COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

SOFTWARE FOR ISOMETRIC GENE TREE RECONCILIATION

Master's thesis

2020

Bc. Dominika Mihálová

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

SOFTWARE FOR ISOMETRIC GENE TREE RECONCILIATION

Master's thesis

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

Bratislava, 2021

Bc. Dominika Mihálová

51059667

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

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

Názov: Software for isometric gene tree reconciliation
Softvér pre izometrickú rekonciliáciu génových stromov

Anotácia: Izometrická rekonciliácia génových stromov je výpočtový problém, v ktorom je cieľom identifikovať zodpovedajúce si vrcholy v dvoch stromoch, z ktorých jeden reprezentuje evolučnú históriu skupiny organizmov a druhý reprezentuje evolučnú históriu jedného génu v rámci týchto organizmov. Cieľom práce je implementovať praktický softvér na rekonciliáciu génových stromov a experimentálne otestovať jeho presnosť na simulovaných aj reálnych biologických dátach.

Vedúci:	doc. Mgr. Bronislava Brejová, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	14.10.2019

Dátum schválenia: 29.11.2019 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

....

študent vedúci práce

Abstrakt

Cieľom diplomovej práce bolo implementovať softvér pre izometrickú rekonciliáciu génových stromov a experimentálne otestovať jeho presnosť na simulovaných aj reálnych biologických dátach. Práca je rozdelená do piatich kapitol.

Prvá kapitola je venovaná základnej terminológií z oblasti bioinformatiky a prehľadu rôznych prístupov k riešeniu problému rekonciliácie s názornými riešeniami tejto problematiky v podobe softvérov.

Ďalšia časť uvádza algoritmy a implementovaný softvér. Opisujú sa vytvorené algoritmy, špecifikujú sa vlastnosti implementovaného softvéru, spôsob spracovania vstupov a následné výstupy.

Štvrtá kapitola prezentuje testovaciu sadu a opisuje vykonané experimenty na implementovanom softvéri pre izometrickú rekonciliáciu.

Záverečná kapitola sa zaoberá interpretáciou výsledkov testovania implementovaného softvéru.

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

Abstract

The main goal of the diploma thesis was to implement software for isometric gene tree reconciliation and to experimentally evaluate its accuracy on simulated and real biological data. The thesis is divided into five chapters.

The first chapter presents the basic terminology in the field of bioinformatics and an overview of different approaches to solving the problem of reconciliation with concrete solutions to this problem in the form of software.

The next section presents algorithms and implemented software. The created algorithms are described, the properties of the implemented software, the method of input processing and subsequent outputs are specified.

The fourth chapter presents a test set and describes the experiments performed on the implemented software for isometric reconciliation.

The final chapter deals with the interpretation of the results of testing the implemented software.

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

List of Figures

Contents

Introduction

1 Overview

In this chapter, we introduce basic information and define essential terminology from the field of bioinformatics. We describe different ways for solving the problem of gene tree reconciliation with examples of existing software in more details.

1.1 Background

Every organism has its complete set of genetic information encoded in a genome. A genome consists of several DNA (deoxyribonucleic acid) molecules and contains all the information, which are required for the organism to function.

DNA is a long molecule composed of two complementary strands. Each strand is made up of four chemical bases: adenine, guanine, cytosine and thymine, and connected to its complementary strand by pairing rules, where adenine is paired up with thymine and cytosine is paired up with guanine. The sequence of these bases encodes the genetic information important for building and maintaining an organism. Specific parts of DNA are called genes.

A gene is a subsequence of a DNA strand that contains information for the synthesis of a specific molecule, usually a protein. It is a basic unit of heredity.

The DNA sequence of a gene can be altered by mutation. It is a process that allows small changes in the DNA of organisms that refers to differences between individuals within a population. We will be working with two types of mutations: duplication and gene loss.

Duplication is a type of mutation where one or more genes are copied and inserted to some other position in the same genome. A duplicated gene sometimes develops a new function [9].

The opposite of duplication is gene loss (deletion). It is a type of mutation in which some part of a DNA sequence containing a gene is left out from the genome during DNA replication or it loses its function.

Speciation is an evolutionary process of in which a single population evolves populations into two distinct species. It can happen for various reasons, for example, when a group separates from other members of its species to a different geographical area. Members of a new group develop their own unique characteristics due to the demands of another environment and this process will differentiate the new species.

Duplications and speciations result in the formation of groups of similar genes, called gene families, from a single gene. A gene family consists of evolutionarily related genes from one or multiple species, which are structurally and usually functionally similar.

Evolutionary relationships formed by evolutionary events are represented in a form of graph called a phylogenetic tree, which is a branching diagram that shows evolutionary relationships between organisms. A phylogenetic tree is a tree T with nodes $V(T)$, edges $E(T)$ and leaves $L(T)$. It is called weighted when branch length $w(u, v)$ is defined for each edge (u, v).

Phylogenetic trees can be either rooted or unrooted (Fig. 1.1). A rooted tree is a phylogenetic tree T where for $(u, v) \in E(T)$: node u is the parent of node v, node v is the children of node u, $root(T)$ does not have parent and leaves $L(T)$ do not have children. An ancestor of node v is any node of tree T on the path from node v to $root(T)$. Every ancestor have at least one descendant. An descendant of node v is any node of tree T of which v is an ancestor [12]. We will denote for $u, v \in V(T)$ that $v <_T u$ if u is ancestor of v and v is descendant of u.

Every rooted tree has a height, which symbolizes the longest path. It is the number of nodes between the $root(T)$ and one of the leaves.

Nodes in rooted trees have levels. The level of node $l(u)$ is the number of ancestors on the path from the node u to the $root(T)$. The root of a tree $root(T)$ has level 0, since it has no parents.

If a rooted tree is weighted, nodes in the tree have depths. The depth of node u, $D(u)$, is the sum of the lengths of all edges between node u and the $root(T)$ of a rooted tree.

For a group of nodes in a rooted tree, their lowest common ancestor (LCA) is the farthest node from the root that has all nodes in the group as descendants.

An unrooted tree is a phylogenetic tree without root. Unrooted tree can be rooted by placing a root r on some edge (u, v). The original edge (u, v) is subdivided into two edges (u, r) and (r, v). If edge (u, v) is weighted, then $w(u, r) + w(r, v) = w(u, v)$.

A special type of an unrooted tree is a semi-rooted tree, where we presume the new root of an unrooted gene tree G is positioned on edge $(u, v) \in E(G)$ and subtrees are rooted at nodes u and v.

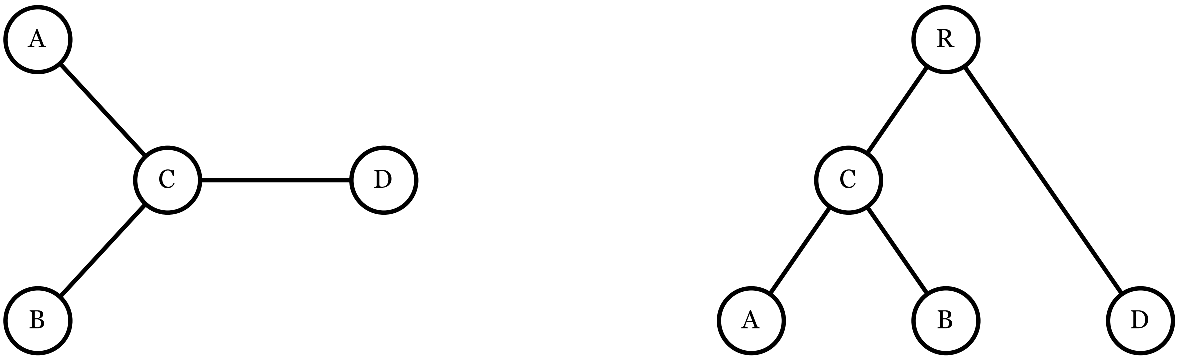

Figure 1.1: Unrooted tree(left) and its rooted version (right) tree. We placed the root R on the edge (C, D) replacing it with two new edges (C, R) and (R, D).

We will be using two types of phylogenetic trees for showing evolutionary relationships: species trees (to describe the evolution of a set of species) and gene trees (to describe the evolution of a particular gene).

A species tree is a phylogenetic tree S where $L(S)$ represent present-day species and internal nodes from $V(S)$ represent speciation events in the history.

A gene tree is a phylogenetic tree G where $L(G)$ represent present-day copies of the gene and internal nodes from $V(S)$ represent duplication and gene loss events in the history.

Phylogenetic trees are reconstructed from a multiple alignments of the DNA sequences of present-day species by various methods [10] to find the most likely phylogenetic tree to given DNA sequences.

1.2 Different approaches to gene tree reconciliation

Evolutionary history is a possible sequence of evolutionary events that lead to observed members of a gene family in present-day species (Fig. 1.2). It illustrates how many duplications and gene losses happened during the evolution of one or more genes inside the evolution of a group of species.

