Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Algorithms for isometric gene tree reconciliation

Master's Thesis

2019
Bc. Radoslav Chládek

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Algorithms for isometric gene tree reconciliation

Master's Thesis

Study programme:	Computer Science
Field of study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. Mgr. Bronislava Brejová, PhD.

Bratislava, 2019
Bc. Radoslav Chládek

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Radoslav Chládek
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Algorithms for isometric gene tree reconciliation

Annotation: Evolution of a group of related species can be often represented as a phylogenetic tree, which captures the order of branching of individual evolutionary lineages. In addition to these species trees it is also possible to reconstruct gene trees representing evolution of a particular gene in a set of related species. The goal of reconciliation is to map nodes of a gene tree to a species tree, thus aiding their interpretation. The problem of isometric reconciliation, in which we consider branch lengths in both trees, is not yet sufficiently explored. The goal of the thesis is to design new algorithms for various variants of the problem and potentially also to implement and test such algorithms.

Supervisor:	doc. Mgr. Bronislava Brejová, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	11.12.2017

Approved: 12.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Radoslav Chládek
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Algorithms for isometric gene tree reconciliation
Algoritmy pre izometrickú rekonciliáciu génových stromov

Anotácia: Evolúcia skupiny príbuzných organizmov sa často dá reprezentovať fylogenetickým stromom, ktorý zobrazuje poradie vetvenia jednotlivých evolučných línií. Okrem takýchto druhových stromov je možné zostaviť aj génové stromy, ktoré reprezentujú evolúciu jedného konkrétneho génu v rámci určitej skupiny druhov. Pri rekonciliácii sa snažíme namapovať vrcholy génového stromu na druhový strom a tým interpretovať ich význam v rámci evolúcie daného génu. Problém izometrickej rekonciliácie, v ktorej berieme do úvahy dĺžky hrán oboch stromov, bol zatiaľ pomerne málo študovaný. Cieľom práce je navrhnúť nové algoritmy na rôzne varianty tohto problému a prípadne takéto algoritmy aj implementovať a experimentálne otestovať.

Vedúci: doc. Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 11.12.2017

Dátum schválenia: 12.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I would like to thank my supervisor doc. Mgr. Broňa Brejová, PhD. for valuable advice, guidance and motivation. Without her extensive knowledge, support and enthusiasm, this thesis would not possibly come into existence.

Abstract

An evolution of a group of related biological species can be represented by a phylogenetic tree that we call a species tree. It is also possible to represent the evolution of one particular gene in a given group of species by a so-called gene tree. In this work, we study the problem of isometric gene tree reconciliation. The goal of this problem is to map the nodes of the gene tree to nodes or edges of the species tree in order to obtain a more comprehensive evolutionary history. Based on this history, we can interpret the events that took place in the evolution of the gene. Previous research has dealt with the case where the phylogenetic trees had exact edge lengths. For practical applicability, we extend the existing algorithms by allowing the input trees to have inexact edge lengths specified by intervals. The main goal of this thesis is to develop efficient algorithms for solving several variants of this problem.

Keywords: phylogenetic tree, isometric gene tree reconciliation, inexact branch lengths, parsimony

Abstrakt

Evolúciu skupiny príbuzných biologických druhov možno reprezentovať fylogenetickým stromom, ktorý nazveme druhový strom. Rovnako je možné reprezentovať evolúciu jedného konkrétneho génu v danej skupine druhov pomocou takzvaného génového stromu. V tejto práci sa zaoberáme problémom izometrickej rekonciliácie génového stromu. Cieľom tohto bioinformatického problému je namapovať vrcholy génového stromu na druhový strom za účelom získania komplexnejšej evolučnej histórie. Na základe tejto histórie vieme interpretovať význam jednotlivých udalostí, ktoré sa odohrali v evolúcii daného génu. Doterajší výskum sa zaoberal prípadom, kde vstupné evolučné stromy mali presne určené dĺžky hrán. Z dôvodu praktickej uplatniteľnosti rozširujeme existujúce algoritmy tak, že dovoľujeme vstupným stromom nepresné dĺžky hrán zadané ako intervaly. Hlavným cieľom práce je vývoj efektívnych algoritmov pre riešenie viacerých variantov tohto problému.

Kľúčové slová: fylogenetický strom, izometrická rekonciliácia génového stromu, nepresné dĺžky hrán, úspornosť

Contents

List of Figures

Introduction

The *theory of evolution* says that all biological species are related. Present-day species originate from mutual ancestral populations, but are all different thanks to the changes that occurred in their genetic information over successive generations.

Evolutionary relationships of biological entities can be shown by a branching diagram called *phylogenetic tree*. Phylogenetic trees are often represented as trees from graph theory, and they can be either *rooted* or *unrooted*. A *species tree* is a phylogenetic tree that shows the evolutionary relationships of various species. The leaves of the tree represent present-day species, and internal nodes represent *speciations*. A speciation is an evolutionary process by which a new distinct species is created from a given population. Evolution of a single gene is shown by a phylogenetic tree called a *gene tree*. The leaves of the tree represent copies of the gene in present-day species, and internal nodes represent speciations or *duplications*. A duplication is an evolutionary event causing that a gene is copied and placed in some other position in the genetic information of an organism.

In this work, we study the problem of *isometric gene tree reconciliation*. Input for this bioinformatics problem consists of a weighted species tree S, a weighted gene tree G and a mapping from the leaves of G to leaves of S. Mapping of the leaves determines the present-day species that contains a given gene copy. The goal of the reconciliation is to find a mapping from internal nodes of G to nodes or edges of S, while all ancestor-descendant relationships remain satisfied. From this mapping, we can determine whether a given internal node of G represents a duplication or a speciation and find the point in the evolutionary history, when this event took place. A specific mapping also implies the number of *gene losses* that had to occur, so that only the genes specified by G are present in the present-day species. Reconciliation that minimizes the number of duplications and gene losses is called *the most parsimonious reconciliation* [11].

In the first mention of isometric gene tree reconciliation problem [14], authors introduce and apply an algorithm for reconciliation of unrooted G and rooted S. An article [3] following their research, modifies the algorithm and improves the running time from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, where N is the total number of nodes in both input trees. Their algorithm for a case when both G and S are rooted has a running time $\mathcal{O}(N \log N)$

too and the article also studies more variants of the problem. Previous research has dealt with the case where the input trees have exact branch lengths. However, branch lengths of reconstructed phylogenetic trees cannot be determined exactly, only with some error. Current algorithms thus may not always succeed in finding the true reconciliation. In this work, we allow the input trees of the reconciliation to have inexact branch lengths specified by intervals, and we present algorithms for several variants of this problem.

In Chapter 1, we explain the necessary concepts from bioinformatics and the notation that will be used throughout this thesis. In Chapter 2, we introduce a polynomial-time algorithm based on linear programming. In Chapter 3, we present less general, but more efficient algorithms for the variant where only gene tree G has inexact branch lengths, and S is rooted with exact branch lengths. We improve the running time of the algorithm for the rooted case to $\mathcal{O}(N)$, if only partial reconciliation is required. Lastly, in Chapter 4, we address the problem of counting the number of duplications and gene losses implied by the isometric reconciliation, and we present an algorithm for finding the most parsimonious reconciliation for the case when G is unrooted with inexact branch lengths.

Chapter 1

Overview

In this chapter, we will introduce basic concepts from bioinformatics along with the notation that will be used in this thesis, and we will briefly describe other related work.

1.1 Basic terms

DNA (Deoxyribonucleic acid) is a molecule carrying the genetic instructions used in the growth, development, functioning and reproduction of all living organisms and many viruses. It is composed of two complementary strands that can be represented as a sequence of single units called nucleotides.

A **gene** is a subsequence of the DNA strand that encodes the creation of a single protein or other molecule with a function.

A **genome** is the entire genetic material of an organism. It consists of all the DNA present in the organism, and thus contains many genes. For example, the human genome contains approximately 20.000 genes [8].

A **phylogenetic tree** is a branching diagram showing evolutionary relationships of biological entities. It can be represented as a tree from graph theory.

For a tree T, we denote its nodes as $V(T)$, its edges as $E(T)$ and its leaves as $L(T)$. Trees that have their branch lengths defined are called **weighted**. For an edge (u, v), we will denote its branch length as $w(u, v)$. Trees can be either rooted or unrooted (Fig. 1.1).

A **rooted tree** is a tree with one designated node called **root**. For a given rooted tree T, we denote its root as $root(T)$. Rooted tree is a directed graph, where all of the edges are oriented away from the root. If $(u, v) \in E(T)$, u is the **parent** of v and v is the **child** of u. We denote the parent of v as $parent(v)$ and the set of its children as $children(v)$. All nodes have exactly one parent, except for the root, which doesn't

have a parent. Leaves don't have any children and internal nodes can have an arbitrary number of children.

An **ancestor** of a node v is an arbitrary node on the path from the root to v, including v. The nodes with an ancestor a are the **descendants** of a. For example, the root is an ancestor of all the nodes of a tree, which are all his descendants. A **subtree** rooted at node u consists of u and all its descendants. We will denote the number of nodes in a subtree rooted at u as N_u.

The **height** of a rooted tree is the number of nodes on the longest path from the root to one of the leaves. If a rooted tree T is weighted, by $D(u)$ we will denote the **depth** of the node u from T. The depth is computed as the sum of the lengths of all the edges on the path from $root(T)$ to u. If u is an ancestor of v, by $D_u(v)$ we denote the depth of v below u, i.e. the sum of the edge lengths on the path from u to v.

The **lowest common ancestor** of a set of nodes in a rooted tree is the shared ancestor of all the nodes in the set, located farthest from the root. For a set of nodes A, we will denote their lowest common ancestor as $lca(A)$.

An **unrooted tree** does not have a root, which means the parent-child relationship between adjacent nodes is not clearly established. A **subdivision** of an edge (u, v) is a process where a new node r is created on the edge (u, v) and the edge is replaced by edges (r, u) and (r, v). When subdividing a weighted edge, $w(r, u) + w(r, v) = w(u, v)$. An unrooted tree can be rooted by choosing one of its edges, adding a new root node by subdividing the chosen edge and orienting all of the edges away from the new root.

A rooted subtree of an unrooted tree can be acquired by removing an edge e from the tree, choosing one of the two newly created components and rooting the component at the respective node incident to e.

Let's assume we have an unrooted tree T and we know the new root of T is situated on edge $(u, v) \in E(T)$. A **semi-rooted tree** of T consists of the root edge (u, v) and the subtrees rooted at u and v acquired by removing the edge (u, v) from T.

1.2 Phylogenetic trees

A **species tree** is a phylogenetic tree that shows the evolutionary relationships among various species. Leaves of the tree represent present-day species and internal nodes represent speciations. An example of a species tree is shown in Fig. 1.2. We will denote a species tree as S.

