR-RS problem for $G$, it is sufficient to compute only the values $|\,Dup(G, S')|$ and $Loss(G, S')$ for each $S' \in \mathsf{SPR}_S(v)$. Bansal et al. [4] (see also Chapter 2) showed how to compute the value $|\,Dup(G, S')|$ for each $S' \in \mathsf{SPR}_S(v)$, in $O(m)$ time. Therefore, in the remainder of this section we concentrate on showing how to compute the value $Loss(G, S')$ for each $S' \in \mathsf{SPR}_S(v)$ in $O(m)$ time as well. Altogether, this implies that the SPR-RS problem for $G$ can be solved in $O(m)$ time.

Recall that an efficient solution for the version of the SPR-RS problem in which the reconciliation cost is defined strictly in terms of gene duplications has already been given in [4]. The problem of additionally incorporating losses into the reconciliation cost might seem like a simple addition to the results in [4], but achieving this without increasing the time complexity is non-trivial and quite technical. This is because, when the species tree is modified, losses behave very differently compared to gene duplications. However, before we proceed to study the behavior of losses in detail, we first introduce some of the basic structural properties studied in Chapter 2 that are helpful in the current setting as well.

### 3.3.1 Basic Structural Properties

Consider the tree $N^{S,v} = \mathsf{SPR}_S(v, rt(S))$. Observe that, since $\mathsf{SPR}_{N^{S,v}}(v) = \mathsf{SPR}_S(v)$, solving the SPR-RS problem on instance $\langle\{G\}, S, v\rangle$ is equivalent to solving it on the instance $\langle\{G\}, N^{S,v}, v\rangle$. Thus, in the remainder of this section, we will work with tree $N^{S,v}$ instead of tree $S$; the reason for this choice becomes clear in light of Lemmas 3.3.3 and 3.3.4.

Since $S$ and $v$ are fixed in the current context, we will, in the interest of clarity, abbreviate

[2]Note: if $Le(G) \neq Le(S)$ then we can simply set the species tree to be $S[Le(G)]$. This takes $O(n)$ time and, consequently, does not affect the time complexity of our algorithm.

$N^{S,v}$ simply to $N$. Similarly, in the remainder of this section, we abbreviate $\mathcal{M}_{G,T}$ to $\mathcal{M}_T$, for any species tree $T$.

Throughout the remainder of this chapter, let $u$ denote the sibling of $v$ in $N$. We color the nodes of $N$ as follows: (i) All nodes in the subtree $N_v$ are colored red, (ii) the root node of $N$ is colored blue, and (iii) all the remaining nodes, i.e. all nodes in $N_u$, are colored green. See Figure 3.2 for an example. Correspondingly, we color the nodes of $G$ by assigning to each $g \in V(G)$ the color of the node $\mathcal{M}_N(g)$.

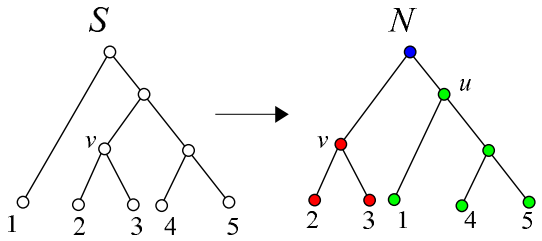


Figure 3.2 Example depicting the construction of the tree $N$ from $S$, and the subsequent coloring of the nodes in $N$.

Now consider the version of $G$ obtained by removing all red nodes from it. The leftover, which we shall call $\Gamma$, must also be a tree. See Figure 3.3 (taken from Chapter 2) for an example.

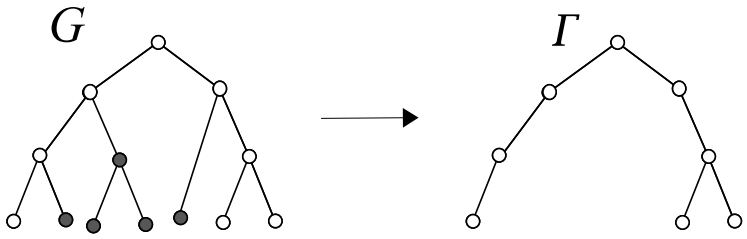


Figure 3.3 The tree $\Gamma$ is obtained from $G$ by removing all the red (shaded) nodes.

The following two Lemmas are taken from Chapter 2.

**Lemma 3.3.1.** *Given $G$ and $N$, if $g \in V(G)$ is either red or green, then $\mathcal{M}_{S'}(g) = \mathcal{M}_N(g)$ for all $S' \in \mathtt{SPR}_N(v)$.*

**Lemma 3.3.2.** *Given $G$ and $N$, if $g \in V(G)$ is a blue node, then $\mathcal{M}_{S'}(g) = \mathrm{lca}_{S'}(v, \mathcal{M}_{\Gamma,N}(g))$ for any $S' \in \mathsf{SPR}_N(v)$.*

Lemmas 3.3.1 and 3.3.2 together completely characterize the mappings from nodes in $V(G)$ for each $S' \in \mathsf{SPR}_S(v)$. This characterization will be used (often without explicit reference to the two lemmas) throughout the proofs of Lemmas 3.3.3 through 3.3.8.

### 3.3.2 Characterizing Losses

To solve the SPR-RS problem efficiently we rely on the following six lemmas, which make it possible to efficiently infer the value of $\mathit{Loss}(G, S', g)$ for any $S' \in \mathsf{SPR}_N(v)$ and any $g \in V(G)$.

Consider any $g \in I(G)$, and let $g'$ and $g''$ be its two children. Let $a = \mathcal{M}_N(g)$, $b = \mathcal{M}_N(g')$ and $c = \mathcal{M}_N(g'')$. Without loss of generality, node $g$ must correspond to one of the following six categories.

1. $g$ is red,
2. $g$ is green,
3. $g$, $g'$, and $g''$ are all blue,
4. $g$ and $g'$ are blue, and $g''$ is green,
5. $g$ and $g'$ are blue, and $g''$ is red, and,
6. $g$ is blue, $g'$ is red, and $g''$ is green.

Lemmas 3.3.3 through 3.3.8 characterize the behavior of the loss cost $\mathit{Loss}(G, S', g)$, for each $S' \in \mathsf{SPR}_N(v)$, for each of these six cases. At this point, it would help to observe that $\mathsf{SPR}_N(v) = \{\mathsf{SPR}_N(v, s) \colon s \in V(N_u)\}$.

**Lemma 3.3.3.** *If $g$ is red then $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, N, g)$ for all $S' \in \mathsf{SPR}_N(v)$.*

*Proof.* The subtree $N_v$ is identical in all trees in $\mathsf{SPR}_N(v)$. Moreover, $g$ and all its descendants must map to the same red nodes under the mappings $\mathcal{M}_{G,N}$ and $\mathcal{M}_{G,S'}$, for any $S' \in \mathsf{SPR}_N(v)$. The lemma follows. □

**Lemma 3.3.4.** *If $g$ is green then $Loss(G, S', g) = Loss(G, N, g) + 1$ if $S' = \mathsf{SPR}_N(v, x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, and $Loss(G, S', g) = Loss(G, N, g)$ otherwise.*

*Proof.* Since $g$ is green, so are $g'$ and $g''$, and therefore, by Lemma 3.3.1 we must have $\mathcal{M}_{S'}(y) = \mathcal{M}_N(y)$ for any $S' \in \mathsf{SPR}_N(v)$ and $y \in \{g, g', g''\}$. Thus, if $S' = \mathsf{SPR}_N(v, x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, then either $d_{S'}(a, b) = d_N(a, b) + 1$ or $d_{S'}(a, c) = d_N(a, c) + 1$; and $d_{S'}(a, b) = d_N(a, b)$, $d_{S'}(a, c) = d_N(a, c)$ otherwise. Hence, following Def. 3.2.4, $Loss(G, S', g) = Loss(G, N, g) + 1$ if $S' = \mathsf{SPR}_N(v, x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, and $Loss(G, S', g) = Loss(G, N, g)$ otherwise. □

**Lemma 3.3.5.** *Let $g$, $g'$ and $g''$ all be blue nodes, $x \in V(N_u)$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$.*

1. *If $S' = \mathsf{SPR}_N(v, x)$ where $x \not<_N a'$, then $Loss(G, S', g) = Loss(G, N, g)$.*
2. *If $S' = \mathsf{SPR}_N(v, x)$ where $x <_N a'$, and $S'' = \mathsf{SPR}_N(v, pa(x))$, then,*
   (a) *$Loss(G, S', g) = Loss(G, S'', g) + 1$ if $b' \leq_N x <_N a'$ or $c' \leq_N x <_N a'$, and,*
   (b) *$Loss(G, S', g) = Loss(G, S'', g)$ otherwise.*

*Proof.* First observe that if $g$, $g'$ and $g''$ are all blue nodes then each of $g$, $g'$ and $g''$ must be a node in tree $\Gamma$; and hence, the mappings $\mathcal{M}_{\Gamma,N}(g)$, $\mathcal{M}_{\Gamma,N}(g')$ and $\mathcal{M}_{\Gamma,N}(g'')$ are well defined. Next, we prove the correctness of each part separately.

**Part 1.** In this case we must have $lca_N(v, a') = lca_N(v, b') = lca_N(v, c')$. Therefore, by Lemma 3.3.2, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S'}(g'')$ where $S' = \mathsf{SPR}_N(v, x)$ and $x \not<_N a'$. Thus, for each $S'$ in this case, we must have $Loss(G, S', g) = 0 = Loss(G, N, g)$.

**Part 2.(a)** This case is relevant only if at least one of $b'$ or $c'$ is not the same as $a'$. Therefore, without any loss of generality we may assume that $b' \neq a'$. Suppose $S' = \mathsf{SPR}_N(v, x)$ where $b' \leq_N x <_N a'$; then, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_N(a', x)$. Also, if $b''$ denotes the child of $a$ in tree $N$ along the path $a' \rightarrow_N b'$, then, by Def. 3.2.4, we must have $Loss(G, \mathsf{SPR}_N(v, b''), g) = 1$, which is indeed one greater than

$Loss(G, \mathsf{SPR}_N(v, a'), g)$. Thus, $Loss(G, S', g) = Loss(G, S'', g) + 1$ if $b' \leq_N x <_N a'$. The argument for the case when $c' \leq_N x <_N a'$ is completely analogous.

**Part 2.(b)** Let $b''$ denote the child of $a$ along the path $a' \rightarrow_N b'$, and $c''$ denote the child of $a$ along the path $a' \rightarrow_N b'$, in tree $N$. When $x <_N a'$ but neither $b' \leq_N x <_N a'$ nor $c' \leq_N x <_N a'$, we must have either (i) $x <_N b''$ but not such that $b' \leq_N x <_N a'$, or (ii) $x <_N c''$ but not such that $b' \leq_N x <_N a'$. In case (i), we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = \mathcal{M}_{S'}(g'') = \mathcal{M}_{S'}(g'') = a'$, and both $\mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g')$ must be nodes along the path $b'' \rightarrow_{S''} b'$ such that $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g'))$ (note that this is true even if $b' \leq_N pa(x) <_N a'$). Thus, for case (i), $Loss(G, S', g) = Loss(G, S'', g)$. An analogous argument holds for case (ii).

□

**Lemma 3.3.6.** *Let $g$ and $g'$ be blue nodes and $g''$ be a green node, $x \in V(N_u) \setminus \{u\}$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$.*

1. *If $S' = \mathsf{SPR}_N(v, x)$ where $x \not\leq_N a'$, and $S'' = \mathsf{SPR}_N(v, pa(x))$, then,*
   - *(a) $Loss(G, S', g) = Loss(G, S'', g) - 1$ if $a' \leq_N x <_N u$,*
   - *(b) $Loss(G, S', g) = Loss(G, S'', g) - 1$ if $a' \leq_N pa(x) <_N u$ but $x$ is not such that $a' \leq_N x <_N u$, and,*
   - *(c) $Loss(G, S', g) = Loss(G, S'', g)$ otherwise.*
2. *Let $S' = \mathsf{SPR}_N(v, x)$ where $x <_N a'$ and $S'' = \mathsf{SPR}_N(v, pa(x))$.*
   - *(a) If $a' \neq b'$ and $b''$ denotes the child of $a'$ along the path $a' \rightarrow_N b'$, then,*
     - *i. $Loss(G, \mathsf{SPR}_N(v, b''), g) = Loss(G, \mathsf{SPR}_N(v, a'), g) - 2$ if $a' \neq c'$. And, $Loss(G, \mathsf{SPR}_N(v, b''), g) = Loss(G, \mathsf{SPR}_N(v, a'), g)$ if $a' = c'$,*
     - *ii. $Loss(G, S', g) = Loss(G, S'', g) + 1$ if $b' \leq_N x <_N b''$,*
     - *iii. $Loss(G, S', g) = Loss(G, S'', g)$ if $x$ is such that $x \in V(N_{b''})$ but not such that $b' \leq_N x <_N a'$,*

*iv.* $Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g)$ *if* $c' \leq_N x <_N a'$, *and,*

*v.* $Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g) - 1$ *otherwise.*

*(b) If* $a' = b'$, *then,*

*i.* $Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g)$ *if* $c' \leq_N x <_N a'$, *and,*

*ii.* $Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g) - 1$ *otherwise.*

*Proof.* First observe that $g$ and $g'$ are blue and $g''$ is green. Thus, each of $g$, $g'$ and $g''$ must be a node of tree $\Gamma$; and hence, the nodes $a'$, $b'$ and $c'$ are well defined. Also observe that $c' = c$. Next, we prove the correctness of each part separately.

**Part 1.(a)** For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_{S'}(x)$, and $\mathcal{M}_{S'}(g'') = c'$. Also observe that the same holds for the case when $x = u$. Thus, for each $x$ such that $a' \leq_N x <_N u$, we have $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c') - 1$. Part 1.(a) of the lemma now follows immediately.

**Part 1.(b)** In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_N(x)$, and $\mathcal{M}_{S'}(g'') = c'$, and, $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g') = pa_{S''}(x)$, and $\mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c') - 1$. Hence, $Loss(G, S', g) = Loss(G, S'', g) - 1$.

**Part 1.(c)** In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = lca_{S''}(x, a')$ and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c')$. Thus, $Loss(G, S', g) = Loss(G, S'', g)$.

**Part 2.(a).i.** Let $T$ and $T'$ denote the trees $\mathsf{SPR}_N(v, a')$ and $\mathsf{SPR}_N(v, b'')$ respectively. Then, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$ and $\mathcal{M}_T(g'') = c'$, and, $\mathcal{M}_{T'}(g) = a'$, $\mathcal{M}_{T'}(g') = pa_{T'}(b'')$ and $\mathcal{M}_{T'}(g'') = c'$. For the case when $a' \neq c'$ we must therefore have $d_T(\mathcal{M}_T(g), c') = d_{T'}(\mathcal{M}_T(g), c') + 1$, and $d_T(\mathcal{M}_{T'}(g), \mathcal{M}_{T'}(g')) = 1$. Note that while $g$ is a duplication under mapping $\mathcal{M}_T$, it is not one under mapping $\mathcal{M}_{T'}$. Thus, by Definition 3.2.4, we must have $Loss(G, T', g) = Loss(G, T, g) - 2$. For the case when, $a' = c'$, we must

have $\mathcal{M}_T(g'') = a'$, and therefore, $d_T(\mathcal{M}_T(g), \mathcal{M}_T(g'')) = 1$, $d_T(\mathcal{M}_T(g), \mathcal{M}_T(g')) = 0$, and, $d_{T'}(\mathcal{M}_T(g), \mathcal{M}_T(g'')) = 0$, $d_{T'}(\mathcal{M}_T(g), \mathcal{M}_T(g')) = 1$. Thus, $Loss(G, T', g) = Loss(G, T, g)$.