The problem of gene tree and species tree reconciliation was introduced in 1979 by Goodman et al. [11] as a method to infer the evolutionary history of duplications and gene losses in a gene family to decode evolutionary relationships between copies of a gene. The goal of reconciliation consists in mapping nodes of a gene tree into a species tree and thus inducing the evolution of a gene family in terms of speciations, duplications and gene losses. An important prerequisite for reconciliation is to have a gene tree without errors as misplaced leaves can lead to a different history of the gene family.

Definition 1 *A reconciliation between gene tree G and species tree S is mapping $\phi : V(G) \rightarrow V(S)$ such that:*

1. $\forall u \in L(G) : \phi(u) = \mu(u)$
2. $\forall u, v \in V(G)$ *such that* $v <_G u$: $\phi(v) <_S \phi(u)$

An example of gene tree reconciliation is shown in Figure 1.2. We are given the gene tree G, the species tree S and a leaf mapping $\mu : L(G) \rightarrow L(S)$ that maps each leaf from G to leaf of its species in S. We will map internal nodes according to the second condition in Definition 1. The node d is mapped to node Y, because it has $\phi(d)$ and $\phi(c)$ as descendants. It cannot be mapped to node X since node X does not have $\phi(c)$ as descendant thus $\phi(c) <_S \phi(d)$ would not hold. Then node e is mapped above node Y to have $\phi(a)$ and $\phi(d)$ as descendants.

The LCA-mapping $\sigma : V(G) \rightarrow V(S)$ maps each node $u \in V(G)$ as low as possible to the unique node $\sigma(u) = LCA(\mu(v) | \forall v \in L(G), v <_G u)$ in S. It satisfy both conditions in Definition 1 and minimize the number of duplications and gene losses. This reconciliation can be found in linear time [12].

In this work, we divide approaches to reconciliation into three types: scoring, probabilistic and isometric. Every one of these approaches has its way to compute the gene tree reconciliation. They will be described with presented examples of software that have implemented gene tree reconciliation.

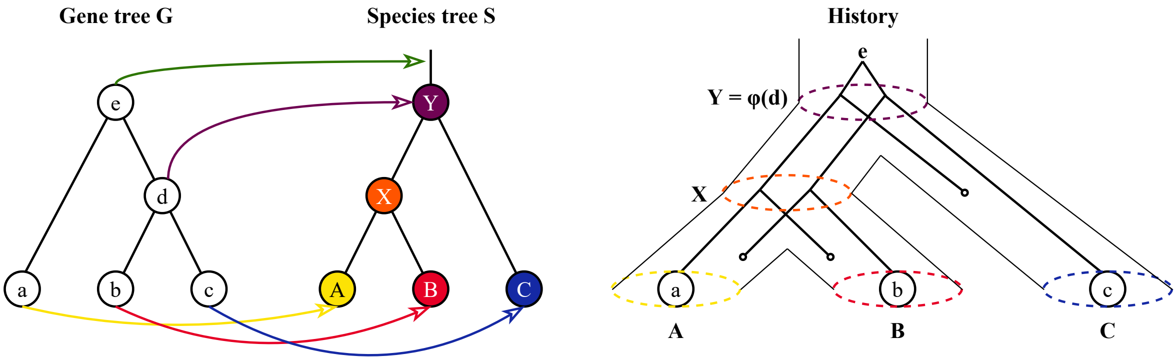

Figure 1.2: Reconciliation and evolutionary history. On the left, the gene tree G is mapped to the species tree S. On the right, we can see the evolutionary history implied by this reconciliation. This history contains one duplication e, two speciations Y and X and three gene losses (empty circles).

1.2.1 Scoring gene tree reconciliation

One of the known approaches to find the best reconciliation is to minimize the duplication-loss score, which signifies the sum of duplications and gene losses during the reconciliation. Various software for scoring gene tree reconciliation are known such as TreeBeST, TreeFix, Treerecs or Notung.

TreeBeST

Software TreeBeST [13] takes a rooted species tree and multiple sequence alignment of gene trees for the gene family as input. It uses a method to merge various input gene trees into one gene tree with a model to penalize duplications and gene losses relative to a known species tree. The method first resolves the topology of a gene tree with five methods: neighbour-joining synonymous distance, non-synonymous distance, p-distance and max-likelihood under the WAG and the HKY model. Then the topology is bootstrapped 100 times. The output of the software is one rooted tree.

The TreeBeST reconciliation method was then compared with PhyML+RAP method, where multiple sequence alignment of gene trees are reconciled to one gene tree without the presence of species tree [19]. To evaluate these two methods, authors developed a duplication consistency score represented by $\frac{intersection}{union}$ of species between left and right branches. Low duplication consistency score means poor topology of the resultant gene tree. PhyML+RAP approach leads to more duplication nodes that TreeBeST and the supplementary duplication nodes made by PhyML+RAP had a low duplication consistency score. Authors found this result unexpected as TreeBeST uses the species tree and tends to produce duplication when the gene tree has extensive extant members on each side of the duplication.

TreeFix

TreeFix [1] takes a rooted species tree, a maximum likelihood gene tree and a multiple sequence alignment of gene trees for the gene family as input. It infers duplications and gene losses using maximum parsimony reconciliation with a duplication-loss cost function, which looks for the reconciliation with the minimum total number of duplications and gene losses. Duplication (D) and gene loss (L) have their costs (c_D and c_L) that are set to one by default and can be changed by the user. The duplication-loss cost for one reconciliation can be written as: $c_D \cdot D + c_L \cdot L$.

The main method [20] uses a hill-climbing search strategy to find an optimal rooted gene tree with the statistically equivalent likelihood to the given maximum likelihood gene tree and with minimum duplication-loss cost as output. Basics of the method are to compute duplication-loss cost and perform neighbour interchange and subtree prune and regraft on the current optimal gene tree. After finding proposals, it chooses the proposal with the lowest duplication-loss cost and with the statistically equivalent likelihood to the given maximum likelihood gene tree. This process is repeated for a given number of iterations.

Authors compared TreeFix with RAxML, SPIMAP, TreeBeST, Notung and tt using simulated and real datasets of two types of species: 12 Drosophila and 16 fungi [21]. The software was judged from the point of view of phylogenetic accuracy in 5 categories (topology, branch, orthologs, duplications, losses) and runtime. TreeFix and SPIMAP show the best accuracy in all 5 categories, Notung has slightly worse accuracy in reconstructing the topology of fungi and precision of inferring duplication and gene losses. tt has problems in the same categories as Notung. The worst accuracy demonstrate RAxML and TreeBeST. While TreeFix and SPIMAP have great phylogenetic accuracy their average running time is longer than others. The best runtime has Notung followed by tt and RAxML.

Treerecs

Software Treerecs [14] takes a rooted species tree, one or more rooted or unrooted gene trees and a mapping of gene tree leaves to species tree leaves. It provides reconciling gene tree within the associated species tree minimizing the duplication and gene loss score and rooting the gene tree along the way if needed. The output is, depending on the input, one or more rooted gene trees.

The main aim of the authors was to create a more efficient software. They compared Treerecs with EcceTERA, Notung and Ranger-DTL in 3 categories: root (find the root that minimizes duplication-loss score), correction and root+correction (do both previous categories at the same time). In the first category, Treerecs is better than Ranger-DTL and Notung, which shows a large increase in execution time as the number of leaves increases. Treerecs show the best performance in the correction of trees followed by Notung, while Ranger-DTL has big execution time even with a small increase of leaves. The last category is only supported by Treerecs and EcceTERA, where Treerecs has also better performance.

Notung

Notung [8] takes a rooted species tree, a rooted or unrooted gene tree and the leaf mapping as input. If the gene tree is not rooted, it can be rooted by Notung rooting mode that gives each edge a root score (weighted sum of duplications and gene losses). Apart from a reconciliation of binary trees, Notung can reconcile binary gene trees with non-binary species trees and non-binary gene trees with the binary species tree. The non-binary tree is a tree with at least one polytomy (a node with more than two children). Reconciliation of binary gene trees to non-binary species trees results into binary gene tree.

They use an algorithm, that can distinguish between duplication and deep coalescence (divergence, when the time of separation of two lineages precede the time of speciation) and leads to the smaller total number of duplications and losses than duplication-loss cost function used in reconciliation of two binary trees [18]. The duplication-loss cost function is the same as in TreeFix. Reconciliation of non-binary gene tree to binary species tree results into non-binary gene tree. The general approach is to convert the non-binary gene tree to binary gene tree that has minimal duplication-loss score when reconciled with the binary species tree. The resolution is then rearranged back to non-binary gene tree, where all nodes and edges not present in the original gene tree are removed and their assigned duplications and gene losses are reassigned to their polytomy.