A **speciation** is an evolutionary process by which populations evolve to become distinct species. It can occur due to various causes, for example, when the population is forced to be split between two geographical areas. Populations become distinct species

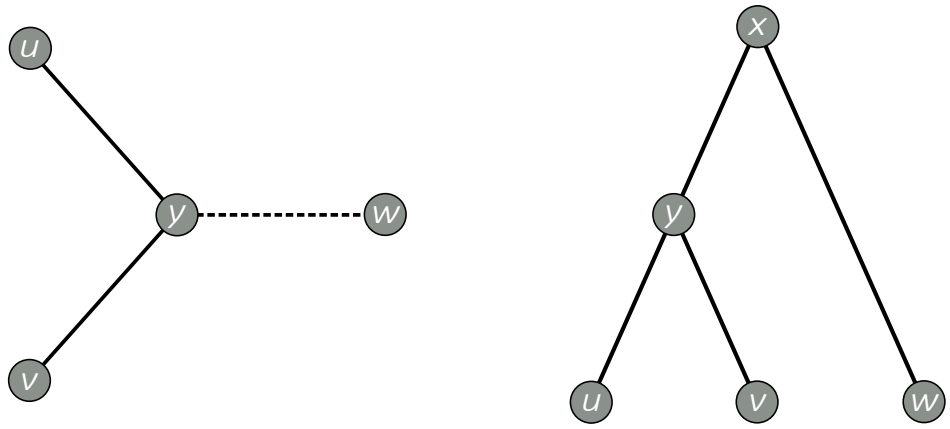

Figure 1.1: Difference between a rooted (right) and unrooted tree (left). Rooting the unrooted tree by the dashed edge will result in the rooted tree on the right, having the new node x as the root.

when their respective individuals cannot produce fertile offspring. When a speciaton occurs, all the genes present in the parent species will be also present in both of the new distinct species.

A **duplication** is an event causing that a sequence containing a gene is copied and inserted to some other position in the genome.

A **gene loss (deletion)** is an event in which a DNA sequence containing a gene is lost from the genome of an organism.

A **gene family** is a set of several similar genes in present-day species. A gene family is formed by duplications, speciations and gene losses that took place in the phylogeny of a single original gene. Genes from the same gene family generally have similar functions.

A **gene tree** is a phylogenetic tree showing the evolution of a single gene. Leaves of the tree represent genes belonging to a single gene family and internal nodes represent speciations or duplications. We will denote a gene tree as G. An example of multiple gene trees is shown in Fig. 1.3.

For a gene tree to be meaningful, the present-day species of origin for all its leaves (leaf mapping) must be known too. For gene tree G, a **paired species tree** S contains the species of origin for every gene in the gene family of G. For the paired S and G, we will denote the **leaf mapping** $L(G) \to L(S)$ as μ. For species a, we will usually denote the leaves of G that map to a as $a_1, a_2, .., a_k$.

By N we will denote the total number of nodes in G and S combined ($|V(G)| + |V(S)|$). We will say that the rooted trees are **balanced**, if their height is in $\mathcal{O}(\log N)$. By h we will denote the sum of heights of rooted G and S.

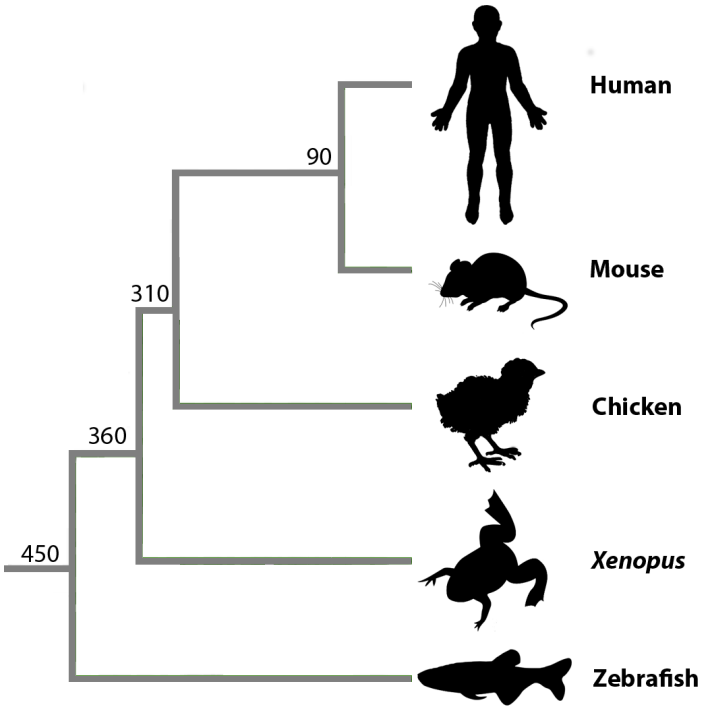

Figure 1.2: An example of a species tree of several commonly studied, model organisms. The numbers represent how many millions of years ago the speciations took place. (Source: [1])

Both gene and species trees can be reconstructed from the DNA sequences of present-day species by various, usually probabilistic, methods [7]. Although phylogenetic trees are naturally represented by rooted trees, many tree reconstruction methods can only produce unrooted trees. In general, the internal nodes of phylogenetic trees can have an arbitrary number of neighbors.

1.3 Gene tree reconciliation

An **evolutionary history** shows an evolution of one or more genes within the evolution of species. It is a possible scenario of how the gene families in present-day species actually developed. It shows how many duplications and gene losses have occurred and when did they happen. An example of multiple gene trees originating from the same history is shown in Figure 1.3.

We don't know what does the true history of a given gene family look like, but we want to find out. A possible history can be acquired from a gene tree and its paired species tree. However, the pair of trees may correspond to many different histories, because it is not clear which nodes of the gene tree correspond to speciations in the species tree.

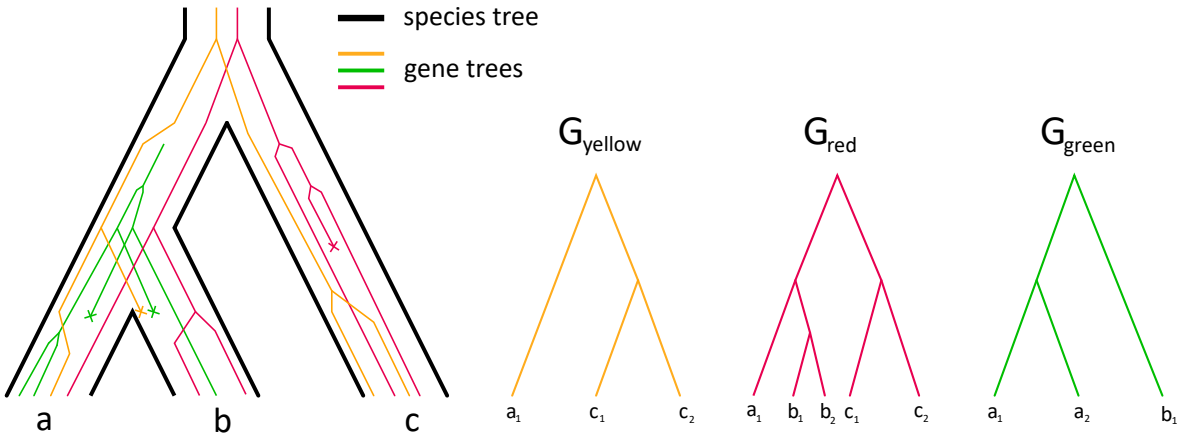

Figure 1.3: An example of a history and gene trees. On the left is a history containing the phylogeny of three genes in a small set of species. Gene losses are shown as a letter X. A single copy of the yellow and red gene was present in the ancestor of all three species. When the first speciation occurs, both of the genes are copied to each of the species. In the right branch of the species tree, we can see that the yellow gene is duplicated once, red gene is duplicated twice, but one of the red gene copies is lost. The green gene is a novel gene for the left subtree of the species tree. The gene is duplicated and after the speciation, one copy of the green gene is lost in each of the two present-day species. The topologies of the individual gene trees are shown on the right. Note that gene losses are not present in a gene tree and gene trees themselves also do not indicate which of their internal nodes are duplications and which are speciations.

Gene tree reconciliation is a problem where we are given a species tree S, a gene tree G and a leaf mapping $\mu: L(G) \to L(S)$. The goal of the reconciliation is to find a mapping Φ that maps the internal nodes of G to the nodes or edges from S, and thus to reconstruct the evolutionary history of the gene family. An example of a gene tree reconciliation is shown in Figure 1.4.

More formally, reconciliation takes G, S, and μ as an input and returns a triple (G_o, S_o, Φ) representing the evolutionary history of the gene family. Gene tree G_o is a rooted version of G, so if G is unrooted, G_o is obtained by subdividing an edge of G and rooting the tree in this new node. If G is rooted, $G_o = G$. Species tree S_o is a rooted version of S that also contains auxiliary nodes so that mapping Φ is from $V(G_o)$ to nodes (not edges) of S_o. Whenever some node u of G_o needs to be mapped to an edge of the species tree, we subdivide the edge by a new node $\Phi(u) \in V(S_o)$. Node $\Phi(u)$ represents a duplication event and has only one child. Duplications may also occur before the first speciation (the root) in the species tree. In this case, for a duplication $u \in V(G)$, a new edge $(\Phi(u), root(S))$ has to be added to S_o. There can be more duplications like this, so generally S_o has a path $r_1, r_2, \ldots, r_k$, where $r_1 = root(S_o)$ and $r_k = root(S)$. Mapping $\Phi: V(G_o) \to V(S_o)$ preserves the ascendant/descendant relationships; specifically for each edge $(u, v) \in E(G_o)$, $\Phi(u)$ is an ancestor of $\Phi(v)$. In addition, for each leaf u of G, $\Phi(u) = \mu(u)$.

Mapping Φ_{lca} maps each node $u \in V(G)$ to node $\Phi_{lca}(u) = lca(\{\mu(v) | v \in L_u\})$, where L_u is the set of all leaves in the subtree of G rooted at u. Node $\Phi_{lca}(u)$ represents the most recent species that could have contained the gene u. For the reconciliation to be meaningful, an internal node $u \in V(G)$ can be mapped only to $\Phi_{lca}(u)$ or its ancestors in S_o. Note that $\Phi_{lca}(u) = \mu(u)$ for $u \in L(G)$.

Mapping of an internal node u from G determines whether the node represents a duplication or a speciation [3]. The mapping also implies the number of gene losses that had to occur, so that only the genes specified by G are present in the present-day species. For example, in Figure 1.4 we can see that the mapping Φ determines both internal nodes x and y as duplications. The mapping of y doesn't imply any gene losses and $\Phi(x)$ implies two gene losses. Generally, the closer node u from G is mapped to $\Phi_{lca}(u)$ (the most recent species for u), the smaller the number of events. We will describe the duplications and deletions implied by the reconciliation more thoroughly in Chapter 4.