**Part 2.(a).ii.** This case is relevant only if $b' \neq b''$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, $\mathcal{M}_{S'}(g') = pa_{S'}(x)$ and $\mathcal{M}_{S''}(g') = pa_{S''}(pa(x))$. Thus, $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g'))+1$, and consequently $Loss(G, S', g) = Loss(G, S'', g)+1$.

**Part 2.(a).iii.** In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, and both $\mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g')$ must be nodes along the path $b'' \rightarrow_{S''} b'$ such that $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g'))$ (note that this is true even if $b' \leq_N pa(x) <_N a'$). The result follows.

**Part 2.(a).iv.** This case exists only if $a' \neq c'$. Let $T$ denote the tree $\mathsf{SPR}_N(v, a')$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$. Therefore, $d_{S'}(a', c') = d_T(\mathcal{M}_T(g), c') = d_N(a', c')+1$. Thus, if $c' \leq_N x <_N a'$, then $Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g)$.

**Part 2.(a).v.** Let $c''$ denote the sibling of $b''$ in tree $N$. Then, in this case, we must have $x \in N_{c''}$. Moreover, $x$ is not such that $c' \leq_N x <_N a'$. Thus, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_{S'}(g'') = c'$. Also, for the tree $T = \mathsf{SPR}_N(v, a')$, we have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_T(a')$ and $d_T(\mathcal{M}_{S'}(g), c') = d_{S'}(a', c')+1$. Hence, $Loss(G, S', g) = Loss(G, T, g)-1$.

**Part 2.(b).i.** The proof for this part is identical to the proof of part *2.(a).iv.*

**Part 2.(b).ii.** Let $T$ denote the tree $\mathsf{SPR}_N(v, a')$. There are two possible cases, either $a' = c'$ or $a' \neq c'$. For $a' = c'$, we must have $Loss(G, T, g) = 1$ and $Loss(G, S', g) = 0$. For $a' \neq c$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_T(g'') = c'$; and hence $Loss(G, S', g) = Loss(G, T, g) - 1$. Thus, part *2.(b).ii.* of the lemma holds for both cases.

□

**Lemma 3.3.7.** *Let $g$ and $g'$ be blue nodes and $c$ be a red node, $x \in V(N_u)$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$.*

1. *If $S' = \mathsf{SPR}_N(v, x)$ where $x <_N a'$, and $S'' = \mathsf{SPR}_N(v, pa(x))$, then $Loss(G, S', g) = Loss(G, S'', g) + 1$.*

2. *If $S' = \mathsf{SPR}_N(v, x)$ where $x \not<_N a'$, then,*

   (a) *$Loss(G, S', g) = Loss(G, N, g)$ if $a' \leq_N x \leq_N u$, and,*

   (b) *$Loss(G, S', g) = Loss(G, S'', g) + 1$ for $S'' = \mathsf{SPR}_N(v, pa(x))$ otherwise.*

*Proof.* First observe that since both $g$ and $g'$ are blue, they must be nodes of tree $\Gamma$; and hence, the mappings $\mathcal{M}_{\Gamma,N}(g)$, and $\mathcal{M}_{\Gamma,N}(g')$ are well defined. Also, since $g'' \notin V(\Gamma)$, by the definition of $\Gamma$, we must have $\mathcal{M}_{\Gamma,N}(g) = \mathcal{M}_{\Gamma,N}(g')$. Next, we prove the correctness of each part separately.

**Part 1.** If $x <_N a'$, then $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$ and $\mathcal{M}_{S'}(g'') = c$. Since $g''$ is red, $c$ must be a node in the pruned subtree $N_v$, therefore, assuming $S'' \neq \mathsf{SPR}_N(v, a')$, we must have $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$ and, consequently, $Loss(G, S', g) = Loss(G, S'', g) + 1$. If $S'' \neq \mathsf{SPR}_N(v, a')$, then we have $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g') = pa_{S''}(a')$ and $\mathcal{M}_{S''}(g'') = c$. And therefore, again, we must have $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$, implying $Loss(G, S', g) = Loss(G, S'', g) + 1$.

**Part 2.(a)** In the tree $N$ we have $\mathcal{M}_N(g) = \mathcal{M}_N(g') = rt(N)$ and $d_N(rt(N), c) = d_{N_v}(v, c) + 1$. Similarly, if $a' \leq_N x \leq_N u$, then $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_{S'}(x)$ and, consequently, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{N_v}(v, c) + 1$. Thus, in this case $Loss(G, S', g) = Loss(G, N, g)$.

**Part 2.(b)** We have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g')$. Now, if $a' \leq_N pa(x) \leq_N u$, then we must have $\mathcal{M}_{S''}(g) = pa_{S''}(pa_N(x))$ and $\mathcal{M}_{S'}(g) = pa_N(x)$, and therefore, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$; otherwise, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = lca_N(x, a')$ and therefore, again, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$. Part 2. (b) of the lemma now follows directly.

□

**Lemma 3.3.8.** *Let $g$ be blue, $g'$ be red, and $g''$ be green. Let $x \in V(N_u)\backslash\{u\}$ and $a' = \mathcal{M}_{\Gamma,N}(g)$.*

1. *If $S' = \mathsf{SPR}_N(v, x)$ where $x \not<_N a'$, and $S'' = \mathsf{SPR}_N(v, pa(x))$, then,*
   (a) *$Loss(G, S', g) = Loss(G, S'', g) - 1$ if $a' \leq_N x <_N u$,*
   (b) *$Loss(G, S', g) = Loss(G, S'', g)$ if $a' \leq_N pa(x) <_N u$ but $x$ is not such that $a' \leq_N x <_N u$, and,*
   (c) *$Loss(G, S', g) = Loss(G, S'', g) + 1$ otherwise.*
2. *If $S' = \mathsf{SPR}_N(v, x)$ where $x <_N a'$, and $S'' = \mathsf{SPR}_N(v, pa(x))$, then,*
   (a) *$Loss(G, S', g) = Loss(G, \mathsf{SPR}_N(v, a'), g) + 2$ if $x \in Ch_N(a')$, and,*
   (b) *$Loss(G, S', g) = Loss(G, S'', g) + 1$ otherwise.*

*Proof.* First observe that since $g$ is blue, the mapping $\mathcal{M}_{\Gamma,N}(g)$ is well defined. Moreover, by the definition of tree $\Gamma$, we must have $a' = \mathcal{M}_{\Gamma,N}(g) = c$. Next, we prove the correctness of each part separately.

**Part 1.(a)** For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = pa_{S'}(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$. Also observe that the same holds for the case when $x = u$. Thus, for each $x$ such that $a' \leq_N x <_N u$, we have $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'))$ and $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g'')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'')) - 1$. Part 1.(a) of the lemma follows immediately.

**Part 1.(b)** In this case, we must have $\mathcal{M}_{S'}(g) = pa_N(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$, and, $\mathcal{M}_{S''}(g) = pa_{S''}(x)$, $\mathcal{M}_{S''}(g') = b$ and $\mathcal{M}_{S''}(g'') = a'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and $d_{S'}(\mathcal{M}_{S'}(g), a') = d_{S''}(\mathcal{M}_{S''}(g), a') - 1$. Hence, $Loss(G, S', g) = Loss(G, S'', g)$.

**Part 1.(c)** In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = lca_{S''}(b, a')$, $\mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = b$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = a'$. Now since $b$ is a node in the pruned subtree $N_v$, we must have

$d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $Loss(G, S', g) = Loss(G, S'', g) + 1$.

**Part 2.(a)** If $x \in Ch_N(a')$ then we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Thus, $Loss(G, S', g) = |d_{S'}(a', b) - 1| + 1$. Now, let $T$ denote the tree $\mathsf{SPR}_N(v, a')$, then we must have $\mathcal{M}_T(g) = pa_T(a')$, $\mathcal{M}_T(g'') = a'$ and $\mathcal{M}_T(g') = b$. Thus, $Loss(G, T, g) = |d_T(pa_T(a'), b) - 1|$. Finally, observe that $d_{S'}(a', b) = d_T(pa_T(a'), b) + 1$, and hence, $Loss(G, S', g) = Loss(G, T, g) + 2$.

**Part 2.(b)** For any $x <_N a'$, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Since $b$ is a node in the pruned subtree $N_v$ and in this case $x <_N y$ for $y \in Ch(a')$, we must have $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $Loss(G, S', g) = Loss(G, S'', g) + 1$.

□

## 3.4 The Algorithm

Observe that $\mathsf{SPR}_N(v) = \{\mathsf{SPR}_N(v, s) \colon s \in V(N_u)\}$. Therefore, the goal of our algorithm is to compute at each node $s \in V(N_u)$ the value $Loss(G, S')$, where $S' = \mathsf{SPR}_N(v, s)$. The first step is to compute the value $Loss(G, N)$. This "loss value" is assigned to the node $u$. To compute the loss value for the rest of the nodes our algorithm makes use of six different types of *counters* at each node in $N_u$; we shall refer to these counters as counter-$i$, for $i \in \{1, \ldots, 6\}$. The loss values behave in non-trivial ways; however, as we shall see, based on Lemmas 3.3.3 through 3.3.8, this behavior can be broken down into six types of patterns (captured by the six counters). These counters make it possible to compute the difference between the values $Loss(G, N)$ and $Loss(G, S')$, where $S' = \mathsf{SPR}_N(v, s)$, for each $s \in V(N_u)$. Next, we describe each of these six counters; throughout our description, $s$ represents some node in $N_u$.

**counter-1** If the value of counter-1 is $x$ at node $s$ then this implies that the tree $\mathsf{SPR}_N(v, s)$ incurs $x$ additional losses over the value $Loss(G, N)$.

**counter-2** If the value of counter-2 is $x$ at node $s$, then this implies that for each $t \leq_N s$ the tree $\mathsf{SPR}_N(v,t)$ incurs an additional $x$ losses over $Loss(G,N)$.

**counter-3** If the value of counter-3 is $x$ at node $s$, then this implies that for each $t \leq_N s$ the tree $\mathsf{SPR}_N(v,t)$ loses $x$ losses from $Loss(G,N)$.

**counter-4** If the value of counter-4 is $x$ at node $s$, then this implies that for each $t \leq_N s$ the tree $\mathsf{SPR}_N(v,t)$ incurs $\alpha_t \cdot x$ additional losses over $Loss(G,N$, where $\alpha_t = d_N(pa(s),t)$.

**counter-5** If the value of counter-5 is $x$ at node $s$, then it is equivalent to incrementing counter-4 at the sibling of each node on the path $u \rightarrow_N s$, except at $u$, by $x$.

**counter-6** If the value of counter-6 is $x$ at node $s$, then it is equivalent to incrementing counter-4 at both children (if they exist) of the sibling of each node along the path $u \rightarrow_N s$, except $u$, and incrementing counter-3 at each node along the path $u \rightarrow_N s$, except at $u$, by $x$.

Remark: Each of the additions or subtractions implied by these counters are independent of each other. This makes it possible to handle each addition or subtraction implied by these counters separately.

In the remainder of this section we first show how to compute the values of these counters, and then the final loss values, at each node in $N_u$.

### 3.4.1 Computing the Counters

We now describe how the values of the six counters are computed. Initially, each counter at each node in $N_u$ is set to 0. Consider any $g \in I(G)$, and let $g'$ and $g''$ be its two children. Recall that node $g$ must fall under one of the following six categories: 1) $g$ is red, 2) $g$ is green, 3) $g$, $g'$, and $g''$ are all blue, 4) $g$ and $g'$ are blue, and $g''$ is green, 5) $g$ and $g'$ are blue, and $g''$ is red, or, 6) $g$ is blue, $g'$ is red, and $g''$ is green.

Let $a = \mathcal{M}_N(g)$, $b = \mathcal{M}_N(g')$ and $c = \mathcal{M}_N(g'')$. Also, whenever properly defined, let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$. Based on Lemmas 3.3.3 through 3.3.8, we

now study how the six counters can be updated so as to capture the behavior of losses in each of these cases.

**Case 1.** By Lemma 3.3.3, nothing needs to be done in this case.

**Case 2.** Based directly on Lemma 3.3.4, the contribution of any node $g$ that satisfies the condition of case 1 can be captured by simply incrementing the value of counter-1 by one at each node on paths $a \rightarrow_N b$ and $a \rightarrow_N b$, except at node $a$.

**Case 3.** From Lemma 3.3.5 it follows that in this case the contribution of $g$ to the loss value changes in a way that is captured by incrementing counter-2 by 1, at each node, except $a'$, on the paths $a' \rightarrow_N b'$ and $a' \rightarrow_N c'$.

**Case 4.** According to Lemma 3.3.6, if $N_v$ is regrafted on an edge of $N_u$ that is not in $N_{a'}$, then the contribution of $g$ to the loss cost is captured by incrementing counter-3 by 1 at each node except $u$ along the path $u \rightarrow_N a'$, and at their siblings. If $N_v$ is regrafted on an edge of $N_u$ that is in $N_{a'}$ then there are two possible cases:

$a' \neq b'$ Recall that $b''$ represents the child of $a'$ along the path $a' \rightarrow_N b'$. In this case, the contribution of $g$ to the loss cost is captured by (i) incrementing counter-3 by two at node $b''$, (ii) incrementing counter-2 by one at each node along the path $b'' \rightarrow_N b'$ except at node $b''$, (iii) incrementing counter-3 by one at the sibling of $b''$, and (iv) incrementing counter-1 by one at each node except $a'$ on the path $a' \rightarrow_N c'$.

$a' = b'$ In this case, the contribution of $g$ to the loss cost is captured by (i) incrementing counter-3 by one at both children of $a'$, and (ii) incrementing counter-1 by one at each node except $a'$ on the path $a' \rightarrow_N c'$.

**Case 5.** By Lemma 3.3.7, for this case, the change in the loss contribution of $g$ is captured by incrementing counter-5 by 1 at node $a'$, and by incrementing counter-4 by 1 at both children of $a'$ in $N$.

**Case 6.** By Lemma 3.3.8, for this case, the change in the loss contribution of $g$ is captured by incrementing counter-6 by 1 at node $a'$, and by incrementing counter-4 and counter-2 by 1 each at both children of $a'$ in $N$.

Based on these counters we now describe our algorithm to solve the SPR-RS problem.

### 3.4.2 Computing the Final Loss Values

Our algorithm considers each internal node of gene tree $G$, one at a time, and updates the relevant counters at the relevant nodes in $N_u$, as shown in the previous subsection. Then, based on these counters, it computes, at each node $s \in V(N_u)$ the value $\alpha(s) = \mathit{Loss}\,G, S' - \mathit{Loss}\,G, N$. A complete description of our algorithm to solve the SPR-RS problem on instance $\langle \{G\}, S, v \rangle$ appears in Procedure-SPR-RS (see Algorithm 2).

**Algorithm 2** Procedure-SPR-RS

1: **Input:** $G$, $S$, $v$
2: Construct the tree $N$ from $S$.
3: Create and initialize to zero six counters counter-$i$, for $i \in \{1, \ldots, 6\}$, at each $s \in V(N_u)$.
4: Construct the mapping $\mathcal{M}_{G,N}$, color the nodes of $N$ and $G$ as described in Section 3.3.1, and construct the mapping $\mathcal{M}_{\Gamma,N}$.
5: Compute the value $\mathit{Loss}(G, N)$.
6: **for** each node $s \in V(N_u)$ **do**
7: Compute the duplication cost $|\,\mathit{Dup}(G, \mathsf{SPR}_N(v, s))|$ as shown in [4] (also Chapter 2).
8: **for all** each node $g \in I(G)$ **do**
9: Update the counters as shown in Section 3.4.1.
10: Perform a post-order traversal of $N_u$ to transform counter-5 into counter-4 (as explained in the definition of counter-5) throughout $N_u$.
11: Perform a post-order traversal of $N_u$ to transform counter-6 into counter-4 and counter-3 (as explained in the definition of counter-6) throughout $N_u$.
12: Perform a pre-order traversal of $N_u$ to transform counter-4 into counter-2. This can be achieved by incrementing counter-2 at node $s \in V(N_u)$ by the sum of the values of counter-4 at each ancestor of $s$ in $N_u$.
13: Use counter-1, counter-2, and counter-3 to compute the value of $\alpha(s)$ at each $s \in V(N_u)$ by calling Procedure-Loss (see Algorithm 3) on parameters $\langle u, 0 \rangle$.
14: The reconciliation cost of $G$ and the tree $\mathsf{SPR}_S(v, s)$, where $s \in V(N_u)$, is given by $|\,\mathit{Dup}(G, \mathsf{SPR}_N(v, s)| + \alpha(s) - \mathit{Loss}(G, N)$.