1.2.2 Probabilistic gene tree reconciliation

Probabilistic methods have been designed to increase the accuracy of reconciled trees. We introduce two software tools that use probabilistic methods to reconcile gene trees: SPIMAP and Phyldog.

SPIMAP

SPIMAP software [16] takes a rooted species tree and multiple gene sequences of species from the species tree. Normally, the gene sequences are compared and clustered according to their similarity, which results in a set of homologous gene families. Each gene family has its multiple sequence alignment that is reconstructed into gene trees and they are reconciled with the known species tree. Into this classic pipeline, SPIMAP inserts a parameter estimation model using Bayesian approach creating a new phylogenomic pipeline. It learns duplication, gene loss rates during clustering and gene, species substitution rates during the process of alignment. These parameters are then used while building and reconciling the gene tree with the known species tree. The output is a special reconciliation file format that contains gene node ID, species node ID and evolutionary event that occurred on a given node.

The parameter estimation model [17] infers duplication and gene loss rate using the birth-death process. The birth-death process is a continuous-time process that generates a gene tree according to the constant birth rate (representing duplication) and death rate (representing gene loss). After running it for a time that represents branch length, all branches that exist at the time are "surviving" and others are "extinct". If a node has no surviving descendants, it is called "doomed". Every branch has its length, which can be written as $\frac{substitutions}{site}$ or a product of a duration of time and a substitution rate. The substitution rates signify the number of substitutions per site per unit of time. The model computes gene-specific rate (measures all rate in a tree) for every gene family and species-specific (specifies rate to given branch in the gene tree) rate for every branch.

To determine if the new phylogenomic pipeline improved accuracy, authors compare SPIMAP with PrIME-GSR, SPIDIR, MrBayes, PHYML, BIONJ, RAxML and SYNERGY. They used the same data as TreeFix: 12 Drosophila and 16 fungi. Firstly, they measured the average runtime. The best runtime, under 1 minute, have RAxML, MrBayes, PhyML and BionJ, which was the fastest method. With the same amount of iterations, SPIMAP was quicker than SPIDIR or PrIME-GSR, which was the slowest method. Next, they decided to apply the duplication consistency score (used in TreeBeST) to determine method with better accuracy. The smallest number of duplication with low duplication consistency score have SPIMAP and SYNERGY. The moderate performance shows PrIME-GSR and SPIDIR. Remaining four methods have a similar number of duplication with low duplication consistency score. Lastly, they evaluate phylogenetic accuracy depending on 5 categories (used in TreeFix) for 6 above-mentioned methods (except RAxML and SYNERGY). SPIMAP has higher accuracy in every category. In the category of inferring the topology, PrIME-GSR has slightly worse accuracy while SPIDIR, MrBayes, PHYML, BIONJ shows bad accuracy in the topology of fungi dataset. The accuracy of reconstructed branches is better than the topology in every method, SPIMAP and PrIME-GSR are first two. SPIDIR, MrBayes, PHYML, BIONJ are a little worse at the sensitivity of detection orthologs in fungi dataset. They have the biggest problem with the precision of inferring the duplications in fungi dataset and losses in both datasets. The same problem has also PrIME-GSR, but only in precision of inferring losses.

PhylDog

Another software is PhylDog [2] takes multiple gene alignments, a mapping between gene names and species names, and a list of species names as input. The method infers species tree, gene trees, duplication and gene loss rates by maximizing the probability of alignments overall gene families composed from the likelihood of a phylogeny given an alignment and the likelihood of the reconciliation of a gene tree with a species tree according to duplication and gene loss rates. This method uses the birth-death process and is similar to SPIMAP, but they differ in two aspects. First, while SPIMAP assumes duplication and gene loss rates to be constant for all branches in the species tree, PhylDog chooses to use a particular pair of duplication and gene loss rates to each branch of the species tree. Second, SPIMAP requires time-anchored species tree (branch length shows the amount of time between two nodes) to compute the likelihood of a gene family. Alternately, PhylDog calculates likelihood from the expected numbers of duplications and gene losses. The output is reconciled gene trees.

PhylDog was compared with TreeBeST and PhyML [3] in terms of the number of duplications and the reconstructed ancestral genome size. PhyML has the biggest number of predicted duplication events, TreeBeST reconciled trees with a much smaller number, but still much higher than PhylDog. The same order of software was also in the number of reconstructed ancestral genome size, where both PhyML and TreeBeST have bigger ancestral genomes that lead to deeper nodes in the species tree.

1.2.3 Isometric gene tree reconciliation

Another variant of reconciliation is isometric gene tree reconciliation, where both species tree S and gene tree G has known branch lengths. These branch lengths are taken into account while mapping a gene tree G to a species tree S. The output of isometric reconciliation is reconciled gene tree with preserved evolutionary distances. The branch lengths of phylogenetic tree express estimated time between evolutionary events. The time can signify the actual geological time, amount of evolutionary changes that happened on the edge or expected number of substitution per site between two nodes.

1.2.3.1 Exact branch length

This problem was introduced and named by Ma et al. [15] in 2008 for the first time. They defined isometric reconciliation of rooted species tree and unrooted gene trees, where all input trees have exact branch lengths. The algorithm executes all input gene trees one by one. First of all, it maps all leaves from the gene tree to leaves in the species tree. Then it takes an unmapped node, which has to be connected with at least 2 already mapped nodes, and call function, that map the unmapped node into the species tree and root the gene tree. The presented algorithm had $O(N^2)$ running time, where N stands for the total number of nodes in the gene tree and the species tree. However, their definition of isometric reconciliation has some flaws since it does not preserve all evolutionary distances between nodes as it allows them to develop a reconciliation that does not satisfy any history of evolution.

As a result, Brejová et al. [4] later corrected and modified algorithm by Ma et al. to a more efficient algorithm with $O(N \log N)$ running time. Their modify algorithm firstly maps every leaf from gene tree to the species tree. Next, it maps all unmapped nodes to the species tree. After all nodes are correctly mapped, the algorithm roots the gene tree and maps the found root to the species tree. Eventually, it verifies if the reconciled tree is correct due to the definition of isometric reconciliation. They also proposed two extensions of the problem. In the first extension, they considered both input trees (gene tree and species tree) to be unrooted and designed an algorithm with $O(N^5 \log N)$ running time. The second extension presents an algorithm, where both input trees are rooted, but gene tree branch lengths are assumed to be scaled by an unknown scaling factor.

Definition 2 *An isometric reconciliation between gene tree G and species tree S with exact branch lengths w, which are strictly positive, is mapping $\phi : V(G) \to V(S) \times R$ such that:*

1. $\forall u \in L(G) : \phi(u) = [\mu(u), 0]$
2. $\forall u, v \in V(G)$ *such that* $v <_G u$*:* $\phi(v) <_S \phi(u)$ *and* $w(u, v) = d(\phi(u), \phi(v))$*, where* d *is the length of path between* u *and* v *in reconciled gene tree.*

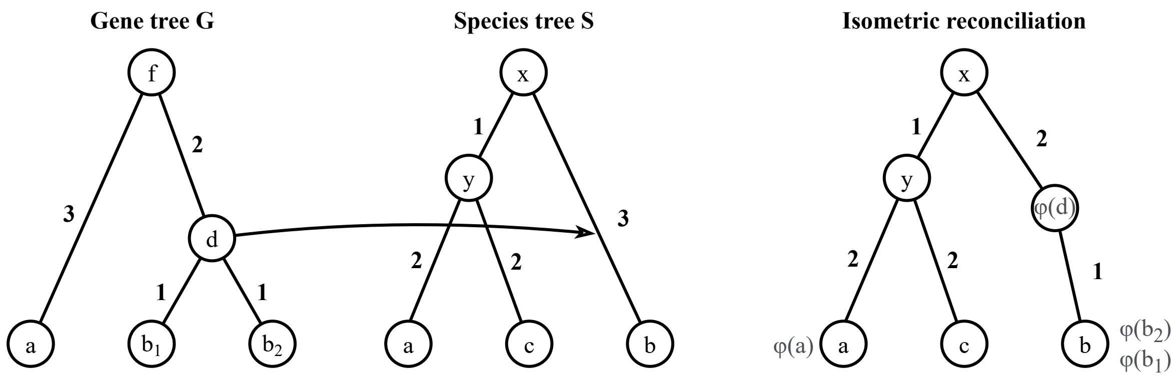

Figure 1.3: Isometric reconciliation. On the left is mapping of the node d of the gene tree G to the edge (x, b) of the species tree S. The result of the isometric reconciliation with mapped node d is on the right.