Since a reconciliation is not unique, the goal of an algorithm may be to find **the most parsimonious reconciliation** minimizing the number of duplications and gene losses.

The problem of gene tree reconciliation, where the trees are unweighted, is being studied since 1979 [9] and multiple algorithms based on the parsimony principle have

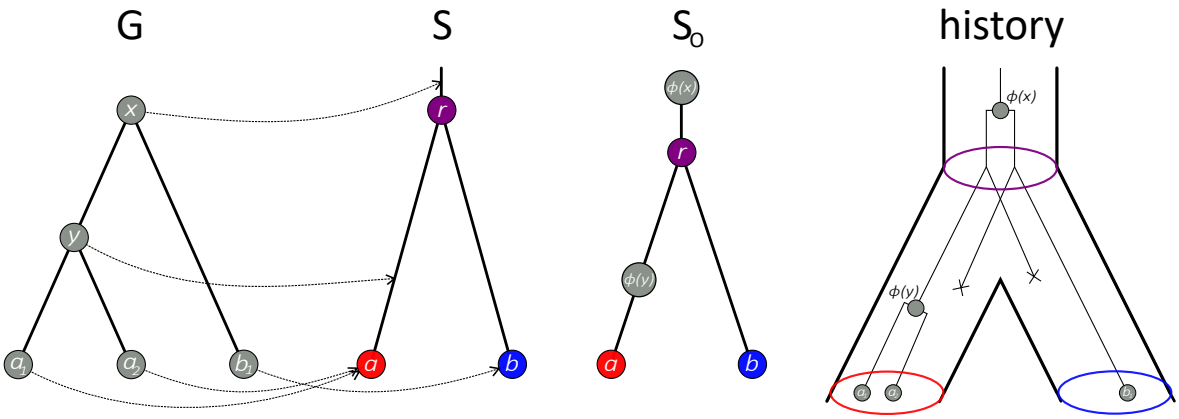

Figure 1.4: Reconciliation example. On the left we can see a possible reconciliation result for a simple rooted G and S. Internal node y is mapped to an edge of S and x is mapped above the root of S. If an internal node from G is mapped to an edge of S, it is a duplication. On the right is the evolutionary history specified by this reconciliation. Because x is a duplication, two copies of the gene are present in the ancestor of species a and b. One of the copies should be present in a and the other in b. That means both of them have to submit to the speciation r. However, the gene copy which should end up in b does not belong to a, so a gene loss of this gene copy had to occur during the evolution of a. The same applies to the gene copies in b. The duplication y does not produce any gene losses, so according to this reconciliation, there were in total two duplications and two losses. The most parsimonious reconciliation for this case is when node x is mapped exactly to r. Node x would be a speciation and no gene losses would occur.

been created [6, 11, 17, 18]. The most parsimonious reconciliation for rooted unweighted trees G and S maps each node u of G to node $\Phi_{lca}(u)$, which is the deepest point in S where the event corresponding to u could have happened. Mapping Φ_{lca} can be computed in $\mathcal{O}(N)$ time for all nodes of G [17].

1.4 Isometric reconciliation

In this section, we will consider that the gene tree G and species tree S are both weighted. Branch lengths of weighted phylogenetic trees represent an estimate of the time that has passed between the evolutionary events represented by the nodes. The time is usually approximated by the number of changes (mutations) between the sequences of present-day species.

A reconciliation is called **isometric**, when both G and S are weighted. Therefore, the problem takes both topology and branch lengths of the trees into account. For convenience, we will assume that G and S have all edges strictly positive. In order

to define the isometric reconciliation more formally, we will define several terms in accordance with previous work Brejová et al. [3], however, our definition will not be as detailed.

An **isometric mapping** from a rooted phylogenetic tree T_1 to a rooted phylogenetic tree T_2 is a mapping Φ of nodes of T_1 to nodes of T_2 such that if node $u \in V(T_1)$ is the parent of v, then $\Phi(u)$ is an ancestor of $\Phi(v)$ and $D_{\Phi(u)}(\Phi(v)) = D_u(v)$. Note that by induction, $D_{\Phi(u)}(\Phi(v)) = D_u(v)$ for any nodes u and v such that u is an ancestor of v in G.

An **input partial history** is a triple (G, S, μ) such that G and S are rooted or unrooted phylogenetic trees with positive edge weights, μ is a mapping from leaves of G to leaves of S, and each internal node v in both G and S satisfies the following. If the tree is rooted, v has at least two children. If the tree is unrooted, v has at least three neighbors.

The **isometric gene tree reconciliation** problem takes the input partial history (G, S, μ), and like the topology-only problem, returns a triple (G_o, S_o, Φ) that represents the evolutionary history of the gene family in G. The features of the trees G_o and S_o remain the same as in the topology-only problem. In short, G_o and S_o are rooted versions of G and S, where S_o was created by potentially subdividing edges of S and potentially adding a path leading to $root(S)$ from above. The edge lengths of G_o and S_o are also strictly positive. Mapping Φ is an isometric mapping from G_o to S_o, such that $\Phi(u) = \mu(u)$ for every leaf u of G ($L(G) = L(G_o)$). Note that the properties of an isometric mapping guarantee that each node is mapped to or above its Φ_{lca}. Every internal node $u \in V(G_o)$ thus satisfies $D(\Phi(u)) \leq D(\Phi_{lca}(u))$. It is possible that the input has no isometric reconciliation. In that case we reject the input as irreconcilable.

1.4.1 Related work

The isometric gene tree reconciliation problem has been introduced in 2008 [14]. An $\mathcal{O}(N^2)$ algorithm is presented for the variant when the gene tree is unrooted, the species tree is rooted and branch lengths are exact. Authors used this form of reconciliation as one of the steps in their polynomial-time algorithm to reconstruct a more detailed evolutionary history of multiple genomes.

The reconciliation algorithm is then modified and improved in an article by Brejová et al. [3] to $\mathcal{O}(N \log N)$ running time. An intuitive algorithm for the case when both G and S are rooted is provided too, with the same running time. The article also studies and presents algorithms for two extensions of the problem which were not considered before. Firstly an $\mathcal{O}(N^5 \log N)$ algorithm for the case when G and S are

both unrooted and secondly an algorithm for the case when both trees are rooted, but the branch lengths of the gene tree are scaled by some unknown factor which needs to be discovered by the algorithm. This was the first step towards a more practically applicable scenario where the branch lengths are not known exactly.

Reconciliation that takes some branch length information into account was also previously studied in several complex probabilistic models considering branch lengths in species tree [15, 10, 4] and models allowing horizontal gene transfer [2, 5]. However, these works ultimately deal with different problems than the one we study here.

1.4.2 Partial reconciliation

As the result of the reconciliation, sometimes we do not necessarily need S_o with edges subdivided by duplications. Some of our new algorithms output a **partial reconciliation**, which is a function $\delta: V(G_o) \to \mathbb{R}$ such that $\delta(u)$ is the mapping depth $D(\Phi(u))$ for each node u of G_o. For unrooted trees we will provide partial reconciliation along with the information about the location of the roots of G and S, so that the topology of the rooted versions of the input trees is known. We will later show that the partial reconciliation can be computed more efficiently than the full reconciliation.

If a full reconciliation exists, partial reconciliation δ exists too and is also a useful step towards finding the full reconciliation. We will also show that we can construct the full reconciliation from a given mapping δ. This may be useful if we are able to find the partial reconciliation efficiently, but the full isometric reconciliation is hard to compute directly.

1.5 Inexact branch lengths

Gene trees, species trees and their edge lengths are in practice estimated from DNA sequences collected from present-day species [7]. The edge lengths depend on the mutations that were observed in the input data. Mutations in evolution are mostly random and the input data are also just some random samples from the studied species. The inferred phylogenetic trees and their branch lengths are thus always estimated with some error.

When the trees undergo reconciliation, even small differences between branch lengths of the trees may prevent the current algorithms from finding the true solution. We build on the work of Brejová et al. [3], and we address this issue by allowing the input trees to have inexact branch lengths defined by intervals.

For each edge (u, v) we define an interval $w(u, v) = \langle w(u, v)_{min}, w(u, v)_{max} \rangle$ of possible edge lengths. When reconciling G and S with inexact branch lengths, we need

to choose a specific length for each edge $(u, v) \in E(G) \cup E(S)$. A combination of the chosen specific lengths for each edge, such that the isometric reconciliation exists, is a feasible solution for this problem. However, we will not represent the solution explicitly as the specific edge lengths, but as specific mapping depths $\delta(u)$ for each node $u \in V(G)$ and depths $D(a)$ for each node $a \in V(S)$. As a result of a successful isometric reconciliation of G and S with inexact branch lengths, we can either provide one possible solution or characterize the set of all feasible solutions. In the rest of this thesis, we study the problem of reconciling trees with inexact branch lengths and present algorithms for solving several variants of this problem.

Chapter 2

Linear programming

In this chapter, we will show that the problem of isometric gene tree reconciliation, where the input trees have inexact branch lengths specified by intervals, can be formulated as a linear program.

Linear programming (LP) is a technique for the optimization of a linear objective function f, subject to linear equality or inequality constraints for n variables. If all the constraints of LP are satisfied, the problem formulated by LP has a solution. When the objective function f is specified, LP finds such values of the variables that f is maximal or minimal. If the number of constraints is polynomial, LP can be solved in polynomial time [12].

2.1 Rooted trees

In order to formulate the problem of isometric reconciliation, we have to specify the variables we want to use. For each node $u \in V(G)$, we introduce a variable X_u representing its mapping depth $X_u = \delta(u) = D(\Phi(u))$. For each node $a \in V(S)$ we represent its unknown depth by variable $Y_a = D(a)$.

First let's assume that both G and S are rooted with inexact branch lengths. In order to find a valid isometric reconciliation, our variables have to satisfy the following constraints:

$$X_v - X_u \leq w(u,v)_{\max} \qquad \forall (u,v) \in E(G) \tag{2.1}$$

$$X_v - X_u \geq w(u,v)_{\min} \qquad \forall (u,v) \in E(G) \tag{2.2}$$

$$Y_b - Y_a \leq w(a,b)_{\max} \qquad \forall (a,b) \in E(S) \tag{2.3}$$

$$Y_b - Y_a \geq w(a,b)_{\min} \qquad \forall (a,b) \in E(S) \tag{2.4}$$

$$X_u \leq Y_{\Phi_{lca}(u)} \qquad \forall u \in V(G) \tag{2.5}$$

$$X_u = Y_{\mu(u)} \qquad \forall u \in L(G) \tag{2.6}$$

$$Y_r = 0 \tag{2.7}$$

The first two inequalities say that the distance between the mapping depths of two neighboring nodes u and v from G must be between the minimal and maximal length of (u, v). Inequalities 2.3-2.4 similarly restrict the depths of neighboring nodes of S by the edge lengths of S. Inequality 2.5 says that each node from u has to be mapped to or above $\Phi_{lca}(u)$. Note that we need to find Φ_{lca} before we can solve the linear program. Equality 2.6 represents the mapping of leaves by μ. Note that if u is a leaf, $\Phi_{lca}(u) = \mu(u)$. In general, the root of S can be situated in any depth. To unify the results, we will state that the root of the species tree is always in depth 0 (equality 2.7). The nodes that map above $root(S)$ are thus mapped to negative depths. Note that the linear program works even when G and S are non-binary trees. Since there is $\mathcal{O}(N)$ nodes and edges in the trees, LP is formulated by $\mathcal{O}(N)$ constraints and can be solved in polynomial time.