**Lemma 3.4.1.** *Procedure-SPR-RS solves the SPR-RS problem on the instance* $\langle \{G\}, S, v \rangle$.

**Algorithm 3** Procedure-Loss

1: **Input:** A node $t$ of the tree $N_u$, and a counter $c$.
2: $c = c +$ counter-2$(t) -$ counter-6$(t)$.
3: $\alpha(t) = \alpha(t) + c +$ counter-1$(t)$.
4: **if** $t$ is not a leaf node of $N$ **then**
5: Let $\{t', t''\} = Ch_N(t)$.
6: Call Procedure-Loss on parameters $\langle t', c\rangle$.
7: Call Procedure-Loss on parameters $\langle t'', c\rangle$.

*Proof.* Procedure-SPR-RS computes the duplication cost $|\,Dup(G, \mathsf{SPR}_N(v,s))|$, at each $s \in V(N_u)$, as shown in [4]. It then computes the values of the six counters, i.e. counter-$i$, for $i \in \{1,\ldots,6\}$, in accordance with Lemmas 3.3.3 through 3.3.8. Then, in Steps 10 through 12, the algorithm encodes the changes in loss cost implied by counter-4, counter-5, and counter-6, at each node in $N_u$, in terms of the values of counter-1, counter-2, and counter-3. procedure-Loss then correctly computes the value of $\alpha(s)$ at each $s \in V(N_u)$ based on these three counters. Thus, for any $\mathsf{SPR}_S(v,s)$, where $s \in V(N_u)$, the values $|\,Dup(G, \mathsf{SPR}_N(v,s))|$ and $\alpha(s)$ are computed correctly. Note that the reconciliation cost of $G$ and $\mathsf{SPR}_S(v,s)$, where $s \in V(N_u)$, is given by $|\,Dup(G, \mathsf{SPR}_N(v,s)| + \alpha(s) - Loss(G,N)$. The lemma follows. □

To simplify our analysis of the time complexity of our algorithm, we assume that all $G \in \mathcal{G}$ have approximately the same size. [3] Recall that $n = |\,Le(S)|$, $m = |\,Le(S)| + |\,Le(G)|$ and $k = |\mathcal{G}|$.

**Lemma 3.4.2.** *Procedure-SPR-RS can be implemented to run in $O(m)$ time.*

*Proof.* We analyze the complexity of Procedure-SPR-RS step-by-step. The total time complexity of Steps 2 and 3 is $O(n)$. Step 4 can be implemented in $O(m)$ time as follows: During an $O(n)$-time preprocessing step we can process the tree $N$ so that lca queries on any two nodes in $V(N)$ can be answered in $O(1)$ time; see [9] for details. Subsequently, the task of constructing the mapping $\mathcal{M}_{G,N}$ only takes $O(|V(G)|)$ time. Coloring the nodes of $G$ and $N$ and constructing the mapping $\mathcal{M}_{\Gamma,N}$ also take $O(|V(G)|)$ time. Thus, the total time complexity of this step is $O(|V(G)| + n)$, which is $O(m)$. In Step 5, the value $Loss(G,N)$ can be computed in $O(m)$

[3] We point out that the $\Theta(n)$ speed-up obtained by our algorithm over the currently best known solution does not depend on this simplifying assumption.

time by first traversing through $N$ to compute the depth of each node, and then traversing through $G$ and computing $Loss(G, N, g)$ for each $g \in I(G)$ according to Definition 3.2.4. The 'for' loop of Step 6 requires $O(m)$ time (see [4]).

Let us now consider the 'for' loop of Step 8 in detail. For any $g$ that satisfies the criteria for case 1, there is nothing to be done. For any $g$ satisfying the criterion for cases 5, or 6, we are required to update the counters at a constant number of nodes in $N_u$. Therefore, all such $g$ can be handled with-in $O(m)$ time. However, for cases 2, 3 and 4, handling each $g$ might, in the worst case, require updating the counters at $\Theta(n)$ nodes, yielding a total time complexity of $O(nm)$ for these cases. This happens because these cases require us to update specific counters along the entire length of certain paths in $N_u$. A simple way to deal with this issue is to only mark the start node and end node of the path for the specified counter. Once this is done for each $g$ satisfying the criterion for cases 2, 3 or 4, we can perform a post-order traversal and set all the relevant counters to their correct value based on the marked start and end nodes. In this way, cases 2, 3, and 4 can be handled in a total of $O(m)$ time as well. This gives us a total time complexity of $O(m)$ for computing all the counters.

In Steps 10 through 12, the algorithm encodes the changes in loss cost implied by counter-4, counter-5, and counter-6, at each node in $N_u$, in terms of the other counters. It is easy to verify that each of these steps can be implemented to run in $O(n)$ time by doing either a post-order or pre-order traversal of $N_u$, as appropriate. Step 13 calls Procedure-Loss on parameters $\langle u, 0\rangle$. Procedure-Loss simply performs a pre-order traversal of the tree $N_u$, spending $O(1)$ at each node. The time complexity of Step 13 is thus $O(n)$. Finally, in Step 14, computing the reconciliation cost for each $\mathsf{SPR}_S(v, s)$, where $s \in V(N_u)$, takes $O(n)$ time. □

Thus, we have the following two theorems.

**Theorem 3.4.1.** *The* SPR*-RS problem can be solved in $O(km)$ time.*

*Proof.* By Lemmas 3.4.1 and 3.4.2 we know that the SPR-RS problem on the restricted instance $\langle\{G\}, S, v\rangle$ can be solved in $O(m)$ time. It is straightforward to extend Procedure-SPR-RS to solve the SPR-RS problem by considering each gene tree in $\mathcal{G}$ separately and combining the computed reconciliation costs. Since there are $k$ gene trees, the theorem follows. □

**Theorem 3.4.2.** *The* SPR-$S$ *problem can be solved in* $O(kmn)$ *time.*

*Proof.* Observe that $\mathsf{SPR}_S = \bigcup_{\{pa(v),v\}\in E(S)} \mathsf{SPR}_S(v)$. The theorem therefore follows immediately from Theorem 3.4.1. □

The time complexity of the best known (naive) solution for the SPR-S problem is $\Theta(kmn^2)$. Our algorithm improves on this by a factor of $n$.

## 3.5 Speeding-Up the TBR Local Search Problem

Heuristics based on the TBR local search problem are particularly desirable since they significantly extend the search space explored at each local search step; however, due to inefficient running times, they have rarely been applied in practice. Our solution to the SPR-RS problem allows us improve to improve the time complexity of the TBR local search problem by a factor of $n$.

Intuitively, a (rooted) TBR operation may be viewed as being like an SPR operation except that the TBR operation allows the pruned subtree to be arbitrarily rerooted before being regrafted. In order to define a TBR operation more formally, we need the following definition.

**Definition 3.5.1** (RR operation)**.** *Let* $T$ *be a tree and* $x \in V(T)$. $\mathsf{RR}(T,x)$ *is defined to be the tree* $T$*, if* $x = rt(T)$*. Otherwise,* $\mathsf{RR}(T,x)$ *is the tree that is obtained from* $T$ *by (i) suppressing* $rt(T)$*, and (ii) subdividing the edge* $\{pa(x),x\}$ *by a new root node.*

**Definition 3.5.2** (TBR operation)**.** *For technical reasons we first define for a tree* $T$ *the* planted tree $\Phi(T)$ *that is the tree obtained by adding an additional edge, called* root edge*,* $\{p, rt(T)\}$ *to* $T$*.*

*Let* $T$ *be a tree,* $e = (u,v) \in E(T)$ *and* $X, Y$ *be the connected components that are obtained by removing edge* $e$ *from* $T$ *where* $v \in X$ *and* $u \in Y$*. We define* $\mathsf{TBR}_T(v,x,y)$ *for* $x \in X$ *and* $y \in Y$ *to be the tree that is obtained from* $\Phi(T)$ *by first removing edge* $e$*, then replacing the component* $X$ *by* $\mathsf{RR}(X,x)$*, and then adjoining a new edge* $f$ *between* $x' = rt(\mathsf{RR}(X,x))$ *and* $Y$ *as follows:*

1. *Create a new node $y'$ that subdivides the edge $(pa(y), y)$.*
2. *Adjoin the edge $f$ between nodes $x'$ and $y'$.*
3. *Suppress the node $u$, and rename $x'$ as $v$ and $y'$ as $u$.*
4. *Contract the root edge.*

**Notation.** We define the following:

1. $\mathsf{TBR}_T(v, x) = \bigcup_{y \in Y} \{\mathsf{TBR}_T(v, x, y)\}$
2. $\mathsf{TBR}_T(v) = \bigcup_{x \in X} \mathsf{TBR}_T(v, x)$
3. $\mathsf{TBR}_T = \bigcup_{(u,v) \in E(T)} \mathsf{TBR}_T(v)$

The TBR-***Scoring (*TBR-*S)*** and TBR-***Restricted Scoring (*TBR-*RS)*** problems can now be defined as follows.

**Problem 7** (TBR-Scoring (TBR-S))**.**

Instance: *A gene tree set $\mathcal{G}$, and a comparable species tree $S$.*

Find: *A tree $T^* \in \mathsf{TBR}_S$ such that $\Delta(\mathcal{G}, T^*) = \min_{T \in \mathsf{TBR}_S} \Delta(\mathcal{G}, T)$.*

**Problem 8** (TBR-Restricted Scoring (TBR-RS))**.**

Instance: *A triple $(\mathcal{G}, S, v)$, where $\mathcal{G}$ is a set of gene trees, $S$ is a comparable species tree, and $(u, v) \in E(S)$.*

Find: *A tree $T^* \in \mathsf{TBR}_S(v)$ such that $\Delta(\mathcal{G}, T^*) = \min_{T \in \mathsf{TBR}_S(v)} \Delta(\mathcal{G}, T)$.*

Our goal is to solve the TBR-S problem. Observe that there are $\Theta(n)$ different ways to select a subtree of $S$ to be pruned. Furthermore, there are $O(n)$ different ways to reroot the pruned subtree. The idea is to directly use the solution to the SPR-RS problem to compute the duplication and loss costs for the $O(n)$-cardinality subset of $\mathsf{TBR}_S$ defined by any fixed pruned subtree and its fixed rooting. In particular,

**Theorem 3.5.1.** *The TBR-S problem can be solved in $O(kmn^2)$ time.*

*Proof.* The algorithm presented in the previous section allows us to compute the loss cost of each tree in $\mathsf{TBR}_S(v, x)$ in $O(km)$ time. Thus, we can compute the loss cost of each tree in $\mathsf{TBR}_S$ with-in $O(kmn^2)$ time. Similarly, the algorithm presented in [4] allows us to compute the duplication cost of each tree in $\mathsf{TBR}_S$ with-in $O(kmn^2)$ time as well. This implies that we can obtain the reconciliation cost (i.e. the duplication cost plus the loss cost) for each tree in $\mathsf{TBR}_S$ in $O(kmn^2)$ time. Thus, the TBR-S problem can be solved in $O(kmn^2)$ time overall. □

The time complexity of the best known (naive) solution for the TBR-S problem is $O(kmn^3)$. Our algorithm improves on this by a factor of $n$.

## 3.6 Experimental Analysis

To evaluate the efficiency, in practice, of our novel local search algorithms, we conducted comparative studies on simulated datasets. In particular, we implemented our algorithm for the SPR-S problem as part of a standard search heuristic for the duplication-loss problem; we refer to our program as *DupLoss*. The two other publicly available programs for the duplication-loss problem, Mesquite [35] and GeneTree [38], both implement similar local search heuristics based on the best known (naive) algorithm for the SPR-S problem. Therefore, for our comparative study, we only compare the runtime of our implementation against the program GeneTree.

We applied the programs DupLoss and GeneTree to the same set of input gene trees and the same randomly generated starting species tree and measured the run time of both programs to compute their final species supertrees. The input gene trees for each run consisted of a set of 20 randomly generated gene trees, all with the same set of taxa.[4] We conducted five such runs, each with a different number of taxa (50, 100, 200, 400, and 1000) in the input trees. All analyses were performed on a 3 Ghz Intel Pentium 4 CPU based PC with Windows XP operating system. As shown in Table 3.1, DupLoss shows a great improvement in runtime and scalability as compared to GeneTree. We could not run GeneTree on input trees with more than 200 taxa.

Table 3.1 GeneTree vs. DupLoss

| Taxa size | GeneTree | DupLoss |
|---|---|---|
| 50 | 11m:42s | 5s |
| 100 | 3h:57m | 33s |
| 200 | 5d:19h:49m | 4m:24s |
| 400 | – | 43m:08s |
| 1000 | – | 19h:27m |

Note that both DupLoss and GeneTree implement exactly the same standard SPR based

[4]Our randomly generated trees have a random (binary) topology and a random assignment of leaf labels.

local search heuristic. However, the final reconciliation costs obtained by the two programs on the same input may still be different; this is because ties are broken arbitrarily if more than one optimal species tree is found during a local search step. In our experiments, we observed little or no difference in the final reconciliation costs.

## 3.7 Outlook and Conclusion

The duplication-loss problem has been an effective way to infer species phylogenies from paralogous data; and is expected to become even more relevant with the rapidly increasing availability of whole genome data. Our highly efficient algorithms for the standard SPR and TBR based heuristics make the duplication-loss problem much more tractable for large-scale phylogenetic analyses.

Bansal and Eulenstein [8] showed that the TBR local search problem for the gene duplication problem could be solved in $O(kmn \log m)$ time. It would be interesting to ascertain if the TBR local search problem for the duplication-loss problem can be solved in better than $\Theta(kmn^2)$ time as well.

# CHAPTER 4. Comparing Partially Resolved Trees

Modified from a paper to be submitted to Algorithmica

Mukul S. Bansal and David Fernández-Baca

## 4.1 Introduction

Evolutionary trees, also known as phylogenetic trees or phylogenies, represent the evolutionary history of sets of species. Such trees have uniquely labeled leaves, corresponding to the species, and unlabeled internal nodes, representing hypothetical ancestors. The trees can be either rooted, if the evolutionary origin is known, or unrooted, otherwise.

This paper addresses the following question: How does one measure how close two evolutionary trees are to each other? Among the motivations for this question is the growth of phylogenetic databases, such as TreeBase [41], with the attendant need for sophisticated querying mechanisms and for means to assess the quality of answers to queries. Another motivation arises from the fact that phylogenetic analyses — e.g., by parsimony [27] — typically produce multiple evolutionary trees (often in the thousands) for the same set of species.

We address this question by defining an appropriate *distance measure* between trees. While several such measures have been proposed before (see below), ours provides a feature that previous ones do not: The ability to deal elegantly with the presence of *unresolved* nodes, also called *polytomies*. For rooted trees these are nodes with more than two children; for unrooted trees, they are nodes of degree greater than three. Polytomies cannot simply be ignored, since they arise naturally in phylogenetic analysis. Furthermore, they must be treated with care: A node may be unresolved because it truly must be so or because there is not enough evidence to break it up into resolved nodes — that is, the polytomies are either "hard" or "soft" [34].