1.2.3.2 Inexact branch lengths

Input to the above-mentioned algorithms, gene trees and species trees with their branch lengths, are practically estimated from DNA sequences, which were gathered from present-day species [10]. The branch lengths are computed from observed mutations in collected DNA sequences. However, mutations happen randomly in the evolution and DNA sequences are also random samples from studied present-day species. It means that inferred gene trees and species trees with their branch lengths are estimated with an error.

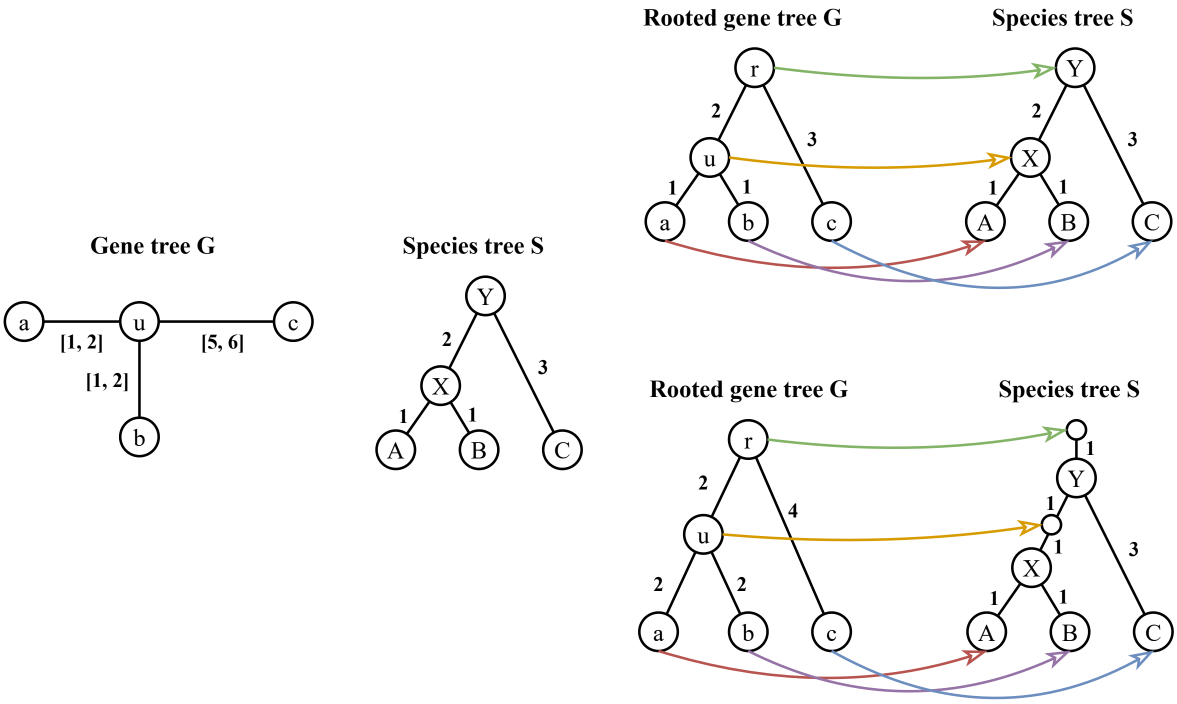

Figure 1.4: Isometric reconciliation with inexact branch lengths. On the left, we can see an unrooted gene tree G with inexact branch lengths and a rooted species tree S. On the right are two possible solutions with rooting the gene tree on the edge (u, c): the first reconciliation is without duplications and gene losses while the second solution maps gene tree to species tree with two duplications (u and r) and four gene losses.

To avoid defectiveness and make isometric reconciliation more precise, the isometric reconciliation with inexact branch lengths was introduces by Chládek in [6] and [5]. They define inexact branch length as an interval, where for every weighted edge stands: $w(u, v) = \langle w(u, v)_{min}, w(u, v)_{max}\rangle$. They present three types of algorithm.

Linear programming algorithm

The algorithm is based on linear programming and takes a rooted species tree and a rooted gene tree with inexact branch lengths as an input. They introduce a term of mapping depth, which represents a depth of mapped node $u \in V(G)$ in species tree: $D(\phi(u))$. The solution of isometric reconciliation is to find mapping depths to all nodes from gene tree to species tree. They suggest a set of 5 inequalities that can be solved by a linear program.

The first two inequalities assume that the difference of mapping depths of two nodes, where the one is a parent and the other is his child, has to be between the maximal and minimal length of the edge. The second two inequalities are similar and say that the distance between depths of two neighbouring nodes in species tree must be within the maximal and minimal length of the edge. Last inequality restricts mapping depth of node from gene tree to be same smaller than the depth of its LCA-mapping node in the species tree. The algorithm also defines that leaves from gene tree map to leaves from species tree and the root of the species tree is in depth 0, so every node, which maps above the root has negative depths.

For N nodes, the algorithm can get to the result in polynomial time. It also works if the gene tree and species tree are non-binary trees. With a few changes in inequalities, the linear program can find reconciliation to semi-rooted and unrooted species and gene trees.

Two-pass algorithm

The two-pass algorithm is faster than the linear programming algorithm. The input for the algorithm is a rooted species tree with exact branch lengths and a rooted gene tree with inexact branch lengths. It consists of two parts: upward and downward sweep.

The upward sweep goes from the leaves of the gene tree to the root and it computes preliminary interval $\langle x[u]_{min}, x[u]_{max}\rangle$ for each node u. The interval is a set of all potential mapping depths values of node u over all reconciliations of a subtree of the gene tree rooted at node u to the species tree. It does not take into account the rest of the gene tree, only the descendants of node u. The highest mapping point of node u, $x[u]_{min}$, is computed by taking the maximum value of both children highest mapping points subtracted by the longest branch length values. Similarly is calculated the lowest mapping point $x[u]_{max}$, where both children lowest mapping points are subtracted by the shortest branch length values and the minimum of these values is taken. If $x[u]_{min} > x[u]_{max}$, the interval is empty thus for the input is no reconciliation.

To get the final interval $\langle X[u]_{min}, X[u]_{max}\rangle$ for each node u, the downward sweep goes from the root of the gene tree to its leaves. For all nodes in gene tree (except root), the final interval is computed from their parent final interval, where the minimal mapping depth $X[u]_{min}$ is the maximum value of the highest mapping point of node u and minimal mapping depth of parent added by the shortest branch length value. The maximum mapping depth $X[u]_{max}$ is computed as the minimum value of the lowest mapping point of node u and maximal mapping depth of parent added by the longest branch length value.

Running time of this algorithm is $O(N)$. The same running time is for semi-rooted gene tree. In this situation, it needs to be used linear programming to obtain final intervals of the possible root and its children. The algorithm can be as well applied to an unrooted gene tree, where the running time is $O(N^2)$, because it needs to run on every edge as we do not know on which edge the root is located. This algorithm will be used in further work.

Parsimonious algorithm

The parsimonious algorithm looks for the most parsimonious solution over all isometric reconciliations by counting the number of duplications and gene losses. The aim is to find the smallest number of duplications and gene losses. In the thesis, [5] are considered 3 different types of parsimonious algorithms for 3 types of gene tree: rooted with exact branch lengths, rooted with inexact branch lengths and semi-rooted or unrooted.

The algorithm designed for reconciliation of a rooted gene tree and species tree with exact branch lengths count duplication and gene losses on subtrees. It has the best running time of $O(N \log N)$. If node u from gene tree is not mapped to its LCA-mapping in the species tree, the mapping of node u to the species tree is considered as duplication. The number of gene losses is computed from a path in species tree between mappings of node u and node v, where u is the parent of v. Each node from the species tree, that occurs on this path, is considered as speciation. It creates a copy of gene represented by the edge (u, v) from gene tree, but the gene continuous to only one child, so there is a loss on the other lineage.

Improved algorithm for a rooted gene tree with inexact branch lengths and a rooted species tree with exact branch lengths counts duplication and gene losses on subintervals. It splits the mapping depth interval into non-overlapping subintervals. The goal is to compute the number of duplication and gene losses for all possible subintervals, which is done by counting function from the previous algorithm. The time complexity of the algorithm is $O(N^3 \log N)$.

The most parsimonious algorithm assumes semi-rooted or unrooted gene tree with inexact branch lengths and a rooted species tree with exact branch lengths. In both cases of the gene tree, the exact location of the root is unknown. The algorithm firstly runs the previous algorithm on nodes of edge, where the possible root can be located. Then, it uses linear programming to find the location of the root based on computed numbers of duplications and gene losses in subintervals of the possible edge nodes. The running time of this algorithm is $O(N^4 \log N)$.

2 Algorithms

In this chapter, we will show different algorithms used to obtain the most parsimonious isometric gene tree reconciliation which will be implemented in our software.