Finding a solution

To find an interval $X[u] = \langle X[u]_{min}, X[u]_{max} \rangle$ or $Y[a] = \langle Y[a]_{min}, Y[a]_{max} \rangle$ of possible mapping depths for each node $u \in V(G)$ and $a \in V(S)$, we can solve the linear program where the variable X_u or Y_a (set as objective function) is first minimized and then maximized. With one variable fixed to any value from its interval, there has to exist a value of all other variables from their respective intervals, such that all LP constraints are satisfied. However, mapping some node to its, for example, maximum possible depth may prevent some other node from being mapped to its minimum/maximum depth.

Justification

From the formulation of LP, it is clear that every reconciliation has to satisfy those constraints. Here, we will prove that every solution of LP corresponds to an isometric reconciliation. To prove this, we will show an algorithm for constructing isometric reconciliation (G, S_o, Φ), given a partial reconciliation δ satisfying the aforementioned LP constraints ($\delta(u)$ of node $u \in V(G)$ is equal to some X_u that satisfies the constraints of LP).

Consider node u in G. If $\delta(u) < 0$, node u should map to a newly added node positioned in distance $\delta(u)$ above the original root of S. If $\delta(u) \geq 0$, we first find node $\Phi_{lca}(u)$, and then locate the edge e in S on the path from $root(S)$ to $\Phi_{lca}(u)$, which involves depth $\delta(u)$. We will subdivide edge e by a new node $\Phi(u)$ at depth $\delta(u)$ and map u to this node. If the depth $\delta(u)$ is equal to $D(a)$ of node a of S_o before subdividing e, u is mapped to node a. This process can be repeated for every node u of G, creating mapping Φ and tree S_o with subdivided edges.

To see that we obtain a valid isometric reconciliation, first consider node u which is a leaf in G. Since $\delta(u) = D(\mu(u))$ and $\Phi_{lca}(u) = \mu(u)$, the above process will map u correctly to $\mu(u)$. Now consider an edge (u, v) of G. Because of condition 2.5, $\Phi(v)$ is an ancestor of $\Phi_{lca}(v)$. Similarly, $\Phi(u)$ is an ancestor of $\Phi_{lca}(u)$ and thus also an ancestor of $\Phi_{lca}(v)$ ($\Phi_{lca}(u)$ is an ancestor of $\Phi_{lca}(v)$). Both $\Phi(u)$ and $\Phi(v)$ are thus on the path from $\Phi_{lca}(v)$ to the root, and as a result, one must be the ancestor of the other. Thanks to conditions 2.1-2.2, v is in a greater depth, which means that $\Phi(u)$ is the ancestor. These conditions also imply that they are in the correct distance, i.e., that $D_{\Phi(u)}(\Phi(v)) \in w(u, v)$.

Solutions decreasing the number of events

The inequalities of our LP formulation may have many solutions and sometimes we want to choose one particular solution. Mapping nodes of G closer to their Φ_{lca}, i.e. their deepest possible mapping point, decreases the number of losses implied by Φ. If we maximize objective function $\sum_{u \in V(G)} X_u$, nodes from G will be mapped to their maximum depth. However, this objective function doesn't involve the depths of nodes in S which can be thus set far from the resulting X_u. Minimization of the sum of distances between $\Phi_{lca}(u)$ and X_u ($\sum_{u \in G}(Y_{\Phi_{lca}(u)} - X_u)$) may be a better alternative. However, a solution where some of these distances are greater, so that other can be shorter, may imply fewer duplications and losses. Neither of these solutions therefore guarantee the most parsimonious reconciliation, but they can provide practical heuristics.

2.2 Semi-rooted and unrooted trees

If G and S are semi-rooted, most of the previous constraints apply to this case too, except for the ones concerning the new root and its future adjacent nodes.

Let q be the root of G located at an unknown position on edge $(u, v) \in E(G)$. Note that in the rooted version of G, $\Phi_{lca}(q) = lca(\{\Phi_{lca}(u), \Phi_{lca}(v)\})$ which can potentially be the root of S, whose exact location is unknown for semi-rooted S. After subdividing (u, v), the root q will be adjacent to nodes u and v, which will no longer be adjacent to each other. We will thus replace constraints (2.1) and (2.2) concerning only the edge (u, v) by the following conditions:

$$w(u, v)_{min} \leq (X_u - X_q) + (X_v - X_q) \leq w(u, v)_{max} \tag{2.8}$$

$$X_u - X_q \geq 0 \tag{2.9}$$

$$X_v - X_q \geq 0 \tag{2.10}$$

After replacing the two constraints (2.3) and (2.4) concerning the root edge of S by analogous conditions too, we get the LP formulation for the semi-rooted G and S.

Constraints 2.8 say that the sum of edge lengths of (u, q) and (v, q) has to belong to interval $w(u, v)$. The last two constraints ensure that u and v are mapped deeper than q. Note that this formulation allows one of the edges (q, u) and (q, v) to be of zero length.

Now let's assume that G and S are unrooted. We won't formulate LP for this case, but we can execute LP for each combination of semi-rooted G and S. The set of solutions for unrooted G and S will thus be separated to multiple subsets, depending on the positions of the roots. Solving LP for all root positions will worsen the time complexity of LP by the factor N^2, therefore it is still polynomial.

In conclusion, we have shown that the problem of isometric reconciliation with inexact branch lengths can be solved in polynomial time.

Chapter 3

Efficient algorithms

In this chapter, we will present faster algorithms for the case where the species tree is rooted with exact edge lengths and we will allow the inexact edge lengths specified by intervals only in the gene tree. We chose this variant because species trees are usually reconstructed on the basis of multiple estimated gene trees [13] and therefore, their topologies and edge lengths are better known in contrast to the generally less accurate gene trees.

3.1 Rooted gene tree

In this section, we will provide an efficient algorithm for the isometric gene tree reconciliation of rooted G with inexact edge lengths. If only partial reconciliation is required, our algorithm has a better running time than the existing algorithm [3] for the full reconciliation of rooted trees with exact branch lengths.

Since species tree S is rooted with exact edge lengths, we don't need to find a new root for S and we don't need to find specific depths of nodes of S, because they are already fixed. However, it is useful to precompute $D(a)$ for each $a \in V(S)$ which can be done in linear time. If G is rooted, the only variable is the mapping depth $\delta(u) = D(\Phi(u))$ for each node $u \in V(G)$. Our goal is to find an interval $X[u] = \langle X[u]_{min}, X[u]_{max} \rangle$ of possible mapping depths for each $u \in V(G)$. These intervals should have the same properties as the intervals computed by LP in the previous chapter.

Algorithm description

Our algorithm (Alg. 3.1) uses two sweeps over the rooted gene tree. In the first (upward) sweep, we create an auxiliary interval $x[u] = \langle x[u]_{\min}, x[u]_{\max} \rangle$ for each node $u \in V(G)$. This interval is the set of all possible values of $X_u = D(\Phi(u))$ over all reconciliations mapping the subtree of G rooted at u to the species tree S. Each point in this interval is therefore a part of some solution consistent with all descendants of

Algorithm 3.1: A linear-time algorithm for reconciling rooted G with interval edge lengths and rooted S with exact edge lengths

```
1  def reconcile(q ∈ V(G)):
2      upward(q)
3      downward(q)
4
5  def upward(u ∈ V(G)):
6      for v ∈ children(u): upward(v)
7      if u ∈ L(G):
8          x[u] = [D(μ(u)), D(μ(u))];
9      else :
10         x[u]_min = max_{v∈children(u)}(x[v]_min − w(u, v)_max)
11         x[u]_max = min(D(Φ_lca(u)), min_{v∈children(u)}(x[v]_max − w(u, v)_min))
12     if x[u]_min > x[u]_max: print ''no solution''; finish
13
14 def downward(u ∈ V(G)):
15     p = parent(u)
16     if p is null: X[u] = x[u] # root
17     else:
18         X[u]_min = max(x[u]_min, X[p]_min + w(p, u)_min)
19         X[u]_max = min(x[u]_max, X[p]_max + w(p, u)_max)
20     for v ∈ children(u): downward(v)
```

u, but not taking into account other parts of G. In the second (downward) sweep, we will specify the searched interval $X[u]$ for each node $u \in V(G)$ by further constraining $x[u]$.

If node u from G is a leaf, it has to be mapped exactly to node $\mu(u)$ of S. Both the auxiliary interval $x[u]$ and the final interval of mapping depths $X[u]$ for leaf u are thus always equal to $\langle D(\mu(u)), D(\mu(u))\rangle$.

First, let's look at the upward sweep of G. For an internal node u, let's say that each $v \in children(u)$ has its interval $x[v]$ known. The highest point $x[u]_{\min}$ where u can be mapped is obtained from child v by taking the highest point where v might map $(x[v]_{\min})$ and subtracting the maximal edge length $w(u,v)_{\max}$ from it. So that the minimal depth $x[u]_{\min}$ can be reached by all children, we set it to $\max_{v\in\text{children}(u)}(x[v]_{\min} - w(u,v)_{\max})$.

The deepest point $x[u]_{\max}$, where u can be mapped, is obtained similarly. From the deepest points of children we subtract the minimal edge lengths and take the minimum

of these depths. We will also take into account that node u can be mapped at least to or above $\Phi_{lca}(u)$. Therefore we constrain $x[u]_{\max}$ to be at most $D(\Phi_{lca}(u))$.

If the interval $x[u]$ is empty ($x[u]_{\max} < x[u]_{\min}$), the input trees have no reconciliation because node u cannot be mapped consistently with all its descendants, and thus we reject the input. Otherwise, we will continue with the sweep. If root q of G is reached, interval $x[q]$ is also the final interval $X[q]$ because it is consistent with the mapping intervals of all other nodes of G.