**Our contributions.** We define and analyze a new kind of distance measure for phylogenies. For rooted trees, our measure is based on the topologies the input trees induce on *triplets*; that is, on three-element subsets of the set of species. For unrooted trees, the measure is based on *quartets* (four-element subsets). Our approach is motivated by the observation that triplet and quartet topologies are the basic building blocks of rooted and unrooted trees, in the sense that they are the smallest topological units that completely identify a phylogenetic tree [44]. Triplet and quartet-based distances thus provide a robust and fine-grained measure of the differences and similarities between trees[1]. In contrast with traditional quartet and triplet distances, our distance measure deals cleanly with the presence of unresolved nodes.

The measure we propose is called *parametric distance*: Given a triplet (quartet) $X$, we compare the topologies that each of the two input trees induces on $X$. If they are identical, the contribution of $X$ to the distance is zero. If both topologies are fully resolved but different, then the contribution is one. Otherwise, the topology is resolved in one of the trees, but not the other. In this case, $X$ contributes $p$ to the distance, where $p$ is a real number between 0 and 1. Parameter $p$ allows one to make a smooth transition between hard and soft views of polytomy. At one extreme, if $p = 1$, an unresolved topology is viewed as different from a fully resolved one. At the other, when $p = 0$, unresolved topologies are viewed as identical to resolved ones. Intermediate values of $p$ allow one to adjust for the degree of certainty one has about a polytomy.

After defining our distance measure, we proceed to study its mathematical and algorithmic properties. We obtain exact and asymptotic bounds on expected values of parametric triplet distance and parametric quartet distance. We present a $O(n^2)$-time algorithm to compute parametric triplet distance and a $O(n^2)$ 2-approximate algorithm for parametric quartet distance. To our knowledge, there was no previous algorithm for computing the parametric triplet distance between two rooted trees, other than by enumerating all $\Theta(n^3)$ triplets. Two algorithms exist that can be directly applied to compute the parametric quartet distance (see also [14]). One runs in time $O(n^2 \min\{d_1, d_2\})$, where, for $i \in \{1, 2\}$, $d_i$ is the maximum degree

[1]Biologically-inspired arguments in favor of triplet-based measures can be found in [21].

of a node in $T_i$ [19]; the other takes $O(d^9 n \log n)$ time, where $d$ is the maximum degree of a node in $T_1$ and $T_2$ [50].[2] Our faster $O(n^2)$ algorithm offers a 2-approximate solution when an exact value of the parametric quartet distance is not required. Additionally, our algorithm gives the exact answer when $p = \frac{1}{2}$.

**Related work.** Several other measures for comparing trees have been proposed; we mention a few. A popular class of distances are those based on symmetric difference between sets of *clusters* (that is, on sets of species that descend from the same internal node in a rooted tree) or of *splits* (partitions of the set of species induced by the removal of an edge in an unrooted tree); the latter is the well-known Robinson-Foulds (RF) distance [42]. It is not hard to show that two rooted (unrooted) trees can share many triplet (quartet) topologies but not share a single cluster (split). Cluster- and split-based measures are also coarser than triplet and quartet distances.

One can also measure the distance between two trees by counting the number of *branch-swapping* operations — e.g., nearest-neighbor interchange or subtree pruning and regrafting operations [27] — needed to convert one of the trees into the other [1]. However, the associated measures can be hard to compute, and they fail to distinguish between operations that affect many species and those that affect only a few. An alternative to distance measures are *similarity* methods such as maximum agreement subtree (MAST) approach [28]. While there are efficient algorithms for computing the MAST [25], the measure is coarser than triplet-based distances.

## 4.2 Preliminaries

**Phylogenies.** By and large, we follow standard terminology (i.e., similar to [13] and [44]). We write $[N]$ to denote the set $\{1, 2, \ldots, N\}$, where $N$ is a positive integer.

Let $T$ be a rooted or unrooted tree. We write $\mathcal{V}(T)$, $\mathcal{E}(T)$, and $\mathcal{L}(T)$ to denote, respectively, the node set, edge set, and leaf set of $T$. A *taxon* (plural *taxa*) is some basic unit of classification;

[2] Note that the presence of unresolved nodes seems to complicate distance computation. Indeed, the quartet distance between a pair of *fully resolved* unrooted trees can be obtained in $O(n \log n)$ time [12].

e.g., a species. Let $S$ be a set of taxa. A *phylogenetic tree* or *phylogeny* for $S$ is a tree $T$ such that $\mathcal{L}(T) = S$. Furthermore, if $T$ is rooted, we require that every internal node have at least two children; if $T$ is unrooted, every internal node is required to have degree at least three. We write $RP(n)$ to denote the set of all rooted phylogenetic trees over $S = [n]$ and $P(n)$ to denote the set of all unrooted phylogenetic trees over $S = [n]$.

An internal node in a *rooted* phylogeny is *resolved* if it has exactly two children; otherwise it is *unresolved*. Similarly, an internal node in an *unrooted* phylogeny is *resolved* if it has degree three, and *unresolved* otherwise. Unresolved nodes in rooted and unrooted trees are also referred to as *polytomies* or *multifurcations*. A phylogeny (rooted or unrooted) is *fully resolved* if all its internal nodes are resolved.

Let $X$ be a subset of $\mathcal{L}(T)$ and let $T[X]$ denote the minimal subtree of $T$ having $X$ as its leaf set. The *restriction* of $T$ to $X$, denoted $T|X$, is the phylogeny for $X$ defined as follows. If $T$ is unrooted, then $T|X$ is the tree obtained from $T[X]$ by suppressing all degree-two nodes. If $T$ is rooted, $T|X$ is obtained from $T[X]$ by suppressing all degree-two nodes except for the root.

A *triplet* is a three-element subset of $S$. A *triplet tree* is a rooted phylogeny whose leaf set is a triplet. The triplet tree with leaf set $\{a, b, c\}$ is denoted by $a|bc$ if the path from $b$ to $c$ does not intersect the path from $a$ to the root. A *quartet* is a four-element subset of $S$ and a *quartet tree* is an unrooted phylogeny whose leaf set is a quartet. The quartet tree with leaf set $\{a, b, c, d\}$ is denoted by $ab|cd$ if the path from $a$ to $b$ does not intersect the path from $c$ to $d$. A triplet (quartet) $X$ is said to be *resolved* in a phylogenetic tree $T$ over $S$ if $T|X$ is fully resolved; otherwise, $X$ is *unresolved*.

Finally, we need some special notation for rooted trees $T$. We write $rt(T)$ to denote the root node of $T$. Let $v$ be a node in $T$. Then, $pa(v)$ denotes the parent of $v$ in $T$ and $Ch(v)$ is the set of children of $v$. Furthermore, $T(v)$ denotes the subtree of $T$ rooted at $v$ and $\overline{T(v)}$ denotes the tree obtained by deleting $T(v)$ from $T$, as well as the edge from $v$ to its parent, if such an edge exists.

## 4.3 Parametric distances

Let $T_1$ and $T_2$ be any two rooted (respectively, unrooted) phylogenies over the same taxon set $S$. Define the following five sets of triplets (quartets) over $S$.

1. $\mathcal{S}(T_1, T_2)$: triplets (quartets) $X$ such that $T_1|X$ and $T_2|X$ are fully resolved, and $T_1|X = T_2|X$.

2. $\mathcal{D}(T_1, T_2)$: triplets (quartets) $X$ such that $T_1|X$ and $T_2|X$ are fully resolved, and $T_1|X \neq T_2|X$.

3. $\mathcal{R}_1(T_1, T_2)$: triplets (quartets) $X$ such that $T_1|X$ is fully resolved, but $T_2|X$ is not.

4. $\mathcal{R}_2(T_1, T_2)$: triplets (quartets) $X$ such that $T_2|X$ is fully resolved, but $T_1|X$ is not.

5. $\mathcal{U}(T_1, T_2)$: triplets (quartets) $X$ such that $T_1|X$ and $T_2|X$ are unresolved.

Let $p$ be a real number in the interval $[0, 1]$. The *parametric triplet (quartet) distance between $T_1$ and $T_2$* is defined as[3]

$$d^{(p)}(T_1, T_2) = |\mathcal{D}(T_1, T_2)| + p\left(|\mathcal{R}_1(T_1, T_2)| + |\mathcal{R}_2(T_1, T_2)|\right). \tag{4.1}$$

When the domain of $d^{(p)}$ is restricted to fully resolved trees, and thus $\mathcal{R}_1(T_1, T_2) = \mathcal{R}_2(T_1, T_2) = \mathcal{U}(T_1, T_2) = \emptyset$, we refer to it simply as the *triplet (quartet) distance*.

Parameter $p$ allows one to make a smooth transition from soft to hard views of polytomy: When $p = 0$, resolved triplets (quartets) are treated as equal to unresolved ones, while when $p = 1$, they are treated as being completely different. Choosing intermediate values of $p$ allows one to adjust for the amount of evidence required to resolve a polytomy[4].

**Distance measures, metrics, and near-metrics.** A *distance measure* on a set $D$ is a binary function $d$ on $D$ satisfying the following three conditions: (i) $d(x, y) \geq 0$ for all $x, y \in D$; (ii) $d(x, y) = d(y, x)$ for all $x, y \in D$; and (iii) $d(x, y) = 0$ if and only if $x = y$. Function $d$

[3]Note that the sets $\mathcal{S}(T_1, T_2)$ and $\mathcal{U}(T_1, T_2)$ are not used in the definition of $d^{(p)}$, but are needed for other purposes.

[4]We note that parametric triplet/quartet distance is a *profile-based metric*, in the sense of [24]. However, the use of the word "profile" in [24] is quite different from our use of the term.

is a *metric* if, in addition to being a distance measure, it satisfies the triangle inequality; i.e., $d(x,z) \leq d(x,y) + d(y,z)$ for all $x, y, z \in D$. Distance measure $d$ is a *near-metric* if there is a constant $c$, independent of the size of $D$, such that $d$ satisfies the *relaxed polygonal inequality*: $d(x,z) \leq c(d(x,x_1) + d(x_1,x_2) + \cdots + d(x_{n-1},z))$ for all $n > 1$ and $x, z, x_1, \ldots, x_{n-1} \in D$ [24].

It is known [6] that (i) $d^{(p)}$ is a metric for $p \geq 1/2$, (ii) $d^{(p)}$ is a near-metric, but not a metric, for $0 < p < 1/2$, and (iii) $d^{(p)}$ is not a distance measure for $p = 0$.

## 4.4 Expected parametric triplet and quartet distances

We now consider the expected value of parametric triplet and quartet distances. Let $u(n)$ and $r(n)$ denote the probabilities that a given quartet is, respectively, unresolved or resolved in an unrooted phylogeny chosen uniformly at random from $P(n)$; thus, $u(n) = 1 - r(n)$. The following are the two main results of this section.

**Theorem 4.4.1.** *Let $T_1$ and $T_2$ be two unrooted phylogenies chosen uniformly at random with replacement from $P(n)$. Then,*

$$E(d^{(p)}(T_1,T_2)) = \binom{n}{4} \cdot \left( \frac{2}{3} \cdot r(n)^2 + 2 \cdot p \cdot r(n) \cdot u(n) \right). \tag{4.2}$$

**Theorem 4.4.2.** *Let $T_1$ and $T_2$ be two rooted phylogenies chosen uniformly at random with replacement from $RP(n)$. Then,*

$$E(d^{(p)}(T_1,T_2)) = \binom{n}{3} \cdot \left( \frac{2}{3} \cdot r(n+1)^2 + 2 \cdot p \cdot r(n+1) \cdot u(n+1) \right). \tag{4.3}$$

It is known [48, 47] that

$$u(n) \sim \sqrt{\frac{\pi(2\ln 2 - 1)}{4n}}. \tag{4.4}$$

Together with Theorems 4.4.1 and 4.4.2, this implies that $E(d^{(p)}(T_1,T_2))$ is asymptotically $\frac{2}{3} \cdot \binom{n}{4}$ for unrooted trees and $\frac{2}{3} \cdot \binom{n}{3}$ for rooted trees.

The proof of Theorem 4.4.1 follows directly from the work of Day [22]; hence, it is omitted (however, we should note that the proof is similar to that of Lemma 4.4.1 below).

The proof of Theorem 4.4.2, follows from two auxiliary results that will be proved in the next subsections. In the statements of these results, we use $u'(n)$ and $r'(n)$ to denote the

probabilities that a given triplet is, respectively, unresolved or resolved in an rooted phylogeny chosen at random from $RP(n)$.

**Lemma 4.4.1.** *Let $T_1$ and $T_2$ be two rooted phylogenies chosen uniformly at random with replacement from $RP(n)$. Then,*

$$E(d^{(p)}(T_1, T_2)) = \binom{n}{3} \cdot \left( \frac{2}{3} \cdot r'(n)^2 + 2 \cdot p \cdot r'(n) \cdot u'(n) \right). \tag{4.5}$$

**Lemma 4.4.2.** *For all $n \geq 1$, $r'(n) = r(n+1)$ and $u'(n) = u(n+1)$.*

*Proof of Theorem 4.4.2.* Simply substitute the expressions for $r'(n)$ and $u'(n)$ given in Lemma 4.4.2 into the expression for $E(d^{(p)}(T_1, T_2))$ given in Lemma 4.4.1. □

### 4.4.1 Proof of Lemma 4.4.1

By the definition of $d^{(p)}$ and the linearity of expectation, it suffices to establish the equalities below.

$$E(\mathcal{D}(T_1, T_2)) = \binom{n}{3} \cdot \frac{2}{3} \cdot r'(n)^2 \tag{4.6}$$

$$E(\mathcal{R}_1(T_1, T_2)) = E(\mathcal{R}_2(T_1, T_2)) = \binom{n}{3} \cdot r'(n) \cdot u'(n)) \tag{4.7}$$

To establish Equation (4.6), consider a triplet $X$. The probability that $X$ is resolved in $T_1$ (or $T_2$) is $r'(n)$. Thus, the probability that $X$ is resolved in both $T_1$ and $T_2$ is $r'(n)^2$. There are exactly three different ways in which any given triplet can be resolved. Hence, if $\alpha$ is resolved in both $T_1$ and $T_2$, the probability that it is resolved differently in both trees is $\frac{2}{3}$. Thus, the probability of a pre-given triplet being resolved in both $T_1$ and $T_2$, but with different types in each, is $\frac{2}{3}r'(n)^2$. By the linearity of expectation and the observation that the total number of triplets in $T_1$ (and $T_2$) is $\binom{n}{3}$, $E(\mathcal{D}(T_1, T_2)) = \binom{n}{3} \cdot \frac{2}{3}r'(n)^2$.

To establish Equation (4.7), we only need to study $E(\mathcal{R}_1(T_1, T_2))$; the expression for $E(\mathcal{R}_2(T_1, T_2))$ follows by symmetry. Consider a triplet $X$. The probability that $X$ is unresolved in $T_1$ is $u'(n)$ and the probability that $X$ is resolved in $T_2$ is $r'(n)$. The expression for $E(\mathcal{R}_1(T_1, T_2))$ now follows by linearity of expectation.

### 4.4.2 Proof of Lemma 4.4.2

We need some preliminary results. The following lemma is well known (see [48, 27]).

**Lemma 4.4.3.** *For all $n \geq 1$, $|RP(n)| = |P(n+1)|$.*

Let us define the function ADD-LEAF : $RP(n) \rightarrow P(n+1)$ as follows. Given a rooted tree $T \in RP(n)$, ADD-LEAF($T$) is the unrooted tree constructed from $T$ by (1) adding a leaf node labeled $n+1$ to $T$ by adjoining it to the root node of $T$ and (2) unrooting the resulting tree.

The next lemma is well known (see, e.g., [44, p. 20]).