The required input for our algorithms is a rooted species tree with exact branch lengths and an unrooted gene tree with inexact branch lengths. The gene tree is sequentially processed by following algorithms. Firstly, we find possible roots and root the unrooted tree gene tree, which results in multiple rooted gene trees. Afterwards, for each rooted gene tree we perform the two-pass algorithm to find a reconciliation (Chapter 1.2.3.2) and count the number of duplications and gene losses in the found reconciliation. Finally, we select the most parsimonious reconciliation among those considered.

2.1 Rooting the gene tree

An unrooted gene tree has an infinite number of possible root location. We present an algorithm to find a finite set of possible roots that are distant by given step on every edge e of an unrooted gene tree G. However, our set of possible roots may not always contain an optimal solution.

For each edge $e \in E(G)$, we transform the unrooted gene tree into a semi-rooted gene tree by rooting the subtrees at vertices $u \in V(G)$ and $v \in V(G)$ of the edge $e = (u, v)$. The edge is subsequently used as the parameter for the Algorithm 1 to select a set of roots on edge e, each root given by a pair of intervals. Let r be the root of a semi-rooted gene tree G then the first interval of the pair represents the length of edge (u, r) and the second interval represents the length of edge (r, v).

At the beginning of the Algorithm 1, we define essential variables. The set of possible pairs of intervals for subdividing the edge e are stored at the variable *intervals*

Algorithm 1 Possible intervals to subdivide given edge e

function GETINTERVALS($e \in E(G), step \in R$)
  difference = $w(e)_{min} - w(e)_{max}$
  if $step > difference \div 2$ **then**
    intervalSize = $difference \div 2$
  else
    intervalSize = step
  $w(u,r)_{min} = w(e)_{min}$
  $w(u,r)_{max} = w(e)_{max}$ - intervalSize
  $w(r,v)_{min} = \epsilon$
  $w(r,v)_{max} =$ intervalSize
  intervals.add($[\epsilon, \epsilon]$, $[w(e)_{min} - \epsilon, w(e)_{max} - \epsilon]$)
  intervals.add($[w(e)_{min} - \epsilon, w(e)_{max} - \epsilon]$, $[\epsilon, \epsilon]$)
  if $step \neq 0$ **then**
    while $w(r,v)_{max} < w(e)_{max}$ **and** $w(u,r)_{max} > 0$ **do**
      intervals.add($[w(u,r)_{min}, w(u,r)_{max}]$, $[w(r,v)_{min}, w(r,v)_{max}]$)
      $w(u,r)_{min}$ -= step
      if $w(u,r)_{min} <= 0$ **then**
        $w(u,r)_{min} = \epsilon$
      $w(u,r)_{max}$ -=step
      $w(r,v)_{min}$ += step
      $w(r,v)_{max}$ += step
  return intervals

that is also the return value of the function. We added condition for special cases, where the difference between the maximal and minimal original length of edge e divided by 2 is less than the size of the step. With the special case condition, we can infer intervals with a better range or intervals that would be normally skipped (Fig. 2.1).

To cover most of the possibilities, we allow rooting the gene tree right above the vertices u and v of the edge e with ϵ distance from the vertices. The ϵ is by default set to 1×10^{-6} and signifies the edge length close to the 0. We do not allow 0 edge length or interval starting with 0 as $[0, \epsilon]$ to avoid mapping the root into vertex u or v.

We get two possible roots after subdividing the edge e right above the vertices u and v. The first possible root subdivides edge e into two edges with interval lengths $w(u,r) = [\epsilon, \epsilon]$ on the left from the root and $w(r,v) = [w(e)_{min} - \epsilon, w(e)_{max} - \epsilon]$ on the right from the root, where $w(e)_{min}$ is original minimal length of the edge e and $w(e)_{max}$ is original maximal length of the edge e. The second option of the root subdivide the edge e with intervals of the same length, that are flipped, so the original interval $w(u,r) = [w(e)_{min} - \epsilon, w(e)_{max} - \epsilon]$ is on the left from the root and $w(r,v) = [\epsilon, \epsilon]$ in on the right from the root.

After creating the first two options for the possible root, we check the size of the step, which is set to 0.01 by default. If the size of the step is 0, we have no distance between possible roots, thus no distance to shift while subdividing the edge e. However, if the step is different from 0, we run a while loop to get possible roots inside the edge e. In each iteration, we subtract the step from the minimal and maximal length of the left interval of the subdivided edge e and add a step to the minimal and maximal length of the right interval of the subdivided edge e. The while loop goes until the maximal length of the right interval is the same or bigger as the original maximal length of the edge e or the maximal length of the left interval reaches 0 or less.

Rooting the gene tree goes over all edges in the unrooted gene tree G. The running time of the function *getIntervals* in Algorithm 1 is $O(p)$, where p stands for number of iterations in while loop that can be expressed as $p = ceil(totalMaxLength/step) - 1$. The result of function *getIntervals* is a set of intervals for subdividing the edge e. Subsequently, the intervals are used for a loop to root the semi-rooted gene tree G resulting in a set of rooted gene trees with inexact branch lengths, which running time is $O(m)$ for $m = p + 2$. Therefore, the total running time of rooting the unrooted gene tree G is $O(Nm)$ with N being the number of all edges in the unrooted tree G.

2.2 Counting algorithm

We present an algorithm for counting the number of duplications and gene losses in a rooted gene tree G with inexact branch lengths depending on a rooted species tree S with exact branch lengths. We allow only evolutionary events as duplication, gene loss and speciation can happen in evolutionary history. The counting algorithm consists of two parts: preprocessing and the main algorithm. In the preprocessing, we compute essential variables for the gene tree G that are subsequently used in the main algorithm.

The prerequisites for the counting algorithm are calculated depth $D(a)$ and level $l(a)$ for each $a \in V(S)$. For the gene tree G, we assume an interval of possible mapping depths $X[u] = [X[u]_{min}, X[u]_{max}]$ for each $u \in V(G)$ that can be computed with the two-pass algorithm (Chapter 1.2.3.2) and LCA-mapping $\sigma(u)$ for each $u \in V(G)$.

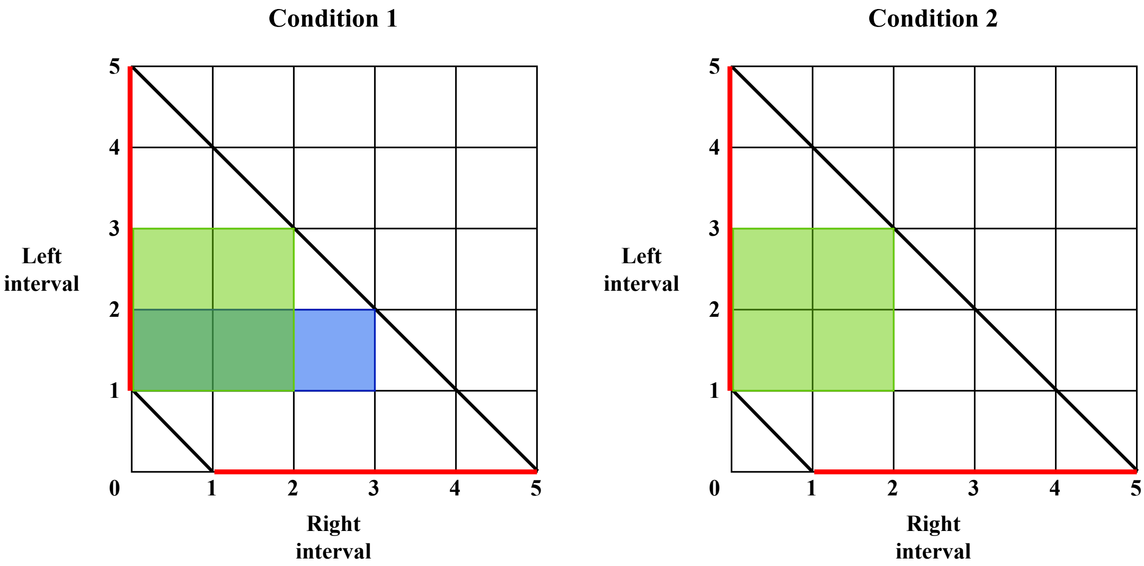

Figure 2.1: The X-axis on the graph stands for the possible length of interval on the right from the root and the y-axis signify the possible length of interval on the left from the root. Space between black lines represents all possible intervals after subdividing the edge e with the root. Red lines are for the root right above the vertices u and v, where one interval takes the original size and the second interval is $[\epsilon, \epsilon]$. Blue rectangle means intervals for the root inside the edge e without the special case condition. Green squares are inferred intervals for the root with special case condition.
Condition 1: $step > difference \div 2$ (the left figure)
Parameters: $w(e) = [1, 5]$ and $step = 3$.
In this case, the size of the step is bigger than the difference between the original minimal and maximal length divided by 2. Without the special case condition, we would only get the blue rectangle intervals $w(u, r) = [1, 2]$ on the left from the root and $w(r, v) = [\epsilon, 3]$ on the right from the root. The special case condition changes the blue rectangle into the green square which gives us intervals $w(u, r) = [1, 3]$ on the left from the root and $w(r, v) = [\epsilon, 2]$ on the right from the root with better coverage of the space.
Condition 2: $step > difference$ (the right figure)
Parameters: $w(e) = [1, 5]$ and $step = 5$.
The size of the step is bigger than the difference between the original minimal and maximal length. Without the special case condition, we would not get any possible intervals for subdividing the edge e. However, the special case condition infer green intervals $w(u, r) = [1, 3]$ on the left from the root and $w(r, v) = [\epsilon, 2]$ on the right from the root.