In the downward sweep, we set the mapping interval $X[u]$ of each node $u \in V(G)$ (except for the root) according to the already known interval $X[p]$ of its parent p. If we intersect $x[u]$ with interval $\langle X[p]_{\min} + w(p,u)_{\min}, X[p]_{\max} + w(p,u)_{\max}\rangle$, we get interval $X[u]$ consistent with both the descendants of u and with the parent, thus consistent with the rest of the tree. Note that this intersection cannot be empty. If we get to run the downward sweep, interval $X[root(G)]$ is not empty and thus there has to be at least one solution for the isometric reconciliation.

The algorithm passes through all nodes of G twice, while computing intervals of mapping depths. As we mentioned in Chapter 1, computation of $\Phi_{lca}(u)$ is in linear time. The running time of the algorithm is thus $\mathcal{O}(N)$.

The most parsimonious reconciliation

If we set the mapping depth of any node u of G to $X[u]_{max}$, we can also set the mapping depth of every child $v \in children(u)$ to $X[v]_{max}$ because our algorithm guarantees that $X[v]_{max}$ can be reached from $X[u]_{max}$ by $w(u,v)$. Choosing the maximum depth $X[u]_{\max}$ as $\delta(u)$ for each node $u \in V(G)$ will thus lead to a valid solution. Also, there isn't a solution where the mapping depth X_u of an arbitrary node u is deeper than $X[u]_{\max}$. Therefore, mapping each node u to depth $\delta(u) = X[u]_{\max}$ is the most parsimonious among all isometric reconciliations, since all nodes are mapped as close to their Φ_{lca} as they can be.

Exact branch lengths are a special case of interval branch lengths, where $w(u,v)_{\min} = w(u,v)_{\max}$ for every edge (u,v). Therefore, our algorithm can be applied to trees with exact branch lengths too. Finding the full reconciliation for rooted G and S with exact branch lengths takes $\mathcal{O}(N \log N)$ time. Our algorithm can be used to find the partial reconciliation of G and S more efficiently, if the full reconciliation is not needed.

We are also able to determine the exact branch lengths of G by the partial (in this case, the most parsimonious) reconciliation δ. Thanks to the property of isometric mapping, unknown branch length $w(u,v)$ of $v \in V(G) \setminus root(G)$ with parent u can be determined as $\delta(v) - \delta(u)$. Determining edge lengths for all edges of G can be therefore achieved in $\mathcal{O}(N)$ time. With branch lengths determined, we can set (G, S, μ) as the

input for the existing algorithm to find the full reconciliation in $\mathcal{O}(N \log N)$ time.

3.2 Semi-rooted and unrooted gene tree

In this section, we assume that the input gene tree G is unrooted with inexact branch lengths. We will present an algorithm that uses semi-rooted versions of G to find all solutions of the isometric reconciliation.

Algorithm description

Let root q of G be situated at an unknown location at the edge (u, v). Edge (u, v) joins two subtrees of G rooted at u and v. We can call $upward(u)$ and $upward(v)$ to determine the possible mapping depth intervals $x[u]$ and $x[v]$ that are consistent with all mapping depths of descendants of u and v. If one of $x[u]$, $x[v]$ is empty, isometric reconciliation doesn't exist.

Our goal is to determine intervals $X[u]$, $X[v]$ and $X[q]$. With those intervals established, we can call $downward(u)$ and $downward(v)$ to determine the possible mapping depth intervals for each node of G and thus to find a solution of the isometric reconciliation.

To obtain $X[u]$, $X[v]$ and $X[q]$, we formulate a linear program of three variables as in Chapter 2. We will use the constraint 2.5 for q (u and v are already mapped above their Φ_{lca} thanks to *upward*) and constraints 2.8-2.10 concerning the semi-rooted case. In addition, we will add inequalities $x[u]_{\min} \leq X_u \leq x[u]_{\max}$ and $x[v]_{\min} \leq X_v \leq x[v]_{\max}$. This LP of constant size represents the formulation of our problem. By maximizing and minimizing each of the objective function X_u, X_v and X_q, we find the desired intervals $X[u]$, $X[v]$ and $X[q]$. If the linear program cannot find a solution, root q cannot be placed on edge (u, v) and thus the isometric reconciliation doesn't exist.

In this algorithm, we passed through all nodes of G by *upward* and *downward*, and we solved LP of constant size. Therefore, the running time is $\mathcal{O}(N)$. For unrooted G, we can use this algorithm for every possible edge, where q can be located, with total $\mathcal{O}(N^2)$ time.

Finding a solution

When we maximize the objective function X_u, the linear program may minimize X_v because it will try to map v as close as possible to q, so the remaining length of edge (u, v) is used for mapping u as deep as possible. Likewise, maximizing X_v may minimize X_u and therefore, depths $X[u]_{\max}$ and $X[v]_{\max}$ may not be achievable at the same time.

In order to find a solution (partial reconciliation) that could intuitively minimize the number of duplications and losses, we can always set $\delta(q)$ to $X[q]_{max}$, with the

additional advantage that the edge length of (u, v) available for mapping u and v is maximal. Then, we can either maximize $\delta(u)$, $\delta(v)$ or $\delta(u)+\delta(v)$. If we choose the first, we set $\delta(u)$ to $X[u]_{\max}$ and find maximal $\delta(v) \in X[v]$ such that the distance between $\delta(u)$ and $\delta(v)$ (through $\delta(q)$) is maximal, considering the remaining edge length. We can maximize $\delta(v)$ by similar approach. If we want to maximize $\delta(u) + \delta(v)$, we can find the values by maximizing objective function $X_u + X_v$ in our LP formulation.

To find δ of all remaining nodes, we will run a downward sweep on the subtrees rooted at u and v such that each node $y \in V(G) \setminus \{u, v, q\}$ with parent x will have its $\delta(y)$ set to $min(\delta(x) + w(x, y)_{max}, X[y]_{max})$.

However, none of the aforementioned ways of determining $\delta(u)$ and $\delta(v)$ guarantee the minimal number of events. We will present an algorithm for finding the most parsimonious solution for this variant in the next chapter.

Chapter 4

Parsimony

In this chapter, we firstly describe the problem of counting the duplications and losses determined by a reconciliation. We will then present an algorithm for finding the total number of events of an isometric reconciliation and we will provide an algorithm for counting the minimal number of events for all reconciliations, where G is rooted with inexact branch lengths. Lastly, we will use these algorithms to find the most parsimonious reconciliation of unrooted G with inexact branch lengths and rooted S with exact branch lengths. In this chapter, we will consider only rooted trees that are binary and unrooted trees, where the internal nodes have exactly three neighbors.

4.1 Preliminaries

In Chapter 1 we mentioned that the reconciliation specifies the number of duplications and gene losses that had to occur. In this section, we will introduce the problem of counting these evolutionary events, and we will define terms and notations that will be used later in this chapter.

If the input trees of a reconciliation are unweighted, the most parsimonious solution is represented by history (G, S, Φ_{lca}). In this history, internal node u is a duplication only if there exists such $v \in children(u)$ that $\Phi_{lca}(u) = \Phi_{lca}(v)$. Otherwise u is a speciation [16]. We will not describe how are the losses calculated in unweighted trees, as it can be done in multiple ways. An example can be found in article Tabaszewski et al. [16]. The authors compute the number of duplications and losses, along with some additional cost functions of a reconciliation of unweighted G and S, in order to reach their goal.

In our algorithms, we will consider that duplications and losses are the only events that could have occured in an evolutionary history, except for speciations. However, we will mention the case where we allow a new gene to be created during the evolutionary history, i.e. an ancestral gene doesn't have to be present at the beginning of the history.

We will generally represent the number of duplications and losses as a pair $DL = (dup, loss)$, where dup and $loss$ are the respective numbers of duplications and losses. We will define the sum of DL_1 and DL_2 as $DL_1 + DL_2 = (dup_1, loss_1) + (dup_2, loss_2) = (dup_1 + dup_2, loss_1 + loss_2)$. We say that DL_1 is lower than DL_2, if $dup_1 + loss_1 < dup_2 + loss_2$. Thus whenever we compare two DLs, we are comparing the sums of their respective duplications and losses.

Internal node u of G represents a speciation if and only if it is mapped exactly to $\Phi_{lca}(u)$. Otherwise, u is a duplication. We will define a boolean function $isDup(u)$ that is $true$ if node u represents a duplication ($D(\Phi(u)) < D(\Phi_{lca}(u))$). If u represents a speciation or it is a leaf, $isDup(u)$ is $false$.

Note that in the topology-only problem, each node u of G is mapped to $\Phi_{lca}(u)$. In isometric reconciliation, if $\Phi_{lca}(u)$ equals to Φ_{lca} of one of its children (u represents a duplication in the topology-only problem), u cannot be mapped to $\Phi_{lca}(u)$. Otherwise, one of the edges connecting u to its child would have to be of zero length. However, we require branch lengths to be strictly positive.

Let (G_o, S_o, Φ) be the resulting history of an isometric reconciliation. We will denote a set of binary nodes of S_o (nodes that represent speciations) as $bin(S_o)$. If node $u \in V(G_o)$ represents a duplication ($isDup(u) = true$), and u is mapped to node $a \in bin(S_o)$, reconciliation implies that a duplication and a speciation event occur at the same time. However, one of the events has to happen before the other, and thus we can choose which of the events was the first. To minimize the number of losses, we will state that the duplication will always happen after the speciation.

To formalize this, by Φ^* we will denote a mapping from $V(G_o)$ to $V(S_o^*)$, where the tree S_o^* is created from S_o by subdividing some of its edges. In particular, duplication node u such that $\Phi(u) \in bin(S_o)$ is mapped to a new node $\Phi^*(u)$ that is added to the path from $\Phi(u)$ to $\Phi_{lca}(u)$ in depth $D(\Phi(u)) + \epsilon$. Tree S_o^* has a property that each node from $V(S_o^*) \setminus bin(S_o)$ represents a duplication and the other nodes represent only speciations. Note that if S is rooted, all nodes of $bin(S_o)$ are from S.

In order to specify the number of gene losses implied by isometric reconciliation, we will define the following term. Let a be a node from S_o^*, **bearing node** of a is the closest descendant of a that is from $bin(S_o)$. We will denote it as $B(a)$. If $a \in bin(S_o)$, $B(a)$ equals a.

To find a bearing node of $\Phi^*(u)$, where $u \in G$, we don't need to actually know $\Phi^*(u)$. Only Φ_{lca} and the partial reconciliation δ are needed. To find the bearing node, we start at $\Phi_{lca}(u)$ and continue on the path to $root(S)$, until we reach the first node b ($\delta(u) \leq D(b)$) that satisfies the following:

$$(b = root(S)) \vee (D(parent(b)) < \delta(u)) \vee (D(parent(b)) = \delta(u) \wedge isDup(u))$$

Node b is then the bearing node of $\Phi^*(u)$. We define the function that returns the bearing node of $\Phi^*(u)$ by $\delta(u)$ as $BN(\delta(u))$. If the function passed through all the nodes from $\Phi_{lca}(u)$ to $root(S)$, the time complexity would be $\mathcal{O}(h)$.