**Lemma 4.4.4.** *Function* ADD-LEAF *is a bijection from the set $RP(n)$ to the set $P(n+1)$.*

For any triplet $X$ over $[n]$, we define two functions $g_X : RP(n) \rightarrow \{0,1\}$ and $f_X : P(n+1) \rightarrow \{0,1\}$ as follows:

$$g_X(T) = \begin{cases} 1 & \text{if triplet } X \text{ is resolved in tree } T \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

$$f_X(T) = \begin{cases} 1 & \text{if quartet } X \cup \{n+1\} \text{ is resolved in tree } T \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

We have the following result.

**Lemma 4.4.5.** *Let $X$ be any triplet over $[n]$. Consider a tree $T \in RP(n)$, and let $T' =$* ADD-LEAF($T$). *Then, $f_X(T') = g_X(T)$.*

*Proof.* Follows from the observation that triplet $X$ is resolved in $T$ if and only if quartet $X \cup \{n+1\}$ is resolved in $T'$. □

We are now ready to complete the proof of Lemma 4.4.2. Let $X$ be any triplet over $[n]$. By definition, $r(n+1)$ is the probability of any given quartet being resolved in a random unrooted tree in $P(n)$. In particular, $r(n+1)$ is the probability that quartet $X \cup \{n+1\}$ is resolved in

a random unrooted tree. Now,

$$\begin{aligned} r(n+1) &= \sum_{T \in P(n+1)} \frac{f_X(T)}{|P(n+1)|} \\ &= \sum_{T \in P(n+1)} \frac{f_X(T)}{|RP(n)|} \\ &= \sum_{T' \in RP(n)} \frac{g_X(T')}{|RP(n)|} \\ &= r'(n), \end{aligned}$$

where the first and last equalities follow from the definitions of $r(n+1)$ and $r(n)$, respectively, the second equality follows from Lemma 4.4.3, and the third follows from Lemma 4.4.4, and Lemma 4.4.5.

Since $u'(n) = 1 - r'(n)$ and $u(n+1) = 1 - r(n+1)$, it follows that $u'(n) = u(n+1)$.

## 4.5 Computing parametric triplet distance

In this section we show that the parametric triplet distance (PTD), $d^{(p)}$, between two phylogenetic trees $T_1$ and $T_2$ over the same set of $n$ taxa can be computed in $O(n^2)$ time.

Before we outline our PTD algorithm, we need some notation. Let $T$ be a rooted phylogenetic tree. Then, $R(T)$ denotes the set of all triplets that are resolved in $T$ and $U(T)$ denotes the set of all triplets that are unresolved in $T$.

The next proposition is easily proved.

**Proposition 4.5.1.** *For any two phylogenies $T_1$, $T_2$ over the same set of taxa,*

1. $|\mathcal{R}_1(T_1, T_2)| + |\mathcal{U}(T_1, T_2)| = |U(T_2)|$
2. $|\mathcal{R}_2(T_1, T_2)| + |\mathcal{U}(T_1, T_2)| = |U(T_1)|$,
3. $|\mathcal{S}(T_1, T_2)| + |\mathcal{D}(T_1, T_2)| + |\mathcal{R}_1(T_1, T_2)| = |R(T_1)|$.

By Prop. 4.5.1 and Eqn. (4.1), the parametric distance between $T_1$ and $T_2$ can be expressed as

$$d^{(p)}(T_1, T_2) = |R(T_1)| - |\mathcal{S}(T_1, T_2)| + p \cdot (|U(T_1)| - |U(T_2)|) + (2p-1) \cdot |\mathcal{R}_1(T_1, T_2)|. \quad (4.10)$$

Our PTD algorithm proceeds as follows. After an initial $O(n^2)$ preprocessing step (Section 4.5.1), the algorithm computes $|R(T_1)|$, $|U(T_1)|$ and $|U(T_2)|$ using a $O(n)$-time procedure (Section 4.5.2). Next, it computes $|\mathcal{S}(T_1, T_2)|$ and $|\mathcal{R}_1(T_1, T_2)|$. As described in Sections 4.5.3 and 4.5.4, this takes $O(n^2)$ time. Then, it uses these values to compute $d^{(p)}(T_1, T_2)$, in $O(1)$ time, via Eqn. (4.10). To summarize, we have the following result.

**Theorem 4.5.1.** *The parametric triplet distance $d^{(p)}(T_1, T_2)$ for two rooted phylogenetic trees $T_1$ and $T_2$ over the same set of $n$ taxa can be computed in $O(n^2)$ time.*

### 4.5.1 The preprocessing step

The purpose of the preprocessing step is to calculate and store the following four quantities for every pair $(u, v)$, where $u \in \mathcal{V}(T_1)$ and $v \in \mathcal{V}(T_2)$: $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$, $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(\overline{T_2(v)})|$, $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(T_2(v))|$, and $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$. These values are stored in a table so that any value can be accessed in $O(1)$ time by subsequent steps of the PTD algorithm.

**Lemma 4.5.1.** *The values $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$, $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(\overline{T_2(v)})|$, $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(T_2(v))|$, and $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$ can be collectively computed for every pair of nodes $(u, v)$, where $u \in \mathcal{V}(T_1)$ and $v \in \mathcal{V}(T_2)$, in $O(n^2)$ time.*

*Proof.* Consider the value $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$:

1. If $u$ and $v$ are both leaf nodes then computing $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$ is trivial,
2. If $u$ is a leaf node, but $v$ is not a leaf node, then we have $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))| = \sum_{x \in Ch(v)} |\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(x))|$, and,
3. If $u$ is not a leaf node then we have $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))| = \sum_{x \in Ch(u)} |\mathcal{L}(T_1(x)) \cap \mathcal{L}(T_2(v))|$.

We compute the value $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$, for every pair $(u, v)$, using an interleaved post order traversal of $T_1$ and $T_2$. This traversal works as follows: For each node $u$ in a post order traversal of $T_1$, we consider each node $v$ in a post order traversal of $T_2$. This ensures that when the intersection sizes for a pair of nodes is computed, the set intersection sizes for all pairs of their children have already been computed.

Once the value $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$ has been computed for some pair $(u, v)$, we must have

1. $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(\overline{T_2(v)})| = |\mathcal{L}(T_1(u))| - |\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$,

2. $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(T_2(v))| = |\mathcal{L}(T_2(v))| - |\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$, and

3. $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})| = n - (|\mathcal{L}(T_1(u))| + |\mathcal{L}(T_2(v))| - |\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|)$.

We now analyze the time complexity of computing these values. For each $u \in \mathcal{V}(T_1)$, the value $|\mathcal{L}(T_1(u))|$ can be computed in $O(n)$ time by a simple post order traversal of $T_1$. The same holds for tree $T_2$. For a pair of nodes $u$ and $v$ from $T_1$ and $T_2$ respectively, the value $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$ can be computed in $O(|\mathit{Ch}(u)| + |\mathit{Ch}(v)|)$ time and all the remaining three set intersection values in $O(1)$ time. Summing this over all possible pairs of edges, we get a total time complexity of $O(\sum_{u \in \mathcal{V}(T_1)} \sum_{v \in \mathcal{V}(T_2)} |\mathit{Ch}(u)| + |\mathit{Ch}(v)|)$, which is $O(n^2)$. □

We store these $O(n^2)$ values in an array indexed by $u$ and $v$, for each $u \in \mathcal{V}(T_1)$ and $v \in \mathcal{V}(T_2)$. This enables constant time insertion and look-up of any stored value, when the two relevant nodes are given.

### 4.5.2 Computing $|R(T_1)|$, $|U(T_1)|$ and $|U(T_2)|$

We use the following terminology. Let $e = (v, \mathit{pa}(v))$ be any internal edge in $T$. Consider any two leaves $x, y$ from $\mathcal{L}(T(v))$, and any leaf $z$ from $\mathcal{L}(\overline{T(v)})$. Then, the triplet $\{x, y, z\}$ must appear resolved as $xy|z$ in $T$; we say that the triplet tree $xy|z$ is *induced* by the edge $(v, \mathit{pa}(v))$. Note that the same resolved triplet tree may be induced by multiple edges in $T$. Additionally, if $x \in \mathcal{L}(T(v_1))$ and $y \in \mathcal{L}(T(v_2))$ for some $v_1, v_2 \in \mathit{Ch}(v)$ such that $v_1 \neq v_2$, then we say that the triplet tree $xy|z$ is *strictly induced* by the edge $(v, \mathit{pa}(v))$.

**Lemma 4.5.2.** *Given a tree $T$ and a triplet $X$, if $T|X$ is fully resolved then $T|X$ is strictly induced by exactly one edge in $T$.*

*Proof.* Let $X = \{a, b, c\}$. Without loss of generality, assume that $T|X = ab|c$. If $v$ denotes the lca of $a$ and $b$ in $T$, the edge $(v, \mathit{pa}(v))$ must induce $ab|c$. Moreover, $v$ must be the only node in $T$ for which there exist nodes $v_1, v_2 \in \mathit{Ch}(v)$ such that $a \in \mathcal{L}(T(v_1))$ and $b \in \mathcal{L}(T(v_2))$. Thus, there is exactly one edge in $T$ that strictly induces $T|X$. □

The following lemma shows how $|R(T_1)|$, $|U(T_1)|$ and $|U(T_2)|$ can all be computed in $O(n)$ time.

**Lemma 4.5.3.** *Given a rooted phylogenetic tree $T$ over $n$ leaves, the values $|R(T)|$ and $|U(T)|$ can be computed in $O(n)$ time.*

*Proof.* We compute the value $|R(T)|$ as follows: First, traverse the tree $T$ in post order to compute the values $\alpha_v = |\mathcal{L}(T(v))|$ and $\beta_v = n - \alpha_v$ at each node $v \in \mathcal{V}(T)$.

For any $v \in \mathcal{V}(T) \setminus \{rt(T)\}$, let $\phi(v)$ denote the number of triplets that are strictly induced by the edge $(pa(v), v)$ in tree $T$. Observe that any triplet that is strictly induced by an edge in $T$ must be fully resolved in $T$. Thus, Lemma 4.5.2 implies that the sum of $\phi(v)$ over all internal nodes $v \in \mathcal{V}(T) \setminus \{rt(T)\}$ yields the value $|R(T)|$. We now show how to compute the value of $\phi(v)$.

Let $X = \{a, b, c\}$ be a triplet that is counted in $\phi(v)$. And, without loss of generality, let $T_1|X = ab|c$. It can be verified that $X$ must satisfy the following two conditions: (i) $a, b \in \mathcal{L}(T(v))$ and $c \in \mathcal{L}(\overline{T(v)})$, and (ii) there does not exist any $x \in Ch(v)$ such that $a, b \in \mathcal{L}(T(x))$. The number of triplets that satisfy condition (i) is $\binom{\alpha_v}{2} \cdot \beta_v$, and the number of triplets that satisfy condition (i), but not condition (ii) is exactly $\sum_{x \in Ch(v)} \binom{\alpha_x}{2} \cdot \beta_v$. Thus, $\phi(v) = \gamma_v - \sum_{x \in Ch(v)} \binom{\alpha_x}{2} \cdot \beta_v$.

Computing $\phi(v)$ requires $O(|\ Ch(v)|)$ time; hence, the time complexity for computing $|R(T)|$ is $O(\sum_{v \in \mathcal{V}(T)} |\ Ch(v)|)$, which is $O(n)$.

Since $|R(T)|+|U(T)| = \binom{n}{3}$, the value $|U(T)|$ is easily computed in $O(1)$ additional time. □

### 4.5.3 Computing $|\mathcal{S}(T_1, T_2)|$

We now describe an $O(n^2)$ time algorithm to compute the size of the set $\mathcal{S}(T_1, T_2)$ of shared triplets; that is, triplets that are fully, and identically, resolved in $T_1$ and $T_2$.

For any $u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1))$ and $v \in \mathcal{V}(T_2) \setminus (rt(T_2) \cup \mathcal{L}(T_2))$, let $s(u, v)$ denote the number of identical triplet trees strictly induced by edge $(u, pa(u))$ in $T_1$ and edge $(v, pa(v))$ in $T_2$. Our algorithm uses the values computed in the preprocessing step to compute the number $s(u, v)$. We have the following result.

**Lemma 4.5.4.** *Given $T_1$ and $T_2$, we have,*

$$|\mathcal{S}(T_1, T_2)| = \sum_{\substack{u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1)), \\ v \in \mathcal{V}(T_2) \setminus (rt(T_2) \cup \mathcal{L}(T_2))}} s(u, v). \tag{4.11}$$

*Proof.* Consider any triplet $X \in \mathcal{S}(T_1, T_2)$. Since $T_1|X$ is fully resolved and $T_1|X = T_2|X$ then, by Lemma 4.5.2, there exists exactly one node $u \in \mathcal{V}(T_1) \setminus rt(T_1)$ and one node $v \in \mathcal{V}(T_2) \setminus rt(T_2)$ such that the edge $(u, pa(u))$ strictly induces $T_1|X$ in $T_1$, and edge $(v, pa(v))$ strictly induces $T_2|X$ in $T_2$. Additionally, neither $u$ nor $v$ can be leaf nodes in $T_1$ and $T_2$ respectively. Thus, $X$ would be counted exactly once in the right-hand side of Equation (4.11) in the value $s(u, v)$. Moreover, by the definition of $s(u, v)$, any triplet tree that is counted on the right-hand side of Equation (4.11) algorithm must belong to the set $\mathcal{S}(T_1, T_2)$. The Lemma follows. □

The following lemma shows how to compute the value of $s(u, v)$.

**Lemma 4.5.5.** *Given any $u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1))$ and $v \in \mathcal{V}(T_2) \setminus (rt(T_2) \cup \mathcal{L}(T_2))$, $s(u, v)$ can be computed in $O(|\, Ch(u)| \cdot |\, Ch(v)|)$ time.*

*Proof.* We will show that $s(u, v) = n_1(u, v) - n_2(u, v) - n_3(u, v) + n_4(u, v)$, where
$n_1(u, v) = \binom{|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|}{2} \cdot |\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$,
$n_2(u, v) = \sum_{x \in Ch(u)} \binom{|\mathcal{L}(T_1(x)) \cap \mathcal{L}(T_2(v))|}{2} \cdot |\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$,
$n_3(u, v) = \sum_{x \in Ch(v)} \binom{|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(x))|}{2} \cdot |\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$, and,
$n_4(u, v) = \sum_{x \in Ch(u)} \sum_{y \in Ch(v)} \binom{|\mathcal{L}(T_1(x)) \cap \mathcal{L}(T_2(y))|}{2} \cdot |\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$.

Consider any triplet tree, $ab|c$, counted in $s(u, v)$. It can be verified that $ab|c$ must satisfy the following three conditions: (i) $a, b \in \mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))$ and $c \in \mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})$, (ii) there does not exist any $x \in Ch(u)$ such that $a, b \in \mathcal{L}(T_1(x))$, and (iii) there does not exist any $x \in Ch(v)$ such that $a, b \in \mathcal{L}(T_2(x))$. Moreover, observe that any triplet tree $ab|c$ that satisfies these three conditions is counted in $s(u, v)$. Therefore, $s(u, v)$ is exactly the number of triplets trees that satisfy all three conditions (i), (ii) and (iii).

The number of triplet trees that satisfy condition (i) is given by $n_1(u, v)$. Some of the triplet trees that satisfy condition (i) may not satisfy conditions (ii) or (iii); these must not

**Procedure:** $\mathcal{S}(T_1, T_2)$
1: **for** each internal node $u \in \mathcal{V}(T_1) \setminus rt(T_1)$ **do**
2: **for** each internal node $v \in \mathcal{V}(T_2) \setminus rt(T_2)$ **do**
3: Compute $s(u, v)$.
4: **return** the sum of all computed $s(\cdot, \cdot)$.