2.2.1 Preprocessing

In the preprocessing, we calculate necessary variables for nodes in the gene tree G that are used in the main algorithm 2.2.2. For each $u \in V(G)$ we compute $l(u)$, $speciesNodeBelow(u)$, $levelDistanceFromParent(u)$ and $mappedToLca(u)$. The level $l(u)$ variable of gene node u signifies number of species nodes on the path from gene node u to the root of the gene tree G. It is calculated from the $speciesNodeBelow(u)$ variable that represents species node $s \in V(S)$, which is right below gene node u, so the gene node u lies on the path from species node s to $parent(s)$. The $levelDistanceFromParent(u)$ means number of species nodes between gene node u and its $parent(u)$. It is not calculated for the root of the gene tree G as it has no parent. The variable $mappedToLca(u)$ presents the truth value of statement: $X[u]_{max} = D(\sigma(u))$, thus if the node u is mapped to $\sigma(u)$ or not.

We use three algorithms for calculating the variables: Algorithm 2 for calculating the $speciesNodeBelow(u)$, Algorithm 3 that sets level distance from parent $u \in V(G)$ to its children $\forall v \in children(u) : levelDistanceFromParent(v)$ and Algorithm 4, which uses both previous algorithms and computes the level of gene node $l(u)$ and $mappedToLca(u)$.

Algorithm 2 Computes species node below given gene node u

function COMPUTESPECIESNODEBELOW($u \in V(G), s \in V(S)$)
  speciesNodeBelow = s
  while $D(s) > X[u]_{max}$ **do**
    speciesNodeBelow = s
    if $parent(s) = null$ **then**
      break
    else
      s = parent(s)
  return speciesNodeBelow

The *computeSpeciesNodeBelow* function in Algorithm 2 saves the given species node s to the variable *speciesNodeBelow*. It always remembers the last species node below the given gene node u and serves as a return value. The while loop iterates over species nodes on the path from given species node s to the root of the species tree. If $D(s) > X[u]_{max}$ stands, it means the species node s is below the gene node u. Every iteration, we save the species node to the return value *speciesNodeBelow* and move on the path closer to the root by assigning $parent(s)$ to the s variable. If the species node s does not have a parent, the gene node u is above the root of the species tree. The while loop ends, if the while condition does not stand, so the new species node s is above gene node u or if the gene node u is above the root.

In the *levelDistanceFromChildren* function show in Algorithm 3, we compute the $levelDistanceFromParent(v)$ for each child v of the given node u. The level distance

Algorithm 3 Sets level distance from parent to children of node u

```
function LEVELDISTANCEFROMCHILDREN(u ∈ V(G))
    for v ∈ children(u) do
        levelDistanceFromParent(v) = l(v) - l(u) - mappedToLca(u)
```

is calculated as the difference between the level of child v and the level of its parent, node u. The distance is subtracted by 1 if the node u is mapped to the same depth as its $\sigma(u)$.

Algorithm 4 Compute levels for nodes from gene tree G

```
function COMPUTELEVEL(u ∈ V(G))
    if u ∈ L(G) then
        mappedToLca(u) = true
        speciesNodeBelow(u) = σ(u)
        l(u) = l(σ(u))
    else
        for v ∈ children(u) do
            computeLevel(v)
        depthDifference = D(σ(u)) − X[u]_max
        if X[u]_max < D(σ(u)) or depthDifference > ϵ then
            mappedToLca(u) = false
            node = max(v ∈ children(u): speciesNodeBelow(v))
            speciesNodeBelow(u) = computeSpeciesNodeBelow(u, node))
        else
            if ∀v ∈ children(u) : speciesNodeBelow(v) = σ(u) then
                if ∀v ∈ children(u) : mappedToLca(v) ∧ v ∉ L(G) then
                    v = v ∈ children(u) : min(v)
                    levelDistanceFromChildren(v)
                mappedToLca(u) = false
            else if ∃v ∈ children(u) : speciesNodeBelow(v) = σ(u) then
                mappedToLca(u) = false
            else
                mappedToLca(u) = true
            speciesNodeBelow(u) = σ(u)
        l(u) = l(speciesNodeBelow(u))
        levelDistanceFromChildren(u)
```

The *computeLevel* function in Algorithm 4 goes over all nodes in gene tree G in the direction from the leaves to the root. The leaves of gene tree $t \in L(G)$ are always mapped to its $\sigma(t)$ and their variables are set according to it. For each inner node $u \in V(G) \setminus L(G)$, we recognize whether the node u has the same maximal mapping depth $X[u]_{max}$ as $\sigma(u)$ or the difference between $X[u]_{max}$ and $\sigma(u)$ is smaller than ϵ, which allows us to determine if the node u is mapped to its $\sigma(u)$ even when the depths are not same because of the rounding error. By default, the ϵ is set to 1×10^{-6}.

In case that node u has not the same depth as $\sigma(u)$ and also their *depthDifference* is bigger than ϵ, we set $mappedToLca(u)$ to false. From the children of node u, we save the closest species node to the gene node u into variable *node* and run function *computeSpeciesNodeBelow* in Algorithm 2 to get $speciesNodeBelow(u)$.

Otherwise, when the node u has the same depth as $\sigma(u)$ or the *depthDifference* is smaller or equal to the ϵ, we consider three cases can happen, where the first two cases check if both or at least one child is already mapped to $\sigma(u)$, because exactly one node from the gene tree can be mapped to exactly one node from species tree to be considerate as speciation. In the first case, we consider that both $v \in children(u)$ have the same $speciesNodeBelow(v)$ as $\sigma(u)$. It means that both children of node u can be mapped to the same species node, so we check if their $mappedToLca(v)$ is set to true. We skip the case, where $children(u)$ are leaves as they are allowed to map to the same species node. If the condition holds such that $children(u)$ are mapped to the same species node and they are not leaves, we take the child v that is closer to the node u and set its $mappedToLca(v)$ to false that also affects the distance from $children(v)$, thus we call the $levelDistanceFromChildren(v)$ to reset the distance. As the other child is already mapped to the species node, we set $mappedToLca(u)$ to false.

The second case checks if at least one $v \in children(u)$ is mapped to $\sigma(u)$. If the condition holds, we can not map another gene node to the species node, so we set $mappedToLca(u)$ to false.

In the third case, any $v \in children(u)$ has the same $speciesNodeBelow(v)$ as the $\sigma(u)$, thus we set the $mappedToLca$ to true.

Lastly, we set the $speciesNodeBelow(u)$ to the $\sigma(u)$ and level of node u to the level of computed $speciesNodeBelow(u)$. Then, we run function *levelDistanceFromChildre* in Algorithm 3.

The *computeLevel* function goes over all nodes in the rooted gene tree G. If the gene tree G is balanced, the function *computeSpeciesNodeBelow* has running time $O(logN)$, where N is number of nodes in gene tree G. The *levelDistanceFromChildren* function is computed in constant time. So the total preprocessing running time is $O(NlogN)$ for balanced gene tree G. However, if the gene tree G is not balanced, the preprocessing running time is $O(N^2)$.

2.2.2 Main algorithm

The prerequisites for the *countDL* function shown in Algorithm 5 are computed in preprocessing. Besides, we need to have set the *countLossesAboveRoot* variable. If it is true, we count losses above root that occurred as a result of the gene tree G not containing genes of all species from the species tree S.

Algorithm 5 Counts duplications and gene losses in gene tree G

function COUNTDL($u \in V(G)$)
  if $u \neq L(G)$ **then**
    v, w = children(u)
    $DL_u = DL_v + DL_w$
  loss = levelDistanceFromParent(u)
  if $parent(u) = null$ **and** $parent(\sigma(u)) \neq null$ **and** *countLossesAboveRoot* **then**
    loss += l(u);
  if not $mappedToLca(u)$ **then**
    duplication = 1
  return DL_u +(duplication, loss)

We count the evolutionary events in the direction from leaves to the root of gene tree G. For each node $u \in V(G)$ and its parent $v \in V(G)$, we consider evolutionary events that occur in the node u and on the edge (u, v). We do not compute evolutionary event in the parent, they are determined directly in the parent. In the root, we only calculate the evolutionary events that happened in the node as the root has no parent thus no edge to consider.