However, Brejová et al. [3] use a data structure called **level ancestor** that can find the ancestor of any node a in distance d from a on the path towards the root in $\mathcal{O}(\log N)$ time. We will denote the result of level ancestor as $anc(a, d)$. Level ancestor looks at the point in distance d from a towards the root, and if there is a node at this point, it returns this node. If there is no node at this point, it returns the furthest ancestor of a in distance smaller than d. We can find the bearing node of u by finding an ancestor $b = anc(\Phi_{lca}(u), D(\Phi_{lca}(u)) - \delta(u))$ in S. The ancestor b is the bearing node, except for the case when $D(b) = \delta_u$ and $isDup(u)$. This is the case, where duplication u is mapped exactly to a node with two children and thus $B(\Phi^*(u))$ is the child of b on the path to $\Phi_{lca}(u)$. We can find the bearing node as $anc(\Phi_{lca}(u), D(\Phi_{lca}(u)) - \delta(u) - \epsilon)$, where $\epsilon > 0$ is a distance smaller than any edge length of S. In conclusion, the function $BN(\delta(u))$ finds the bearing node of $\Phi^*(u)$ in $\mathcal{O}(\log N)$ time.

Sometimes we will want to count only such nodes of a path in S_o^* that are from S. By $path_S(a, b)$ we denote all nodes of S on the path in S_o^* from node $a \in V(S)$ to node $b \in V(S)$. We denote the number of nodes in $path_S(a, b)$ as $|path_S(a, b)|$.

Let a be an arbitrary node of S. By $h_S(a)$ we will denote the height of a in S, i.e. the number of nodes on the longest path from a to a leaf of S (the height of a leaf is 1). The heights of all nodes in S can be straightforwardly computed in $\mathcal{O}(N)$ time. Note that if a is an ancestor of b in S, $|path_S(a, b)|$ can be computed as $h_S(a) - h_S(b) + 1$.

4.2 Counting the events of a reconciliation

Here, we will consider that G and S are trees with exact branch lengths. Our goal is to find the total DL for a reconciliation (G_o, S_o^*, Φ^*). We will denote this number as DL_{Φ^*}.

Let u, v and w be nodes from G_o, such that v and w are children of u. By DL_u we denote the DL caused by the mapping of u. More precisely, DL_u is the total DL of a subtree of G rooted at u minus the sum of DL counts of the subtrees rooted at v and w. If u is a leaf, it always maps to a leaf of S and it doesn't have any children, hence $DL_u = (0, 0)$.

If u is an internal node, DL_u can have many losses, but at most one duplication. Firstly, we will analyze the number of losses for an internal node u, such that $isDup(u) = true$. Duplication node u and its children v and w are mapped by Φ^*. We can divide the problem into two cases. The first case is when $\Phi^*(v)$ is an ancestor of $\Phi^*(w)$ or vice versa. In the second case, there doesn't exist a path from any leaf to

$root(S)$ that contains both $\Phi^*(v)$ and $\Phi^*(w)$.

For both of the cases, we need to find bearing nodes $B_u = B(\Phi^*(u))$, $B_v = B(\Phi^*(v))$ and $B_w = B(\Phi^*(w))$ and the node $x = lca(\{B_v, B_w\})$. In the first case, x is one of B_v, B_w, depending on which one is the ancestor of the other. In the second case $x = \Phi_{lca}(u)$.

When a single gene enters $\Phi^*(u)$, the number of lost genes before reaching node x is the same in both cases. Node u is a duplication, so two gene copies exist on the path from $\Phi_{lca}(u)$ to x. These two gene copies have to be lost after each speciation on this path (except for x), in the branch that doesn't lead to x. The number of genes lost before reaching x is then $2(|path_S(B_u, x)| - 1)$.

In the first case, let's say $\Phi^*(v)$ is an ancestor of $\Phi^*(w)$. Node x thus equals B_v. One of the gene copies reaches $\Phi^*(v)$ before or at the same time as it reaches B_v. The other gene copy continues to be passed on by the speciations leading to B_w, until it reaches $\Phi^*(w)$. Therefore, a single gene copy has to be lost for every speciation on the path from B_v to B_w except for B_w. The number of these gene losses is thus $|path_S(B_v, B_w)|-1$. Note that this number is the same if $\Phi^*(w)$ is an ancestor of $\Phi^*(v)$.

In the second case, both gene copies are passed on by x. Each copy has to reach its designated node $\Phi^*(v)$ or $\Phi^*(w)$. Therefore, one of the gene copies that isn't needed anymore, can be lost right after speciation x in both of the new species and the other preserved copy has to reach its designated node. The preserved copy is then lost after each speciation in the species that doesn't lead to the designated node. The total number of these gene losses is thus $(|path_S(x, B_v)| - 1) + (|path_S(x, B_w)| - 1) = |path_S(B_v, B_w)| - 1$.

To sum it up, if the node u is a duplication, we found out that both cases of mapping its children imply the same number of gene losses, specifically $2(|path_S(B_u, x)| - 1) + |path_S(B_v, B_w)|-1$. To compute this number, we would have to find $x = lca(\{B_v, B_w\})$, but finding the lca takes $\mathcal{O}(h)$ time.

We can avoid finding the lca due to the fact that $|path_S(B_u, x)|+|path_S(x, B_v)|-1 = |path_S(B_u, B_v)|$. Note that for both cases, the following holds: $|path_S(B_v, B_w)| = |path_S(x, B_v)| + |path_S(x, B_w)| - 1$. If we apply these equations to the total number of gene losses, the result is that $2(|path_S(B_u, x)| - 1) + |path_S(B_v, B_w)| - 1 = |path_S(B_u, B_v)| + |path_S(B_u, B_w)| - 2$. We can interpret this number by the following consideration. Right after the duplication u occurs, one of the two gene copies is designated to reach v and the other to reach w. If we look at the gene copy designated to reach v, it has to be lost in every speciation on the path from B_u to B_v, except for B_v. The same holds for the other gene copy, therefore, the number of gene losses implied by the mapping of u is $|path_S(B_u, B_v)| + |path_S(B_u, B_w)| - 2$. Node B_u is an ancestor of both B_v and B_w, thus the number of losses can be computed as $(h_S(B_u) - h_S(B_v) + 1) + (h_S(B_u) - h_S(B_w) + 1) - 2 = 2h_S(B_u) - h_S(B_v) - h_S(B_w)$.

Algorithm 4.1: A function that returns the number of duplications and losses DL_u implied by mapping δ of internal node u and its children.

```
1 def countDL(u ∈ V(G_o), δ: {u} ∪ children(u) → ℝ):
2     v, w = children(u)
3     losses = 2h_S(BN(δ(u))) − h_S(BN(δ(v))) − h_S(BN(δ(w)))
4     if isDup(u):
5         return (1, losses)
6     else:
7         return (0, losses − 2)
```

If node u of G is a speciation, it has to be mapped to $\Phi_{lca}(u)$. Children v and w have to be mapped under $\Phi_{lca}(u) = B_u = x$. Here, only one gene copy is passed on by x to each child, and this gene copy has to reach $\Phi^*(v)$ and $\Phi^*(w)$ in their respective lineages. The total number of losses is $|path_S(B_u, B_v)| + |path_S(B_u, B_w)| - 4 = 2h_S(B_u) - h_S(B_v) - h_S(B_w) - 2$ which is two less then the number of losses implied by the duplication, because the two gene losses right after x are not present here.

On the basis of all previous observations, we define function *countDL* (Alg. 4.1). For a node $u \in V(G_o)$ and the partial mapping δ, the function returns DL_u. The function may now seem more generalized then needed, but that's because we will use this function in the next section too. Finding the bearing nodes is in $\mathcal{O}(\log N)$ time. The lengths of paths $path_S(u, v)$, where u is an ancestor of v, are expressed by h_S and thus can be computed in constant time (with precomputed h_S). The time complexity of *countDL* is $\mathcal{O}(\log N)$.

Using *countDL*, we can now write a recursive function for finding DL_{Φ^*} (although only δ is needed) of the rooted gene tree reconciliation with exact branch lengths. When the algorithm recursively reaches $root(G_o)$, we are not necessarily done. If $root(G_o)$ is mapped below $root(S)$, we can potentially add more losses, based on the model of evolution we choose. If we allow that a new gene can be created (inserted) during the evolution of species in S, we are done. If we do not allow gene insertions, there has to be a gene entering $\Phi^*(root(G_o))$ and this gene has to go through $root(S)$. This means there has to be one gene loss for every node in $path_S(root(S), B(\Phi^*(root(G_o))))$ except for $B(\Phi^*(root(G_o)))$. The final algorithm for this case is shown as pseudocode in Alg. 4.2. The algorithm calls *countDL* for each node from G_o, so its time complexity is $\mathcal{O}(N \log N)$.

Algorithm 4.2: An algorithm for counting the total DL_{Φ^*} using only the partial reconciliation δ.

```
1 def totalDL():
2   return subtreeDL(root(G_o)) + (0, h_S(root(S)) − h_S(BN(δ(root(G_o)))))
3
4 def subtreeDL(u ∈ V(G)):
5   if u ∈ L(G):
6     return (0, 0)
7   else:
8     v, w = children(u)
9     return countDL(u, δ) + subtreeDL(v) + subtreeDL(w)
```

4.3 Set of solutions for inexact branch lengths

Here, we will deal with the problem of finding the minimal number of duplications and losses among all feasible reconciliatons, where G is rooted with interval branch lengths and S is rooted with exact branch lengths. We need to solve this problem in order to find the most parsimonious reconciliation for the variant, where G is unrooted with inexact branch lengths.

Let us assume that the *upward* and *downward* algorithms presented in Chapter 3 were run on G. Therefore, each node u from G has its possible mapping depth interval $X[u] = \langle X[u]_{min}, X[u]_{max} \rangle$ defined.

We will generally denote an interval of mapping depths as I, its supremum as I_{max} and infimum as I_{min}. We will consider closed, open and half-open intervals. By $\delta_I = (I_{min} + I_{max})/2$, we denote a representative mapping depth (partial reconciliation) implied by the interval I. The interval doesn't have to be closed, but the mean of its endpoints always belongs to I. We denote the minimal number of all duplications and losses in the subtree rooted at u implied by mapping u to δ_I as DL_I.

Our goal is to count DL_I for each subinterval I from $\langle X[root(G)]_{min}, X[root(G)]_{max} \rangle$. Every subinterval I from $X[u]$ of an arbitrary node $u \in V(G)$ needs to have the property that minimal DL of the subtree rooted at u is the same for any value of $\delta(u) \in I$. Also, we want all the subintervals of $X[u]$ to not overlap and the adjacent intervals to have different DL_I. Note that the subinterval with maximal depth has the minimal possible DL_I. While rising to lower depths, DL_I of subintervals will increase.