Figure 4.1 Computing $|\mathcal{S}(T_1, T_2)|$

be counted in $s(u, v)$. The value $n_2(u, v)$ is exactly the number of triplet trees that satisfy condition (i) but not condition (ii). Similarly, $n_3(u, v)$ is exactly the number of triplet trees that satisfy condition (i) but not (iii). Thus, the second and third terms must be subtracted from the first term. However, there may be triplet trees that satisfy condition (i) but neither (ii) nor (iii), and, consequently, get subtracted in both the second and third terms. In order to adjust for these, the value $n_4(u, v)$ counts exactly those triplet trees that satisfy condition (i) but not (ii) and (iii). □

A summary of our algorithm to compute $|\mathcal{S}(T_1, T_2)|$ appears in Figure 4.1.

**Lemma 4.5.6.** *Given two rooted phylogenetic trees $T_1$ and $T_2$ on the same $n$ leaves, the value $|\mathcal{S}(T_1, T_2)|$ can be computed in $O(n^2)$ time.*

*Proof.* By Lemma 4.5.4, the algorithm of Figure 4.1 computes the value $|\mathcal{S}(T_1, T_2)|$ correctly. We now analyze its complexity. The running time of the algorithm is dominated by the complexity of computing the value $s(u, v)$ for each pair of internal nodes $u \in \mathcal{V}(T_1)$ and $v \in \mathcal{V}(T_2)$. According to Lemma 4.5.5, the value $s(u, v)$ can be computed in $O(|\,Ch(u)| \cdot |\,Ch(v)|)$ time. Thus, the total time complexity of the algorithm is $O(\sum_{u \in \mathcal{V}(T_1)} \sum_{v \in \mathcal{V}(T_2)} |\,Ch(u)| \cdot |\,Ch(v)|)$, which is $O(n^2)$. □

### 4.5.4 Computing $|\mathcal{R}_1(T_1, T_2)|$

Next, we describe an $O(n^2)$-time algorithm that computes the cardinality of the set $\mathcal{R}_1(T_1, T_2)$ of triplets that are resolved only in tree $T_1$. First, we need a definition. Let $X$ be a triplet that is unresolved in $T_2$. Let $v$ be the least common ancestor (lca) of $X$ in $T_2$. We say that $X$

is *associated* with $v$. Observe that node $v$ must be internal and unresolved. Note also that $X$ is associated with exactly one node in $T_2$.

For any $u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1))$ and $v \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_1))$, let $r_1(u, v)$ denote the number of triplets $X$ such that $T_1|X$ is strictly induced by edge $(u, pa(u))$ in $T_1$, and $X$ is associated with the node $v$ in $T_2$.

The triplets counted in $r_1(u, v)$ must be resolved in $T_1$ but unresolved in $T_2$. Our algorithm computes the value $|\mathcal{R}_1(T_1, T_2)|$ by computing, for each $u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1))$ and $v \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)$, the value $r_1(u, v)$. We claim that the sum of all the computed $r_1(u, v)$'s yields the value $|\mathcal{R}_1(T_1, T_2)|$.

**Lemma 4.5.7.** *Given $T_1$ and $T_2$, we must have,*

$$|\mathcal{R}(T_1, T_2)| = \sum_{\substack{u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1)), \\ v \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1(u, v). \tag{4.12}$$

*Proof.* Consider any triplet $X \in \mathcal{R}_1(T_1, T_2)$. By Lemma 4.5.2, there exists exactly one node $u \in \mathcal{V}(T_1) \setminus rt(T_1)$ such that the edge $(u, pa(u))$ strictly induces $T_1|X$ in $T_1$. Also observe that there must be exactly one unresolved node $v \in \mathcal{V}(T_2)$ with which $X$ is associated. Additionally, neither $u$ nor $v$ can be leaf nodes in $T_1$ and $T_2$ respectively. Thus, $X$ would be counted exactly once in the right-hand side of Equation (4.12); in the value $r_1(u, v)$. Moreover, by the definition of $r_1(u, v)$, any triplet that is counted in the right-hand side of Equation (4.12) must belong to the set $\mathcal{R}_1(T_1, T_2)$. The lemma follows. □

Given a path $u_1, u_2, \ldots, u_k$, where $k \geq 2$, in tree $T_1$ such that $u_k$ is an internal node and $u_1$ is an ancestor of $u_k$, let $\gamma(u_1, u_k, v)$ denote the number of triplets $X$ such that $T_1|X$ is induced by every edge $(u_{i-1}, u_i)$, for $2 \leq i \leq k$, in $T_1$ and $X$ is associated with node $v$ in $T_2$.

The following lemma shows how the value of $r_1(u, v)$ can be computed by first computing certain $\gamma(\cdot, \cdot, \cdot)$ values.

**Lemma 4.5.8.** *For any $u \in \mathcal{V}(T_1) \setminus (rt(T_1) \cup \mathcal{L}(T_1))$ and $v \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2))$,*

$$r_1(u, v) = \gamma(pa(u), u, v) - \sum_{x \in Ch(u)} \gamma(pa(u), x, v).$$

*Proof.* Let $X = \{a, b, c\}$ be a triplet that is counted in $r_1(u, v)$. And, without loss of generality, let $T_1|X = ab|c$. It can be verified that $X$ must satisfy the following three conditions: (i) $X$ must be associated with $v$ in $T_2$, (ii) $a, b \in \mathcal{L}(T_1(u))$ and $c \in \mathcal{L}(\overline{T_1(u)})$, and (iii) there must not exist any $x \in Ch(u)$ such that $a, b \in \mathcal{L}(T_1(x))$. Moreover, observe that if there exists a triplet $X = \{a, b, c\}$ that satisfies these three conditions, then $X$ will be counted in $r_1(u, v)$; these three conditions are thus necessary and sufficient.

Now observe that $\gamma(pa(u), u, v)$ counts exactly those triplets that satisfy conditions (i) and (ii), while $\sum_{x \in Ch(u)} \gamma(pa(u), x, v)$ counts exactly those triplets that satisfy conditions (i) and (ii), but not condition (iii). The lemma follows immediately. □

To compute the value of $\gamma(\cdot, \cdot, \cdot)$ efficiently we rely on the following lemma.

**Lemma 4.5.9.** *Consider a path $u_1, u_2, \ldots, u_k$, where $k \geq 2$, in tree $T_1$ such that $u_k$ is an internal node and $u_1$ is an ancestor of $u_k$. And let $v \in \mathcal{V}(T_2)$ be an internal unresolved node. Then,*

$$\gamma(u_1, u_k, v) = n_1(u_1, u_k, v) - n_2(u_1, u_k, v) - n_3(u_1, u_k, v) - n_4(u_1, u_k, v), \text{ where}$$

$$n_1(u_1, u_k, v) = \binom{|\mathcal{L}(T_2(v)) \cap \mathcal{L}(T_1(u_k))|}{2} \cdot |\mathcal{L}(T_2(v)) \cap \mathcal{L}(\overline{T_1(u_2)})|,$$

$$n_2(u_1, u_k, v) = \sum_{x \in Ch(v)} \binom{|\mathcal{L}(T_2(x)) \cap \mathcal{L}(T_1(u_k))|}{2} \cdot |\mathcal{L}(T_2(x)) \cap \mathcal{L}(\overline{T_1(u_2)})|,$$

$$n_3(u_1, u_k, v) = \sum_{x \in Ch(v)} \binom{|\mathcal{L}(T_1(u_k)) \cap \mathcal{L}(T_2(x))|}{2} \cdot \Big( |\mathcal{L}(T_2(v)) \cap \mathcal{L}(\overline{T_1(u_2)})| - |\mathcal{L}(T_2(x)) \cap \mathcal{L}(\overline{T_1(u_2)})| \Big),$$

*and*

$$n_4(u_1, u_k, v) = \sum_{x \in Ch(v)} |\mathcal{L}(T_2(x)) \cap \mathcal{L}(T_1(u_k))| \cdot |\mathcal{L}(T_2(x)) \cap \mathcal{L}(\overline{T_1(u_2)})| \cdot \big( |\mathcal{L}(T_2(v)) \cap \mathcal{L}(T_1(u_k))| - |\mathcal{L}(T_2(x)) \cap \mathcal{L}(T_1(u_k))| \big).$$

*Proof.* Consider those triplets $X$ for which $T_1|X$ is induced by every edge $(u_{i-1}, u_i)$, for $2 \leq i \leq k$, in $T_1$, and $T_2|X$ is a subtree of $T_2(v)$. Let us call these triplets *relevant*. Any relevant triplet must have all three leaves from $\mathcal{L}(T_2(v))$, two leaves from $\mathcal{L}(T_1(u_k))$, and the third leaf from $\mathcal{L}(\overline{T_1(u_2)})$. Also note that any triplet that satisfies these three conditions must be relevant. The number of triplets that satisfy these conditions is exactly $n_1(u_1, u_k, v)$.

Any relevant triplet $X$ must belong to one of the following four categories:

1. *The lca of $X$ in $T_2$ is not node $v$* : This implies that, in addition to being a relevant

**Procedure:** $\mathcal{R}_1(T_1, T_2)$
1: **for** each internal node $u \in \mathcal{V}(T_1) \setminus \{rt(T_1)\}$ **do**
2: **for** each internal unresolved node $v \in \mathcal{V}(T_2)$ **do**
3: Compute $r_1(u, v)$.
4: **return** the sum of all computed $r_1(\cdot, \cdot)$.

Figure 4.2 Computing $|\mathcal{R}_1(T_1, T_2)|$

triplet, all three leaves of $X$ must belong to the same subtree of $T_2$ rooted at a child of $v$. The number of such triplets is thus simply $n_2(u_1, u_k, v)$.

2. *The lca of $X$ in $T_2$ is node $v$, $X$ is resolved in $T_2$ and $T_1|X = T_2|X$* : A relevant triplet $X$ satisfies this criterion if and only if there exists a child $x \in Ch(v)$, such that the two leaves of this triplet that belong to $\mathcal{L}(T_1(u_k))$ in tree $T_1$ also occur in $\mathcal{L}(T_2(x))$, and, the third leaf (which occurs in $\mathcal{L}(\overline{T_1(u_2)})|$ in $T_1$) occurs in $\mathcal{L}(T_2(y))$ where $y \in Ch(v) \setminus \{x\}$. The number of such $X$ is thus equal to $n_3(u_1, u_k, v)$.

3. *The lca of $X$ in $T_2$ is node $v$, $X$ is resolved in $T_2$, but $T_1|X \neq T_2|X$* : A relevant triplet $X$ satisfies this criterion if and only if there exists a child $x \in Ch(v)$, such that a pair of the leaves of $X$ that occur in $\mathcal{L}(T_1(u_k))$ and $\mathcal{L}(\overline{T_1(u_2)})$ respectively in tree $T_1$ occur in $\mathcal{L}(T_2(x))$ in tree $T_2$, and, the third leaf (which occurs in $\mathcal{L}(T_2(x))$ in $T_1$) occurs in $\mathcal{L}(T_2(y))$ where $y \in Ch(v) \setminus \{x\}$. The number of such $X$ is thus given by $n_4(u_1, u_k, v)$.

4. *The lca of $X$ in $T_2$ is node $v$, and $X$ is unresolved in $T_2$* : By definition, the number of relevant triplets that satisfy this criterion is exactly $\gamma(u_1, u_k, v)$.

We have shown that $n_2(u_1, u_k, v)$, $n_3(u_1, u_k, v)$, and $n_4(u_1, u_k, v)$ are exactly the number of relevant triplets belonging to categories 1, 2, and 3 respectively. The lemma follows. □

**Remark.** We do not compute the quantity $\gamma(u_1, u_k, v)$ directly because doing so seems to require higher time complexity than our indirect method.

A summary of our algorithm for computing $|\mathcal{R}_1(T_1, T_2)|$ appears in Figure 4.2.

**Lemma 4.5.10.** *Given two phylogenetic trees $T_1$ and $T_2$ on the same $n$ leaves, the value $|\mathcal{R}_1(T_1, T_2)|$ can be computed in $O(n^2)$ time.*

*Proof.* By Lemma 4.5.7, the algorithm of Figure 4.2 computes the value $|\mathcal{R}_1(T_1, T_2)|$ correctly. We now analyze its complexity. For any given candidate nodes $u, v$, Lemma 4.5.9 shows how to compute $\gamma(\cdot, \cdot, v)$ in $O(Ch(v))$ time, and consequently, by Lemma 4.5.8, the value $r_1(u, v)$ can be computed in $O(|\ Ch(u)| \cdot |\ Ch(v)|)$ time. Thus, the total time complexity of the algorithm is $O(\sum_{u \in \mathcal{V}(T_1)} \sum_{v \in \mathcal{V}(T_2)} |\ Ch(u)| \cdot |\ Ch(v)|)$, which is $O(n^2)$. □

## 4.6 An approximation algorithm for parametric quartet distance

We now consider the problem of computing the parametric quartet distance (PQD) between two unrooted trees. Our main result is an $O(n^2)$-time 2-approximate algorithm for PQD.

Our approach is similar to the one for computing the parametric triplet distance. Observe that Proposition 4.5.1 and, thus, Equation (4.10) hold even when the unit of distance is quartets instead of triplets. Christiansen et al. [19] show how to compute the values $|\mathcal{S}(T_1, T_2)|$, $|R(T_1)|$, $|U(T_1)|$, and $|U(T_2)|$ within $O(n^2)$ time. In Section 4.6.1 we show how to compute, in $O(n^2)$ time, a value $y$ such that $|\mathcal{R}_1(T_1, T_2)| \leq y \leq 2|\mathcal{R}_1(T_1, T_2)|$. Now, let us substitute the values of $|R(T_1)|$, $|U(T_1)|$, $|U(T_2)|$ and $|\mathcal{S}(T_1, T_2)|$ into Equation (4.10), and use the value of $y$ instead of $|\mathcal{R}_1(T_1, T_2)|$. Assuming $p \geq 1/2$, it can be seen that the result is a 2-approximation to $d^{(p)}(T_1, T_2)$.

To summarize, we have the following result.

**Theorem 4.6.1.** *Given two unrooted phylogenetic trees $T_1$ and $T_2$ on the same $n$ leaves, and a parameter $p \geq 1/2$, a value $x$ such that $d^{(p)}(T_1, T_2) \leq x \leq 2 \cdot d^{(p)}(T_1, T_2)$ can be computed in $O(n^2)$ time.*

We note that the $(2p - 1) \cdot |\mathcal{R}_1(T_1, T_2)|$ term in Eqn. (4.10) vanishes when $p = \frac{1}{2}$. In this case, we don't even need to compute $|\mathcal{R}_1(T_1, T_2)|$ to get the exact value of $d^{(p)}(T_1, T_2)$.

### 4.6.1 Computing a 2-approximate value of $|\mathcal{R}_1(T_1, T_2)|$

Let $(u, v)$ be an edge in tree $T$. Removal of this edge splits the tree $T$ into two subtrees. We denote the subtree that contains node $u$ by $T(\overleftarrow{u, v})$, and the other subtree by $T(\overleftarrow{v, u})$. Also, we define $adj(u)$ to be the set of nodes that are adjacent to $u$. An (undirected) edge $(u, v) \in \mathcal{E}(T)$

can be viewed as two directed edges $(\overleftarrow{u,v})$ and $(\overleftarrow{v,u})$. Let $\overleftarrow{\mathcal{E}}(T)$ denote the set of directed edges in tree $T$. Though our trees do not really contain any directed edges, the idea of viewing an undirected edge as two directed ones will be useful for stating our algorithm and proving its correctness.

To achieve the claimed time complexity, our algorithm relies on a preprocessing step which computes and stores, for each pair of directed edges $(\overleftarrow{u_1,v_1}) \in \overleftarrow{\mathcal{E}}(T_1)$ and $(\overleftarrow{u_2,v_2}) \in \overleftarrow{\mathcal{E}}(T_2)$, the quantity $|\mathcal{L}(T_1(\overleftarrow{u_1,v_1})) \cap \mathcal{L}(T_2(\overleftarrow{u_2,v_2}))|$. This can be accomplished in $O(n^2)$ by arbitrarily rooting $T_1$ and $T_2$ at any internal node and proceeding as in the preprocessing step for the triplet distance case (see Section 4.5.1).