The number of duplications and gene losses are depicted as pair $DL = (duplication, loss)$. The sum of the pairs DL_1 and DL_2 is computed as $DL_1 + DL_2 = (duplication_1 + duplication_2, loss_1 + loss_2)$. For each node $u \in V(G)$, we calculate the DL_u, which corresponds to the number of duplications and gene losses inferred in the subtree of node u. Thereafter, we infer the duplication and gene losses in the node u and on the edge above the node u.

The number of gene losses on the edge (u, v) is calculated as the number of species nodes on the path from node u to node v, which is precomputed in variable $levelDistanceFromParent(u)$. If $\sigma(root(G)) \neq root(S)$, thus the $\sigma(root(G))$ is below $root(S)$ means that some species do not have their gene in the gene tree. Therefore, the gene loss occurred before the $root(G)$, which resolves in extra gene loss that can be added if the variable *countLossesAboveRoot* is true. The truth value of *countLossesAboveRoot* is set by the user.

Duplication and speciation are easy to determine since it depends on whether the node u is mapped to $\sigma(u)$ or above, which we already precomputed in $mappedToLca(u)$.

The function return pair DL corresponds to the number of duplication and gene losses in the subtree of node u, in the node u and on the edge above node u.

Gene losses and duplications are computed in constant time for one gene node. The *countDL* function goes over all nodes in gene tree G thus the running time is $O(N)$, where N is the number of all node in the gene tree G.

3 Implementation

The implementation of our software is built on Chládek's source code from his diploma thesis [5]. We apply his implementation of basic classes for defining the phylogenetic tree, parsing an unrooted gene tree and a rooted species tree from a file with several changes. We also use his implementation of the two-pass algorithm, which we mentioned before in Chapter 1.2.3.2, to compute the gene tree possible mapping depths.

On this basis, we implement our algorithms described in Chapter 2 to find the most parsimonious reconciliation.

Our software is implemented in the programming language Java and has a command-line interface.

3.1 Classes and variables

The implemented software consists of 15 classes that we briefly introduce:

- Class Node - represents a node in a phylogenetic tree
- Class Edge - represents an edge in a phylogenetic tree
- Class Interval - represents a length of edge as interval
- Class DL - represents a score of reconciliation
- Class UnrootedNode - represents a node in an unrooted gene tree
- Class UnrootedTree - represents an unrooted gene tree
- Class RootedExactNode - represents a node in a rooted species tree with exact branch lengths

- Class RootedExactTree - represents a rooted species tree with exact branch lengths
- Class RootedIntervalNode - represents a node in a rooted gene tree with inexact branch lengths
- Class RootedIntervalTree - represents a rooted gene tree with inexact branch lengths
- Class Parser - parses phylogenetic trees and their leaf-mapping from files
- Class Loader - load arguments from a command line and calls Parser on files with stored phylogenetic trees
- Class Printer - prints reconciliation solutions into files
- Class Reconciliator - computes reconciliation with its score
- Class Main - the main class of the software that calls other classes to load files, compute and print reconciliation

3.2 Differences from the original source code

As our source code is built on Chládek's source code, we show the differences in relations and entities between source codes in Fig. 3.2. The differences, new variables and classes are described below by classes in more details.

Class Interval

We change the name of parameters to *minLength* and *maxLength* and delete the unnecessary variable *OriginalMappingDepth* as we use the *Interval* to store the minimal and maximal length of new possible edges that can be created after subdividing the original root edge in function*getIntervals* (Algorithm 1) in *Reconciliator* class. The parameters are numbers of type double.

Class RootedExactNode

In the *RootedExactNode* class, we define a new *level* variable. It is a number of type integer. The variable is set while parsing the species tree in the *Parser* class and used for computing the level of gene nodes as shown in function *computeLevel* (Algorithm

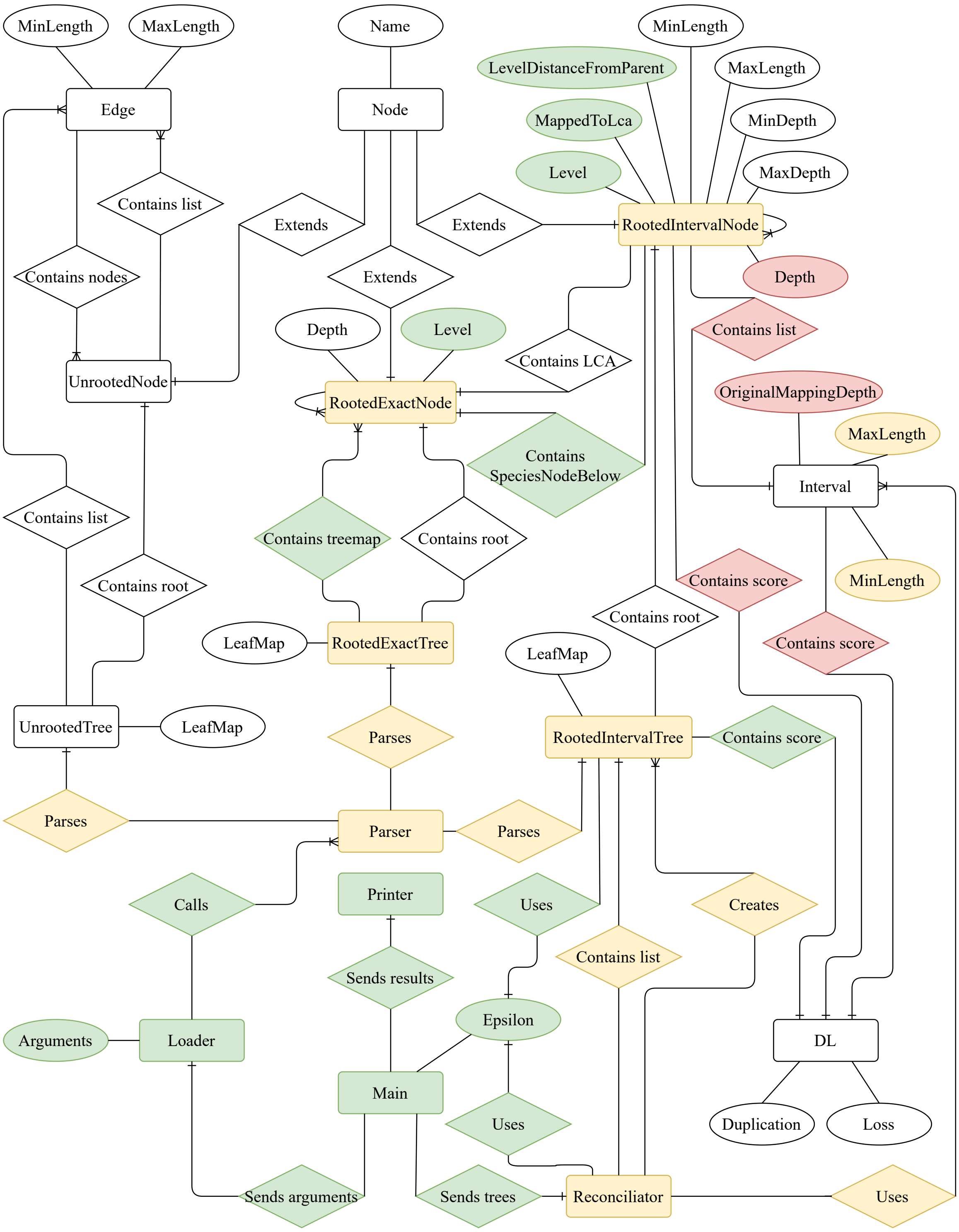

Figure 3.1: Red entities and relations represent deleted variables or functions. Yellow entities and relations show changes to the original source code. Green entities and relations depict added classes, variables or functions.

4) in *Reconciliator* class.

Class RootedExactTree

We add structure *TreeMap* containing leaves, where the key is the name of the leaf and the value is an object representation of that leaf as *RootedExactNode*. It is used to set the LCA-mapping variable of leaves in a gene tree according to the leaf-mapping.

Class RootedIntervalNode

The unnecessary *depth* variable was deleted as we store the mapping depth of a *RootedIntervalNode* in variables *minDepth* and *maxDepth*. We also delete the *DL*, because we do not store the reconciliation score in nodes and the list of *Interval* since we have the mapping depth interval and the interval of the edge above stored in separated variables. Besides, we add new variables: *level*, *mappedToLca*, *levelDistanceFromParent* and *speciesNodeBelow* that are computed and used in algorithms in Chapter 2.2. The *level* and *levelDistanceFromParent* are numbers of type integer. The *mappedToLca* is data type boolean *speciesNodeBelow* is object of type *RootedExactNode*.