If $u \in L(G)$, its mapping depth interval $\langle D(\mu(u)), D(\mu(u)) \rangle$ cannot be divided into subintervals and its DL_I is $(0, 0)$. Let u be an internal node of G with a pos-

sible mapping interval $\langle X[u]_{min}, X[u]_{max}\rangle$. If u is mapped to depth $X[u]_{max}$ and $X[u]_{max} = D(\Phi_{lca}(u))$, u is a speciation. When the mapping depth decreases, u becomes a duplication. The number of losses changes (increases) only when the mapping reaches or surpasses a node from S. If u is a duplication, mapping exactly to a node $a \in V(S)$ results in mapping on the branch just underneath a, so the intervals representing these two mappings can be joined. The mapping depth interval $\langle X[u]_{min}, X[u]_{max}\rangle$ can thus be divided to non-overlapping subintervals $\langle X[u]_{min}, d_1\rangle\langle d_1, d_2\rangle..\langle d_k, X[u]_{max}\rangle$, where d_i are depths of consecutive nodes of $path_S(root(S), \Phi_{lca}(u))$ with depth at least $X[u]_{min}$ and at most $X[u]_{max}$. Each interval has DL_I lower than DL_I of the previous interval. For the splitting of intervals described in this paragraph, we define function $splitInterval(u)$ that returns an array of the searched subintervals. The algorithm passes through the nodes of $path_S(root(S), \Phi_{lca}(u))$, while creating a new subinterval for each node with the depth from interval $\langle X[u]_{min}, X[u]_{max}\rangle$. Its time complexity is thus $\mathcal{O}(h)$.

The subintervals of u have to be split further, depending on the subintervals of its children. For each mapping depth of u, we want to find the total minimal DL of the subtree rooted at u, considering the children's subintervals and the length of their parental edges.

Let children v and w have their arrays $subInt_v$ and $subInt_w$ of non-overlapping subintervals with minimal DL_I already precomputed. If a child is a leaf, it doesn't contribute in any way to the subintervals of u, since leaves have only one interval with DL_I equal to $(0, 0)$. If the child is an internal node, it has its own mapping subintervals with various DL_I. Let I be a mapping depth subinterval of child v. By $I^{shifted}$, we will denote a shifted interval, such that $I^{shifted}_{max} = I_{max} - w(u, v)_{min}$ and $I^{shifted}_{min} = I_{min} - w(u, v)_{max}$, where $w(u, v)$ is the parental edge of v. The shifted interval represents possible mapping depths for u, considering the child's mapping subinterval I.

If we shift non-overlapping subintervals, we get overlapping intervals. We want to minimize DL for every mapping depth of u, so if subintervals overlap, we prioritize the deepest of the subintervals with the lowest DL and the other intervals will have the intersection cut out. We define function $shiftIntervals(u, v)$ that returns an array of non-overlapping subintervals of $\langle X[u]_{min}, X[u]_{max}\rangle$ by processing the mapping subintervals of a child of u.

The algorithm starts with the deepest (lowest DL_I) subinterval I of child v, shifts I, and continues as follows. If the entire interval $I^{shifted}$ is deeper than $X[u]_{max}$, find the first higher interval I such that $X[u]_{max} \in I^{shifted}$ and set $I^{shifted}_{max}$ to $X[u]_{max}$. Add $I^{shifted}$ to the resulting array and cut the next higher shifted interval by setting its maximum to $I^{shifted}_{min}$. Since all edges are strictly positive, the new maximum of the

Algorithm 4.3: A pseudocode of function $shiftIntervals$ which returns an array of subsequent non-overlapping subintervals of possible mapping depths of u, considering mapping subintervals of node v, where u is a parent of v.

```
1  def shiftIntervals(u ∈ V(G), v ∈ children(u)):
2      shiftedInts = []
3      prevMin = X[u]_max
4      for (I ∈ subInt_v) :
5          I^shifted_min = I_min − w(u, v)_max
6          if X[u]_max < I^shifted_min: continue
7          if I^shifted_min < X[u]_min: I^shifted_min = X[u]_min
8          I^shifted = [I^shifted_min, prevMin]
9          DL^orig_{I^shifted} = DL_I
10         δ^orig_{I^shifted} = (I_max + I_min)/2
11         shiftedInts.add(I^shifted)
12         if I^shifted_min == X[u]_min: break
13         prevMin = I^shifted_min
14     return shiftedInts
```

higher shifted interval is always bigger than or equal to its minimum, so no interval will be cut out completely. This higher shifted interval will now be the next $I^{shifted}$ and will be added to the resulting array. This process continues until subinterval I, such that $X[u]_{min} \in I^{shifted}$ is reached. Set $I^{shifted}_{min}$ to $X[u]_{min}$, add $I^{shifted}$ to the results and return the array. For each shifted interval $I^{shifted}$, we keep the original $DL^{orig}_{I^{shifted}} = DL_I$ and $\delta^{orig}_{I^{shifted}} = \delta_I$, as it will be needed in the next algorithm. The pseudocode of $shiftIntervals$ is shown in Alg. 4.3.

Because the shifting and cutting is in constant time, the time complexity of this algorithm depends only on the number of subintervals of v. If we denote the maximal number of subintervals as $|subInt_{max}|$, this algorithm works in $\mathcal{O}(|subInt_{max}|)$ time. Since we want the subintervals to have different DL_I, we can estimate the number of subintervals $|subInt_u|$ of node $u \in V(G)$ by the maximal number of duplications and deletions implied by a mapping of a subtree of G rooted at u. The number of duplications is in $\mathcal{O}(N_u)$, where N_u is the number of nodes in the subtree rooted at u, as there is at most one duplication for each node in the subtree. If we look back at the function $countDL$, we can see that for each mapping of a node, there are $\mathcal{O}(h)$ losses. Therefore, there are maximally $\mathcal{O}(N_u + N_u h) = \mathcal{O}(N_u h)$ duplications and losses, and $|subInt_{max}| = \mathcal{O}(Nh)$.

After calling the functions $splitInterval(u)$, $shiftIntervals(u, v)$ and

$shiftIntervals(u, w)$, we get three arrays of subintervals, where the maximum of the deepest subinterval is $X[u]_{max}$ and the minimum of the highest subinterval is $X[u]_{min}$. Our goal is to create a function $countSubintervalDL()$ that merges the three arrays to one array and counts DL_I for each of its subinterval.

Let's name the deepest subintervals of $shiftIntervals(u, v)$, $splitInterval(u)$ and $shiftIntervals(u, w)$ as I^{left}, I^{middle} and I^{right}, respectively. The maximum of the new deepest mapping subinterval I of u is naturally $X[u]_{max}$. The minimum I_{min} will be set as the maximum of I^{left}_{min}, I^{middle}_{min} and I^{right}_{min}. There can be more that one interval from $\{I^{left}, I^{middle}, I^{right}\}$ with its infimum equal to I_{min}. Let's denote this set of intervals that were used completely as CI.

As for DL_I, we will call our function $countDL(u, \delta) = DL_u$, where δ is a specific mapping depth of u, v and w. In our case, we take $\delta(u) = \delta_I = (I_{max} + I_{min})/2$, $\delta(v) = \delta^{orig}_{I^{left}}$ and $\delta(w) = \delta^{orig}_{I^{right}}$. The sum $DL_u + DL^{orig}_{I^{left}} + DL^{orig}_{I^{left}}$ is the searched DL_I which is the total minimal number of duplications and losses in a subtree rooted at u, where u is mapped to a depth from I.

After we compute DL_I, we will set each interval from CI to its subsequent, less deep interval from its respective array, and repeat the process above, until CI contains all three highest subintervals I^{left}, I^{middle} and I^{right} which have their minimum equal to $X[u]_{min}$. Note that by induction, each subinterval of $subInt_u$ has different DL_I because both $shiftIntervals$ (the subintervals of children are shifted after running $countSubintervalDL$ on them) and $splitInterval$ produce arrays of intervals with different DL_I.

The function $countSubintervalDL()$ is shown in Alg. 4.4. In the main $do - while$ cycle, we gradually pass through the subintervals of arrays $splitIntervals(u)$, $shiftIntervals(u, v)$ and $shiftIntervals(u, w)$ which have their respective sizes in $\mathcal{O}(h)$, $\mathcal{O}(N_v h)$ and $\mathcal{O}(N_w h)$. For each of these $\mathcal{O}(h + N_v h + N_w h) = \mathcal{O}(N_u h)$ subintervals, we call the function $countDL$ which works in $\mathcal{O}(\log N)$ time. The time complexity without the recursive calls to children (lines $9 - 10$ in Alg. 4.4) is thus $\mathcal{O}(Nh \log N)$. Calling the function for every node thus results in the running time $\mathcal{O}((\sum_{u \in V(G)} N_u) h \log N)$.

If S is balanced, $\mathcal{O}(h) = \mathcal{O}(\log N)$ and if G is balanced, $\mathcal{O}(\sum_{u \in V(G)} N_u) = \mathcal{O}(N \log N)$. The time complexity of $countSubintervalDL$ in the case S and G are balanced is then $\mathcal{O}(N \log^3 N)$. However, G and S don't have to be balanced and thus, in the general case, the time complexity is $\mathcal{O}(N^3 \log N)$.

Algorithm 4.4: An algorithm for finding the array of non-overlapping subintervals of a node u, such that each subinterval I represents the depths where u can be mapped with the minimal DL_I. Function $next(A)$ returns the next element of array A.

```
1  def countSubintervalDL(u ∈ V(G)):
2      if u ∈ L(G):
3          I = ⟨D(μ(u)), D(μ(u))⟩
4          DL_I = (0, 0)
5          subint_u = [I]
6          return
7
8      v, w = children(u)
9      countSubintervalDL(v)
10     countSubintervalDL(w)
11     splitInts = splitIntervals(u)
12     shiftedInts_v = shiftIntervals(u, v)
13     shiftedInts_w = shiftIntervals(u, w)
14     I^left = shiftedInts_v[0]
15     I^middle = splitInts[0]
16     I^right = shiftedInts_w[0]
17     I_max = X[u]_max
18     do :
19         I_min = max(I^left_min, I^middle_min, I^right_min)
20         DL_u = countDL(u, δ_I, δ^orig_I^left, δ^orig_I^right)
21         DL_I = DL_u + DL^orig_I^left + DL^orig_I^left
22         subint_u.add(I)
23         if I^left_min == I_min: I^left = next(shiftedInts_v)
24         if I^middle_min == I_min: I^middle = next(splitInts)
25         if I^right_min == I_min: I^right = next(shiftedInts_w)
26         I_max = I_min
27     while I^left_min, I^middle_min, I^right_min ≠ X[u]_min
```

4.4 The most parsimonious reconciliation

In this section, we will use the previous knowledge from this chapter to find the most parsimonious reconciliation for the case when G is unrooted with inexact branch lengths and S is rooted with exact branch lengths. In Chapter 3 we have presented an algorithm for finding a possible solution for this variant. However, the algorithm doesn't necessarily produce the most parsimonious reconciliation.