Consider any two leaves $a, b$ from $\mathcal{L}(T(\overleftarrow{u,v}))$, and any two leaves $c, d$ from $\mathcal{L}(T(\overleftarrow{v,u}))$. Then, the quartet $\{a, b, c, d\}$ must appear resolved as $ab|cd$ in $T$; we say that the quartet tree $ab|cd$ is *induced* by the edge $(u, v)$. Note that the same resolved quartet tree may be induced by multiple edges in $T$. Additionally, if $x \in u_1$ and $y \in u_2$ for some $u_1, u_2 \in adj(u) \setminus \{v\}$ such that $u_1 \neq u_2$, then we say that the quartet tree $ab|cd$ is *strictly induced* by the directed edge $(\overleftarrow{u,v})$.

Consider a quartet $\{a, b, c, d\}$. Then, the corresponding quartet tree is unresolved in $T$ if and only if there exists exactly one node $w$ such that the paths from $w$ to $a$, $w$ to $b$, $w$ to $c$, and $w$ to $d$ do not share any edges. We say that quartet $\{a, b, c, d\}$ is *associated* with node $w$ in $T$. Thus, each unresolved quartet tree from $T$ is associated with exactly one node in $T$.

For any directed edge $(\overleftarrow{u,v}) \in \overleftarrow{\mathcal{E}}(T_1)$ and $w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_1)$, let $r_1((\overleftarrow{u,v}), w)$ denote the number of quartets $X$ such that $T_1|X$ is strictly induced by the directed edge $(\overleftarrow{u,v})$ in $T_1$, and $X$ is associated with the node $w$ in $T_2$.

The quartets counted in $r_1((\overleftarrow{u,v}), w)$ must be resolved in $T_1$ but unresolved in $T_2$. We have the following result.

**Lemma 4.6.1.** *Given $T_1$ and $T_2$, we have,*

$$2 \cdot |\mathcal{R}_1(T_1, T_2)| = \sum_{\substack{(\overleftarrow{u,v}) \in \overleftarrow{\mathcal{E}}(T_1), \\ w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1((\overleftarrow{u,v}), w).$$

*Proof.* Let $X = \{a, b, c, d\}$ be any quartet in $|\mathcal{R}_1(T_1, T_2)|$. Without loss of generality, assume that $T_1|X = ab|cd$, and that $X$ is associated with node $w \in V(T_2) \setminus \mathcal{L}(T_2)$. Since $X$ appears

resolved in $T_1$, $\overleftarrow{\mathcal{E}}(T_1)$ must have exactly two directed edges, say $(\overleftarrow{u_1, v_1})$ and $(\overleftarrow{u_2, v_2})$, which strictly induce $ab|cd$. Thus, $X$ is counted in exactly two of the $r_1(\cdot,\cdot)$'s, namely, $r_1((\overleftarrow{u_1, v_1}), w)$, and $r_1((\overleftarrow{u_2, v_2}), w)$. The lemma follows. □

Thus, we can compute $|\mathcal{R}_1(T_1, T_2)|$ by computing all the $O(n^2)$, $r_1((\overleftarrow{u, v}), w)$'s. However, doing so seems to require at least $\Theta(n^2 \cdot d)$ time, where $d$ is the degree of $T_1$. Instead, our algorithm computes a 2-approximate value of $|\mathcal{R}_1(T_1, T_2)|$ in $O(n^2)$ time. For this we rely on the next lemma.

**Lemma 4.6.2.** *Given $T_1$ and $T_2$, let $T_1'$ denote the rooted tree obtained from $T_1$ by designating any internal node in $V(T_1)$ as the root. Then,*

$$|\mathcal{R}_1(T_1, T_2)| \leq \sum_{\substack{u \in \mathcal{V}(T_1') \setminus (rt(T_1') \cup \mathcal{L}(T_1')), \\ w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1((\overleftarrow{u, pa(u)}), w) \leq 2 \cdot |\mathcal{R}_1(T_1, T_2)|.$$

*Proof.* First, observe that if $u \in \mathcal{L}(T_1')$ and $w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)$, then $r_1((\overleftarrow{u, pa(u)}), w) = 0$. Therefore, we must have

$$\sum_{\substack{u \in \mathcal{V}(T_1') \setminus (rt(T_1') \cup \mathcal{L}(T_1')), \\ w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1((\overleftarrow{u, pa(u)}), w) = \sum_{\substack{u \in \mathcal{V}(T_1') \setminus rt(T_1'), \\ w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1((\overleftarrow{u, pa(u)}), w).$$

Second, observe that $\mathcal{E}(T_1) = \mathcal{E}(T_1')$ and, therefore, by Lemma 4.6.1, we must have

$$\sum_{\substack{u \in \mathcal{V}(T_1') \setminus rt(T_1'), \\ w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)}} r_1((\overleftarrow{u, pa(u)}), w) \leq 2 \cdot |\mathcal{R}_1(T_1, T_2)|.$$

This proves the second inequality in the lemma.

To complete the proof, we now prove the first inequality. Let $X = \{a, b, c, d\}$ be any quartet in $|\mathcal{R}_1(T_1, T_2)|$, and, without loss of generality, assume that $T_1|X = ab|cd$, and that $X$ is associated with node $w \in V(T_2) \setminus \mathcal{L}(T_2)$. Since $X$ appears resolved in $T_1$, $\overleftarrow{\mathcal{E}}(T_1)$ must have exactly two directed edges, say $(\overleftarrow{u_1, v_1})$ and $(\overleftarrow{u_2, v_2})$, which strictly induce $ab|cd$. Consider the edge $(u_1, v_1) \in \mathcal{E}(T_1')$. There are two possible cases: Either $v_1 = pa(u_1)$, or $u_1 = pa(v_1)$. If $v_1 = pa(u_1)$ then the quartet $X$ will be counted in the value $r_1((\overleftarrow{u_1, pa(u_1)}), w)$. Otherwise, if $u_1 = pa(v_1)$, then $u_1, v_1, v_2, u_2$ must appear on a same root-to-leaf path in $T_1'$. Consequently,

we must have $v_2 = pa(u_2)$ and the quartet $X$ would be counted in the value $r_1((\overleftarrow{u_2, pa(u_1)}), w)$. Thus, we must have $|\mathcal{R}_1(T_1, T_2)| \leq \sum_{u \in \mathcal{V}(T_1') \setminus rt(T_1')} \sum_{w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2)} r_1((\overleftarrow{u, pa(u)}), w)$. The lemma follows. □

Thus, the idea for efficiently computing 2-approximate value of $|\mathcal{R}_1(T_1, T_2)|$ is to first root $T_1$ arbitrarily at any internal node and then compute the value $r_1((\overleftarrow{u, pa(u)}), w)$ for each non-root node $u \in V(T_1)$ and each $w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_1)$.

We now direct our attention to the problem of efficiently computing all the required values $r_1(\cdot, \cdot)$. Given a path $u_1, u_2, \ldots, u_k$ in $T_1$, where $k \geq 2$, let $\gamma(u_1, u_k, w)$ denote the number of quartets $X$ such that $T_1|X$ is induced in $T_1$ by every edge $(u_{i-1}, u_i)$, $2 \leq i \leq k$, and $X$ is associated with node $w$ in $T_2$.

The following lemma is analogous to Lemma 4.5.8, and shows how the value $r_1(\cdot, \cdot)$ can be computed by first computing certain $\gamma(\cdot, \cdot, \cdot)$ values.

**Lemma 4.6.3.** *Let $(u, v) \in \mathcal{E}(T_1)$, and $w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2))$, then,*

$$r_1((\overleftarrow{u, v}), w) = \gamma(u, v, w) - \sum_{x \in adj(u) \setminus \{v\}} \gamma(x, v, w).$$

*Proof.* Let $X = \{a, b, c, d\}$ be a quartet that is counted in $r_1((\overleftarrow{u, v}), w)$. And, without loss of generality, let $T_1|X = ab|cd$ such that $a, b \in \mathcal{L}(T_1(\overleftarrow{u, v}))$. It can be verified that $X$ must satisfy the following three conditions: (i) $X$ must be associated with node $w$ in $T_2$, (ii) $a, b \in \mathcal{L}(T_1(\overleftarrow{u, v}))$ and $c, d \in \mathcal{L}(T_1(\overleftarrow{v, u}))$, and (iii) there must not exist any $x \in adj(u) \setminus \{v\}$ such that $a, b \in \mathcal{L}(T_1(\overleftarrow{x, u}))$. Moreover, observe that if there exists a quartet $X = \{a, b, c, d\}$ that satisfies these three conditions, then $X$ will be counted in $r_1((\overleftarrow{u, v}), w)$; these three conditions are thus necessary and sufficient.

Now observe that $\gamma(u, v, v)$ counts exactly all those quartets that satisfy conditions (i) and (ii), while $\sum_{x \in Ch(u)} \gamma(pa(u), x, v)$ counts exactly all those quartets that satisfy conditions (i) and (ii), but not condition (iii). The lemma follows. □

To state our next results we need the following notation. Given phylogenetic trees $T_1$ and $T_2$, consider a path $u_1, u_2, \ldots, u_k$ where $k \geq 2$, in tree $T_1$, and an internal node $w \in \mathcal{V}(T_2)$ of

degree at least 4. Let $P = \mathcal{L}(T_1(\overleftarrow{u_1, u_2}))$, $Q = \mathcal{L}(T_1(\overleftarrow{u_k, u_{k-1}})$ and let $x_1, \ldots, x_{|\ adj(w)|}$ denote the neighbors of $w$. Consider the quartets that are induced by every edge $(u_{i-1}, u_i)$, $2 \leq i \leq k$, in $T_1$: Let us call these quartets *relevant*. Observe that a quartet is relevant if and only if it contains exactly two leaves from $P$ and two leaves from $Q$. Let

1. $n_1(u_1, u_k, w)$ denote the number of relevant quartets $X$ for which there exists a neighbor $x$ of $w$ in tree $T_2$, such that $X$ is completely contained in $T_2(\overleftarrow{x, w})$,

2. $n_2(u_1, u_k, w)$ denote the number of relevant quartets $X$ for which there exist two neighbors $x, y$ of $w$ in tree $T_2$, such that $T_2(\overleftarrow{x, w})$ contains three leaves from $X$ and $T_2(\overleftarrow{y, w})$ contains the other leaf,

3. $n_3(u_1, u_k, w)$ denote the number of relevant quartets $X$ for which there exist two neighbors $x, y$ of $w$ in tree $T_2$, such that $T_2(\overleftarrow{x, w})$ contains two leaves from $X$ and $T_2(\overleftarrow{y, w})$ contains the other two leaves, and

4. $n_4(u_1, u_k, w)$ denote the number of relevant quartets $X$ for which there exist three neighbors $x, y, z$ of $w$ in tree $T_2$, such that $T_2(\overleftarrow{x, w})$ contains two leaves from $X$, $T_2(\overleftarrow{y, w})$ contains one leaf from $X$, and $T_2(\overleftarrow{z, w})$ contains the remaining leaf.

Then, we must have the following.

**Lemma 4.6.4.**

$$\gamma(u_1, u_k, w) = \binom{|P|}{2} \cdot \binom{|Q|}{2} - n_1(u_1, u_k, w) - n_2(u_1, u_k, w) - n_3(u_1, u_k, w) - n_4(u_1, u_k, w). \tag{4.13}$$

*Proof.* The term $\binom{|P|}{2} \cdot \binom{|Q|}{2}$ is the number of relevant quartets. Furthermore, each relevant quartet must occur in tree $T_2$ in exactly one of the five configurations captured by the terms $n_1(u_1, u_k, w)$, $n_2(u_1, u_k, w)$, $n_3(u_1, u_k, w)$, $n_4(u_1, u_k, w)$, and $\gamma(u_1, u_k, w)$. The lemma follows. □

The following four lemmas deal with the computation of the values $n_1(u_1, u_k, w)$, $n_2(u_1, u_k, w)$, $n_3(u_1, u_k, w)$, and $n_4(u_1, u_k, w)$.

**Lemma 4.6.5.** *The value $n_1(u_1, u_k, w)$ can be computed in $O(|\, adj(w)|)$ time.*

*Proof.* We will show that

$$n_1(u_1, u_k, w) = \sum_{i=1}^{|\, adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P|}{2} \cdot \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|}{2}. \tag{4.14}$$

The right hand side of Equation (4.14) counts all those quartets which are completely contained in $\mathcal{L}(T_2(\overleftarrow{x, w}))$ for some $x \in adj(w)$ and which have two elements from $P$ and two from $Q$. These are exactly the quartets that must be counted in $n_1(u_1, u_k, w)$. □

**Lemma 4.6.6.** *The value $n_2(u_1, u_k, w)$ can be computed in $O(|\, adj(w)|)$ time.*

*Proof.* We will show that

$$\begin{aligned} n_2(u_1, u_k, w) &= \sum_{i=1}^{|\, adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P|}{2} \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q| \cdot |\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap Q| \quad (4.15) \\ &+ \sum_{i=1}^{|\, adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|}{2} \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P| \cdot |\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap P|. \end{aligned}$$

The quartets $X$ counted in $n_2(u_1, u_k, w)$ are exactly those quartets for which there exist two neighbors $x, y$ of $w$ such that either (i) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ contains two leaves from $P$ and one from $Q$, and $X \cap \mathcal{L}(T_2(\overleftarrow{y, w}))$ contains a leaf from $Q$ or (ii) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ contains two leaves from $Q$ and one from $P$, and $X \cap \mathcal{L}(T_2(\overleftarrow{y, w}))$ contains a leaf from $P$. The first term on the right hand side of Equation (4.15) is exactly the number of quartets that satisfy condition (i), and the second term on the right hand side is exactly the number of quartets satisfying condition (ii). □

**Lemma 4.6.7.** *The value $n_2(u_1, u_k, w)$ can be computed in $O(|\, adj(w)|)$ time.*

*Proof.* We will show that

$$\begin{aligned} n_3(u_1, u_k, w) &= \sum_{i=1}^{|\, adj(w)|} \left\{ \alpha - \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P|}{2} \right\} \cdot \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|}{2} \quad (4.16) \\ &+ \frac{1}{2} \sum_{i=1}^{|\, adj(w)|} \{\gamma - |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P| \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|\} \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P| \\ &\qquad \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|. \end{aligned}$$

Where,

$$\alpha = \sum_{i=1}^{|\,adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P|}{2}, \tag{4.17}$$

$$\gamma = \sum_{i=1}^{|\,adj(w)|} |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P| \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|. \tag{4.18}$$

The quartets $X$ counted in $n_3(u_1, u_k, w)$ are exactly those quartets for which there exist two neighbors $x, y$ of $w$ such that either (i) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ contains two leaves from $P$, and $T_2(\overleftarrow{y, w})$ contains two leaves from $Q$, or (ii) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ and $X \cap \mathcal{L}(T_2(\overleftarrow{y, w}))$ both contain one leaf each from $P$ and $Q$. The first term on the right hand side of Equation (4.16) is exactly the number of quartets that satisfy condition (i). The sum in the second term on the right hand side counts the quartets satisfying condition (ii) exactly twice each (due to the symmetry between $x$ and $y$ in condition (ii)). This explains the $\frac{1}{2}$ multiplicative factor. □

**Lemma 4.6.8.** *The value $n_2(u_1, u_k, w)$ can be computed in $O(|\,adj(w)|)$ time.*

*Proof.* We will show that

$$\begin{aligned} n_4(u_1, u_k, w) &= \sum_{i=1}^{|\,adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P|}{2} \cdot \binom{|\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap Q|}{2} \\ &+ \sum_{i=1}^{|\,adj(w)|} \binom{|\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q|}{2} \cdot \binom{|\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap P|}{2} \\ &+ \sum_{i=1}^{|\,adj(w)|} |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap P| \cdot |\mathcal{L}(T_2(\overleftarrow{x_i, w})) \cap Q| \cdot |\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap P| \\ &\qquad \cdot |\mathcal{L}(T_2(\overleftarrow{w, x_i})) \cap Q| \\ &- 2 \cdot n_3(u_1, u_k, w). \end{aligned} \tag{4.19}$$

The quartets $X$ counted in $n_4(u_1, u_k, w)$ are exactly those quartets for which there exist three neighbors $x, y, z$ of $w$ such that either (i) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ contains two leaves from $P$, and $T_2(\overleftarrow{y, w})$ and $T_2(\overleftarrow{z, w})$ each contain a leaf from $Q$, or (ii) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w}))$ contains two leaves from $Q$, and $X \cap \mathcal{L}(T_2(\overleftarrow{y, w}))$ and $X \cap \mathcal{L}(T_2(\overleftarrow{z, w}))$ each contain a leaf from $P$, or (iii) $X \cap \mathcal{L}(T_2(\overleftarrow{x, w})$ contains a leaf from $P$ and a leaf from $Q$, $X \cap \mathcal{L}(T_2(\overleftarrow{y, w}))$ contains a leaf from $P$, and $X \cap \mathcal{L}(T_2(\overleftarrow{z, w}))$ contains a leaf from $Q$.

**Procedure:** Approx-$\mathcal{R}_1(T_1, T_2)$
1: Convert the unrooted tree $T_1$ into a rooted one by rooting it at any internal node.
2: **for** each internal node $u \in \mathcal{V}(T_1) \setminus rt(T_1)$ **do**
3: **for** each internal unresolved node $w \in \mathcal{V}(T_2)$ **do**
4: Compute $r_1((\overleftarrow{u, pa(u)}), w)$.
5: **return** the sum of all computed $r_1(\cdot, \cdot)$.

Figure 4.3 Computing a 2-approximation to $|\mathcal{R}_1(T_1, T_2)|$

The first term on the right hand side of Equation (4.19) counts all the quartets that satisfy condition (i), and, in addition, all the quartets that satisfy condition (i) from the proof of Lemma 4.6.7. Similarly, the second term on the right hand side counts the quartets that satisfy condition (ii), along with all the quartets that satisfy condition (i) from the proof of Lemma 4.6.7. The third term on the right hand side counts those quartets that satisfy condition (iii), and also counts, exactly twice each (again due to symmetry), those that satisfy condition (ii) from the proof of Lemma 4.6.7. □

**Lemma 4.6.9.** *Given two unrooted phylogenetic trees $T_1$ and $T_2$ on the same size $n$ leaf set, a value $y$ such that $|\mathcal{R}_1(T_1, T_2)| \leq y \leq 2 \cdot |\mathcal{R}_1(T_1, T_2)|$ can be computed in $O(n^2)$ time.*

*Proof.* Our algorithm to compute a 2-approximate value of $|\mathcal{R}_1(T_1, T_2)|$ is summarized in Figure 4.3. Lemma 4.6.2 immediately implies that the algorithm computes a value between $|\mathcal{R}_1(T_1, T_2)|$ and $2 \cdot |\mathcal{R}_1(T_1, T_2)|$.

We now analyze the time complexity of our algorithm. By Lemmas 4.6.5, 4.6.6, 4.6.7, and 4.6.8, the values $n_1(u_1, u_k, w)$, $n_2(u_1, u_k, w)$, $n_3(u_1, u_k, w)$, and $n_4(u_1, u_k, w)$ can all be computed within $O(|\, adj(w)|)$ time. Hence, by Lemma 4.6.4, the value of any $\gamma(\cdot, \cdot, w)$ can be computed in $O(|\, adj(w)|)$ time. Lemma 4.6.3 now implies that, for any given $(u, v) \in \mathcal{E}(T_1)$ and $w \in \mathcal{V}(T_2) \setminus \mathcal{L}(T_2))$, the value $r_1((\overleftarrow{u, v}), w)$ can be computed within $O(|\, adj(u)| \cdot |\, adj(w)|)$ time. Thus, the total time complexity of the algorithm is $O(\sum_{u \in \mathcal{V}(T_1')} \sum_{w \in \mathcal{V}(T_2)} |\, Ch(u)| \cdot |\, adj(w)|)$, which is $O(n^2)$. □

# CHAPTER 5. General Conclusion

This thesis deals with three different problems related to phylogeny construction: The gene-duplication problem, the duplication-loss problem, and the problem of effectively and efficiently comparing phylogenies. Our algorithm for the gene-duplication problem, presented in Chapter 2, has already been successfully applied to biological data. Burleigh et al., in [15], used an implementation of our algorithm to infer the plant tree of life on 136 species from 18,896 input gene trees. This analysis would have been infeasible to perform without our efficient algorithm. However, a lot more needs to be done in order to make the gene-duplication and duplication-loss problems even more amenable to large scale phylogenetic analyses. For example, it would help to have efficient branch and bound or integer programming based techniques for solving the gene-duplication and duplication-loss problems exactly. This could be used to verify the performance of the local search heuristics used to solve these problems in practice. Efficient and effective ways to deal with error in the input trees are also needed.

Our work on comparing phylogenies was inspired in part by the problem of building consensus trees. One way to obtain a consensus tree from a host of input trees is to define an appropriate distance measure between trees, and then to declare the median tree as the consensus tree. Since our parametric distance measure is a metric for $p \geq 1/2$, we can easily find a 2-approximation to the median tree (by simply picking an input tree that minimizes the total distance to the other input trees as our solution). However, it is not known whether computing the median tree under our parametric triplet/quartet distance measure is easy; we suspect that it is NP-hard. It would also be interesting to see if our measure can be generalized to supertrees, while maintaining some of its desirable mathematical and algorithmic properties.

# BIBLIOGRAPHY

[1] B. L. Allen and M. Steel, *Subtree transfer operations and their induced metrics on evolutionary trees*, Annals of Combinatorics **5** (2001), 1–13.

[2] Lars Arvestad, Ann-Charlotte Berglund, Jens Lagergren, and Bengt Sennblad, *Bayesian gene/species tree reconciliation and orthology analysis using mcmc*, ISMB (Supplement of Bioinformatics), 2003, pp. 7–15.

[3] ———, *Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution*, RECOMB, 2004, pp. 326–335.

[4] M. S. Bansal, J. G. Burleigh, O. Eulenstein, and A. Wehe, *Heuristics for the gene-duplication problem: A $\Theta(n)$ speed-up for the local search*, RECOMB, 2007, pp. 238–252.

[5] Mukul S. Bansal, *Algorithms for minimum bipartite fill-in and the gene-duplication problem*, Master's thesis, Iowa State University, Ames, Iowa, USA, 2006.

[6] Mukul S. Bansal, Jianrong Dong, and David Fernández-Baca, *Comparing and aggregating partially resolved trees*, LATIN, 2008, pp. 72–83.

[7] Mukul S. Bansal and Oliver Eulenstein, *The gene-duplication problem: Near-linear time algorithms for NNI based local searches*, ISBRA, 2008, pp. 14–25.

[8] ———, *An $\Omega(n^2/\log n)$ speed-up of TBR heuristics for the gene-duplication problem*, IEEE/ACM Transactions on Computational Biology and Bioinformatics **5** (2008), no. 4, 514–524.

[9] M. A. Bender and M. Farach-Colton, *The LCA problem revisited*, LATIN, 2000, pp. 88–94.

[10] P. Bonizzoni, G. D. Vedova, and R. Dondi, *Reconciling a gene tree to a species tree under the duplication cost model*, Theor. Comput. Sci. **347** (2005), no. 1-2, 36–53.

[11] M. Bordewich and C. Semple, *On the computational complexity of the rooted subtree prune and regraft distance*, Annals of Combinatorics **8** (2004), 409–423.

[12] Gerth S. Brodal, Rolf Fagerberg, and Christian N. S. Pedersen, *Computing the quartet distance in time $O(n \log n)$*, Algorithmica **38** (2003), no. 2, 377–395.

[13] David Bryant, *Building trees, hunting for trees, and comparing trees: Theory and methods in phylogenetic analysis*, Ph.D. thesis, Department of Mathematics, University of Canterbury, New Zealand, 1997.

[14] David Bryant, John Tsang, Paul Kearney, and Ming Li, *Computing the quartet distance between evolutionary trees*, SODA '00: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2000, pp. 285–286.

[15] J. G. Burleigh, M. S. Bansal, O. Eulenstein, S. Hartmann, A. Wehe, and T. Vision, *Genome-Scale Phylogenetics: Inferring the Plant Tree of Life from 18,896 Discordant Gene Trees*, Under review, 2009.

[16] Cedric Chauve, Jean-Philippe Doyon, and Nadia El-Mabrouk, *Gene family evolution by duplication, speciation, and loss*, Journal of Computational Biology **15** (2008), no. 8, 1043–1062.

[17] Duhong Chen, Oliver Eulenstein, David Fernández-Baca, and J. Gordon Burleigh, *Improved heuristics for minimum-flip supertree construction*, Evolutionary Bioinformatics **2** (2006), 347–356.

[18] K. Chen, D. Durand, and M. Farach-Colton, *Notung: a program for dating gene duplications and optimizing gene family trees*, Journal of Computational Biology **7** (2000), 429–447.

[19] Chris Christiansen, Thomas Mailund, Christian N.S. Pedersen, Martin Randers, and Martin Stig Stissing, *Fast calculation of the quartet distance between trees of arbitrary degrees*, Algorithms for Molecular Biology **1** (2006), no. 16.

[20] J. A. Cotton and R. D. M. Page, *Phylogenetic supertrees: Combining information to reveal the tree of life*, ch. Tangled tales from multiple markers: reconciling conflict between phylogenies to build molecular supertrees, pp. 107–125, Springer-Verlag, 2004.

[21] J. A. Cotton, C. S. Slater, and M. Wilkinson, *Discriminating supported and unsupported relationships in supertrees using triplets*, Systematic Biology **55** (2006), no. 2, 345–350.

[22] William H. E. Day, *Analysis of quartet dissimilarity measures between undirected phylogenetic trees*, Systematic Zoology **35** (1986), no. 3, 325–333.

[23] O. Eulenstein, *Predictions of gene-duplications and their phylogenetic development*, Ph.D. thesis, University of Bonn, Germany, 1998, GMD Research Series No. 20 / 1998, ISSN: 1435-2699.

[24] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee, *Comparing partial rankings*, SIAM J. Discrete Math. **20** (2006), no. 3, 628–648.

[25] M. Farach and M. Thorup, *Optimal evolutionary tree comparison by sparse dynamic programming*, Proc. 35th Annual Symposium on Foundations of Computer Science (Piscataway, NJ), IEEE Computer Society Press, 1994, pp. 770–779.

[26] M. Fellows, M. Hallett, C. Korostensky, and U. Stege., *Analogs & duals of the MAST problem for sequences & trees*, European Symposium on Algorithms (ESA), 1998, pp. 103–114.

[27] J. Felsenstein, *Inferring phylogenies*, Sinauer Assoc., Sunderland, Mass, 2003.

[28] C. R. Finden and A. D. Gordon, *Obtaining common pruned trees*, J. Classification **2** (1985), no. 1, 225–276.

[29] M. Goodman, J. Czelusniak, G. W. Moore, A. E. Romero-Herrera, and G. Matsuda, *Fitting the gene lineage into its species lineage. a parsimony strategy illustrated by cladograms constructed from globin sequences*, Systematic Zoology **28** (1979), 132–163.

[30] P. Górecki and J. Tiuryn, *On the structure of reconciliations*, RECOMB Comparative Genomics Workshop, 2004.

[31] R. Guigó, I. Muchnik, and T. F. Smith, *Reconstruction of ancient molecular phylogeny*, Molecular Phylogenetics and Evolution **6** (1996), no. 2, 189–213.

[32] M. T. Hallett and J. Lagergren, *New algorithms for the duplication-loss model*, RECOMB, 2000, pp. 138–146.

[33] B. Ma, M. Li, and L. Zhang, *From gene trees to species trees*, SIAM J. Comput. **30** (2000), no. 3, 729–752.

[34] W. P. Maddison, *Reconstructing character evolution on polytomous cladograms*, Cladistics **5** (1989), 365–377.

[35] W. P. Maddison and D.R. Maddison, *Mesquite: a modular system for evolutionary analysis. version 2.6. http://mesquiteproject.org*, 2009.

[36] B. Mirkin, I. Muchnik, and T. F. Smith, *A biologically consistent model for comparing molecular phylogenies*, Journal of Computational Biology **2** (1995), no. 4, 493–507.

[37] R. D. M. Page, *Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas*, Systematic Biology **43** (1994), no. 1, 58–77.

[38] ———, *GeneTree: comparing gene and species phylogenies using reconciled trees*, Bioinformatics **14** (1998), no. 9, 819–820.

[39] ———, *Extracting species trees from complex gene trees: reconciled trees and vertebrate phylogeny*, Molecular Phylogenetics and Evolution **14** (2000), 89–106.

[40] R. D. M. Page and J. Cotton, *Vertebrate phylogenomics: reconciled trees and gene duplications*, Pacific Symposium on Biocomputing, 2002, pp. 536–547.

[41] W. Piel, M. Sanderson, M. Donoghue, and M. Walsh, *Treebase*, http://www.treebase.org, Last accessed 2 February 2007.

[42] D. F. Robinson and L. R. Foulds, *Comparison of phylogenetic trees*, Mathematical Biosciences **53** (1981), 131–147.

[43] M. J. Sanderson and M. M. McMahon, *Inferring angiosperm phylogeny from EST data with widespread gene duplication*, BMC Evolutionary Biology **7 (Suppl 1): S3** (2007).

[44] C. Semple and M. Steel, *Phylogenetics*, Oxford University Press, 2003.

[45] J. B. Slowinski, A. Knight, and A. P. Rooney, *Inferring species trees from gene trees: A phylogenetic analysis of the elapidae (serpentes) based on the amino acid sequences of venom proteins*, Molecular Phylogenetics and Evolution **8** (1997), 349–362.

[46] Yun S. Song, *On the combinatorics of rooted binary phylogenetic trees*, Annals of Combinatorics **7** (2003), no. 3, 365–379.

[47] M.A. Steel and D. Penny, *Distributions of tree comparison metrics — some new results*, Systematic Biology **42** (1993), no. 2, 126–141.

[48] Michael A. Steel, *Distributions on bicoloured evolutionary trees*, Ph.D. thesis, Massey University, 1989.

[49] Ulrike Stege, *Gene trees and species trees: The gene-duplication problem is fixed-parameter tractable*, WADS, 1999, pp. 288–293.

[50] M. Stissing, Christian N. S. Pedersen, Thomas Mailund, Gerth Stølting Brodal, and Rolf Fagerberg, *Computing the quartet distance between evolutionary trees of bounded degree*, APBC (David Sankoff, Lusheng Wang, and Francis Chin, eds.), Advances in Bioinformatics and Computational Biology, vol. 5, Imperial College Press, 2007, pp. 101–110.

[51] Ilan Wapinski, Avi Pfeffer, Nir Friedman, and Aviv Regev, *Automatic genome-wide reconstruction of phylogenetic gene trees*, ISMB/ECCB (Supplement of Bioinformatics), 2007, pp. 549–558.

[52] ———, *Natural history and evolutionary principles of gene duplication in fungi*, Nature **449** (2007), 54–61.

[53] A. Wehe, M. S. Bansal, J. G. Burleigh, and O. Eulenstein, *DupTree: a program for large-scale phylogenetic analyses using gene tree parsimony*, Bioinformatics **24** (2008), no. 13, 1540–1541.

[54] L. Zhang, *On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies*, Journal of Computational Biology **4** (1997), no. 2, 177–187.