Let's assume that G is semi-rooted at q, with q at an unknown location at the edge (u, v). This edge separates two trees with roots u and v. As a result of solving the constant-size LP and the *upward* and *downward* algorithms from the previous chapter, each node $u \in V(G)$ has its mapping depth interval $X[u]$ specified.

We can always map the root q to its maximal depth ($\delta(q) = X[q]_{max}$), since it minimalizes DL and also the remaining length of the root edge remains at its maximum. When q is mapped to $X[q]_{max}$, the linear program ensures that the maximal length of the root edge can reach $X[u]_{max}$ and likewise it can reach $X[v]_{max}$. However, both $X[u]_{max}$ and $X[v]_{max}$ are not necessarily reached at the same time. Our goal is to choose mapping depths of u and v such that the total DL_{Φ^*} of the reconciliation (G_o, S_o^*, Φ^*) is minimal and to count DL_{Φ^*}.

After we set the mapping depth of q to $X[q]_{max}$, it is convenient for us to recalculate $X[u]$ and $X[v]$ because their minimal values are now not necessarily reachable. For example, we can solve the same LP again, but instead of three variables we will have two, because we will set X_q to $X[q]_{max}$. By minimizing the objective functions X_u and X_v of this LP, we will compute $X[u]_{min}$ and $X[v]_{min}$.

Let's call the function *countSubintervalDL* on nodes u and v. Their respective intervals $X[u]$ and $X[v]$ are thus split to subintervals. Each subinterval I represents a possible mapping depth of the respective node, with a certain distinct minimal number of duplications and deletions DL_I of the subtree rooted at the node. Mapping the nodes to depths from the deepest subintervals results in the smallest number of duplications and losses.

Our algorithm will go through all subintervals of u and find the deepest possible subinterval of v such that the edge length of (u, v) can reach both u and v. We will denote the current mapping depths of u and v as δ_u and δ_v and set δ_u to $X[u]_{max}$. Let's start in the deepest subinterval I^u of u. If u is mapped to depth $\delta_u = I^u_{min}$, DL_I remains the same, and an additional part of the length of (u, v) can be used to maximize the depth of v. Then we use our LP again, with only one variable, as we can set X_q to $X[q]_{max}$ and X_u to δ_u. By maximizing the objective function X_v, we will also compute our desired δ_v. To find mapping depth subinterval I^v of v that contains δ_v, we can use binary search on the subintervals of v, since they are sorted by depth. The

number of events DL_{min} with these particular mapping depths ($\delta(u) = \delta_u$, $\delta(v) = \delta_v$, $\delta(q) = X[q]_{max}$) can be evaluated as $countDL(q, \delta) + DL_{I^u} + DL_{I^v}$. We will remember DL_{min}, δ_u and δ_v, and we will set I^u as the next subinterval above the previous I^u. This procedure will continue until such I^u is reached that $I^u_{min} = X[u]_{min}$. Then will choose the desired mapping depths δ_u and δ_v that imply the lowest DL_{min}.

This algorithm iterates through $\mathcal{O}(Nh)$ subintervals of u, for each subinterval solves LP of constant size, finds the deepest subinterval of v by binary search in $\mathcal{O}(\log(Nh)) = \mathcal{O}(\log N)$ time, and counts the number of duplications and deletions implied by mapping of q by running $countDL$. The time complexity of this algorithm is thus $\mathcal{O}(Nh \log N)$.

After we have found the best plausible $\delta_u = \delta(u)$ and $\delta_v = \delta(v)$ from subintervals I^u and I^v, we can find the partial reconciliation δ with the maximal possible mapping depths of the nodes in the subtrees rooted at u and v. Each node $y \in V(G) \setminus \{u, v, q\}$ with parent x will have its $\delta(y)$ set to $min(\delta(x) + w(x, y)_{max}, X[y]_{max})$.

The total DL_{Φ^*} is the sum of the minimal DL_{min} we have previously found in our algorithm and the losses on the path from $root(S)$ to $B(\Phi^*(q))$ (if we do not allow gene insertions).

The algorithm for choosing the best subintervals has lower time complexity than $countSubintervalDL$, so the overall running time of finding the most parsimonious reconciliation of a semi-rooted tree is $\mathcal{O}(Nh^2 \log N)$ (in the general case $\mathcal{O}(N^3 \log N)$). Finding the most parsimonious reconciliation of all semi-rooted trees of an unrooted G thus has running time $\mathcal{O}(N^4 \log N)$ in the general case.

Implementation

We have implemented the algorithm for the most parsimonious reconciliation of unrooted gene tree with interval branch lengths, described in this chapter, in programming language Java. Our implementation is not as efficient as described here because we did not use the *level ancestor* data structure, therefore it runs in $\mathcal{O}(N^5)$ time.

The program takes multiple unrooted gene trees as a part of the input (along with the species tree S) and transforms them to gene tree G with interval branch lengths, so that our algorithm can be ran. More details can be found in the enclosed source code.

Conclusion

In this thesis, we have presented polynomial-time algorithms for multiple variants of the isometric reconciliation problem, where the edge lengths of the input trees are not known exactly, but are rather given as intervals of possible values. Our algorithms represent an important step towards the practical applicability of the isometric reconciliation, since the exact edge lengths of reconstructed phylogenetic trees are not known in practice.

Except for the general algorithm based on linear programming, our algorithms solve the case where the species tree is rooted with exact edge lengths and the intervals of edge lengths are considered only in the gene tree. If the gene tree is rooted, our $\mathcal{O}(N)$ algorithm is even more efficient than the existing algorithms for exact edge lengths, if only partial reconciliation is required. Our algorithms are able to find the most parsimonious reconciliation among the feasible solutions for both rooted and unrooted gene trees. If the gene tree is unrooted, we provide an algorithm that finds all feasible solutions in $\mathcal{O}(N^2)$ time, but finding the most parsimonious reconciliation has a running time $\mathcal{O}(N^4 \log N)$ in the worst case.

The natural next step related to our work is to develop efficient algorithms for isometric reconciliation, where the gene tree has interval edge lengths, and the species tree is unrooted with exact edge lengths. One option is to go through all semi-rooted forms of the species tree and make use of the algorithms we have presented in this thesis. However, there may be a more efficient solution than iterating through all semi-rooted trees. This also applies to our algorithm for an unrooted gene tree.

Other open problems are more efficient algorithms the case, where the interval edge lengths are considered in the species tree too. A good way to start might be to first analyze the case where the gene tree has exact edge lengths. Then by using this knowledge and the information provided in this thesis, it might be possible to develop efficient algorithms for the case when both input trees have interval branch lengths.

Brejová et al. [3] have introduced a variant, where all edge lengths are exact, but are also scaled by some unknown factor $\beta > 0$. This approach can be combined with the interval lengths and the scaling factor β could be incorporated to our algorithms

too.

The inaccuracy of branch lengths does not have to be necessarily represented as an interval for each edge length. Instead, lengths can be given as a single number w and allowed to be deviated by a small amount. For example, each edge length would ultimately be from an interval $\langle w(1-\epsilon), w(1+\epsilon)\rangle$, where $\epsilon > 0$ is an unknown global relative error. The goal would be to minimize the value of ϵ, so that the reconciliation exists.

Bibliography

[1] Xenopus database. `http://www.xenbase.org/anatomy/intro.do`. Accessed: 16.04.2019.

[2] Mukul S Bansal, Eric J Alm, and Manolis Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012.

[3] Broňa Brejová, Askar Gafurov, Dana Pardubská, Michal Sabo, and Tomáš Vinař. Isometric gene tree reconciliation revisited. *Algorithms for Molecular Biology*, 12(1):17, 2017.

[4] Jean-Philippe Doyon, Cedric Chauve, and Sylvie Hamel. Algorithms for exploring the space of gene tree/species tree reconciliations. In *RECOMB International Workshop on Comparative Genomics*, pages 1–13. Springer, 2008.

[5] Jean-Philippe Doyon, Celine Scornavacca, K Yu Gorbunov, Gergely J Szöllősi, Vincent Ranwez, and Vincent Berry. An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers. In *RECOMB International Workshop on Comparative Genomics*, pages 93–108. Springer, 2010.

[6] Oliver Eulenstein. A linear time algorithm for tree mapping. 1997.

[7] Joseph Felsenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

[8] Mark B Gerstein, Can Bruce, Joel S Rozowsky, Deyou Zheng, Jiang Du, Jan O Korbel, Olof Emanuelsson, Zhengdong D Zhang, Sherman Weissman, and Michael Snyder. What is a gene, post-encode? history and updated definition. *Genome research*, 17(6):669–681, 2007.

[9] Morris Goodman, John Czelusniak, G William Moore, Alejo E Romero-Herrera, and Genji Matsuda. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Biology*, 28(2):132–163, 1979.

[10] Pawel Górecki, Gordon J Burleigh, and Oliver Eulenstein. Maximum likelihood models and algorithms for gene tree evolution with duplications and losses. *BMC bioinformatics*, 12(1):S15, 2011.

[11] Roderic Guigo, Ilya Muchnik, and Temple F Smith. Reconstruction of ancient molecular phylogeny. *Molecular phylogenetics and evolution*, 6(2):189–213, 1996.

[12] Leonid G Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[13] Liang Liu and Dennis K Pearl. Species trees from gene trees: reconstructing bayesian posterior distributions of a species phylogeny using estimated gene tree distributions. *Systematic biology*, 56(3):504–514, 2007.

[14] Jian Ma, Aakrosh Ratan, Brian J Raney, Bernard B Suh, Webb Miller, and David Haussler. The infinite sites model of genome evolution. *Proceedings of the National Academy of Sciences*, 105(38):14254–14261, 2008.

[15] Bengt Sennblad and Jens Lagergren. Probabilistic orthology analysis. *Systematic biology*, 58(4):411–424, 2009.

[16] Pawel Tabaszewski, Pawel Górecki, and Oliver Eulenstein. Phylogenetic consensus for exact median trees. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 366–375. ACM, 2018.

[17] Louxin Zhang. On a mirkin-muchnik-smith conjecture for comparing molecular phylogenies. *Journal of Computational Biology*, 4(2):177–187, 1997.

[18] Christian M Zmasek and Sean R Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, 17(9):821–828, 2001.

Appendix

The DVD enclosed to this thesis contains the source code of the implementation of our algorithms.