Class RootedIntervalTree

In the *RootedIntervalTree*, we implement our algorithms from Chapter 2.2 and their are called from the *Reconciliator* class. We add the variable of type *DL* to store the number of duplication and gene losses inferred in the gene tree.

Class Parser

We change the parsing method for a species tree and an unrooted gene tree. In the species tree parsing method, we infer *level* in species tree nodes. In the unrooted gene tree parsing method, we resolve inexact branch lengths if the given gene tree has inexact branches or we transform exact branch lengths to inexact branch lengths if the given gene tree has exact branch lengths. To modify the exact branch lengths into inexact branch lengths, the user needs to set the tolerance value that has to be from interval $[0, 1]$. The interval of the edge is compute as: $[length - (tolerance \cdot length), length + (tolerance \cdot length)]$.

Furthermore, we add a parsing method for a rooted gene tree that can either resolve inexact branch lengths or transform exact branch lengths to inexact branch lengths. We included a method to parse mapping of gene leaves to species leaves from a file and a method for transforming a rooted gene tree to an unrooted gene tree. This allows us to forget the original root of a gene tree and find a new one with function *getIntervals* (Algorithm 1). All methods in the class are called from the *Loader* class.

Class Loader

The *Loader* class is initialized with arguments from the command line. It loads the arguments and calls functions from *Parser* according to the requirements specified in the arguments. We recognize 11 arguments that are listed in Table 3.1.

Table 3.1: Input arguments

Argument	**Description**
-help	shows help information with all possible input arguments
-S <species tree>	path to the rooted species tree file in Newick tree format (mandatory input)
-G <gene tree>	path to the rooted or unrooted gene tree in Newick tree format (mandatory input)
-M <species map>	mapping of genes to species
-t <tolerance>	required tolerance from interval $[0, 1]$ (By default, it is set to 0.5)
-s <step>	required step from $[0, \infty]$ (By default, it is set to 0.01)
-r	signifies that the given gene tree is rooted (By default, the given gene tree is considered to be unrooted)
-reroot	signifies that the given rooted gene tree is wished to be rerooted
-l	signifies counting the gene losses above the root of given gene tree
-p <print type>	required print type from two options *"sol"* and *"rel"* (By default, the print type is set to be *"sol"*
-epsilon <epsilon>	required epsilon as number of type double (By default, it is set to 1×10^{-6}

The main method is *loadArgs* which is called from the *Main* class and returns an array of *Object*: a rooted species tree, a rooted gene tree or an unrooted gene tree, *step* and *countLossesAboveRoot* needed for algorithms from Chapter 2.2 implemented in *Reconciliator* class along with *dirPathGene* and *printType* required in *Printer* class functions.

Class Reconciliator

The algorithms from Chapter 2 and the two-pass algorithm by Chládek from Chapter 1.2.3.2 are called in the *Reconciliator* class. These are the main methods to obtain the most parsimonious reconciliation.

The *Reconciliator* class is initialized with a rooted species tree, a rooted gene tree or an unrooted gene tree, *step* and *countLossesAboveRoot* that are inferred in *Loader* class. It uses *Interval* to store the minimal and maximal length of new edges in function *getIntervals* (Algorithm 1), which are used for subdividing the root edge and creating the *RootedIntervalTree*. The class returns a list of *RootedIntervalTree* with most parsimonious reconciliation that has its reconciliation score store in *DL*.

Class Printer

The class is initialized with list of *RootedIntervalTree*, *dirPathGene* and *printType*. We recognise two types of printing the results of the reconciliation into the file: *"rel"* and *"sol"*.

With the *"rel"* type, the function prints relations between genes in the gene tree for each inferred *RootedIntervalTree*. If solutions contain more trees with the same topology and reconciliation score, it prints the number of the same tree after each gene relation printout. For each $u \in L(G)$, it prints *"gene"* and name of the node u. For each $v \in V(G) \setminus L(G)$, it allows evolutionary events as speciation, duplication and gene loss depending on what occurred in the node v or on the branch from the node v to its parent $parent(v)$. The speciation and duplication are inferred in the node v. They print as *"spec"* for speciation and *"dup"* for duplication with names of $children(v)$ separated by a tab. The gene loss happens on the branches. It prints as *"loss"* with the name of node v. The name of each node consists of genes found in the subtree of that node separated by commas.

The *"sol"* type prints gene trees in special Newick format, where branch lengths are inexact. All computed solutions are printed into one *.txt* file, where solutions are separated with an empty line.

The file with results is saved in the same directory as the given gene tree.

Class Main

The *Main* class receives arguments given by user and send it to the *Loader* class for processing. Sequentially, sends a rooted species tree, a rooted gene tree or an unrooted gene tree, *step* and *countLossesAboveRoot* to the *Reconciliator* class to infer the most parsimonious reconciliation, which returns list of *RootedIntervalTree*. Eventually, it sends the list of *RootedIntervalTree*, *printType* and *dirPathGene* to the *Printer* to print the results. Also, it stores the value of ϵ that are required in functions *getIntervals* (Algorithm 1) in *Reconciliator* class and *computeLevel* (Algorithm 4) in *RootedIntervalTree* class.

4 Experiments and results

We evaluate our software for isometric reconciliation on a simulated and real dataset that were used to evaluate SPIMAP [17] and TreeFix [20] in previous studies. The simulated dataset consists of 1000 simulated gene families of two clades of species: 12 *Drosophila* genomes and 16 fungal genomes, generated with the SPIMAP model. The real dataset includes 5351 real gene families from 16 fungal genomes.

To compare the results, we run other software for computing the gene tree reconciliation: Notung [8], TreeFix [20] and Treerecs [7], on the same dataset.

4.1 Simulated dataset

In the simulated dataset, we know the correct gene tree with its evolutionary events, which allows us to test several aspects. We evaluate our software for isometric reconciliation for two cases: without rerooting the gene tree and with rerooting the gene tree. In the experiments, we measure the precision of inferred duplications and gene losses, the average runtime for computing the results for each gene tree and the number of gene trees without solution, as our isometric reconciliation software takes branch lengths into account and occasionally the gene tree can not fit the species tree correctly, because some small errors in branch lengths. In the second case with rerooting the gene tree, we also measure the precision of the correctly inferred root.

4.1.1 Without rerooting the gene tree

We take the rooted gene tree as it is from the dataset and run a reconciliation with *countDL* function (Algorithm 5 in Chapter 2.2.2). We evaluate it several times with different tolerance setting to scale up the edges. The tolerance setting are $0.0, 1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-3}, 0.1, 0.3, 0.5, 1.0$ and the $\epsilon = 1 \times 10^{-6}$. We compare our results with only Notung and Treerecs, as the TreeFix software automatically reroot the given gene tree.

Flies dataset

On the flies dataset (Tab. 4.1), our software has a problem to infer reconciliation for all gene trees with tolerance set to 0.0, thus 3.97% of the gene trees are without reconciliation. Still, the reconciled trees have perfect precision and sensitivity of inferred duplication and gene losses. With the tolerance set to 1×10^{-6}, we computed isometric reconciliation for all gene trees, but some nodes are incorrectly detected as duplications or gene losses, so the inferred duplications have precision 99.93% with the sensitivity of 100% and gene losses has precision 99.76% with the sensitivity of 99.98%. It is caused by a rounding error on the last decimal place since the edge length of the gene trees in the dataset has 6 decimal places. Higher tolerance settings show the same results, where our isometric reconciliation software infer reconciliation for all gene trees from the dataset with 100% precision and sensitivity of duplication and gene losses. The Notung and Treerecs computed reconciliation for all gene trees with perfect precision and sensitivity of inferred duplication and gene losses. However, the average running time for each gene tree is best in our reconciliation software, which is about 0.003617 for each tree in each tolerance setting. In comparison, the Treerecs average time is bigger by 90.23% from our average running time. The Notung has the longest average time, which is bigger by 94.49% from the Treerecs average running time and by 99.46% from our average running time.

Fungi dataset

The results for the fungi dataset (Tab. 4.2) are similar to the results for the flies dataset. Our isometric reconciliation software has also a problem inferring reconciliation for all gene trees with a tolerance setting of 0.0, ergo the 9.03% of gene trees has no reconciliation solution. For the remaining 90.97% of gene trees, our software finds the correct reconciliation with 100% precision and sensitivity of inferred duplications and gene losses. Our software recognizes isometric reconciliation for all gene trees from the dataset with tolerance set to 1×10^{-6}, but the precision of duplications and gene losses decrease to 99.99% and 99.98%, respectively, because of the rounding error on the last decimal place since the edge lengths of gene trees from the fungi dataset has also 6 decimal places. However, the sensitivity of inferred duplications and gene losses stays

Table 4.1: Flies: results of phylogenetic software on simulated dataset without rerooting the gene tree

Software[a]	**W/o sol**[b]	**Duplication**		**Gene loss**		**Runtime**[g]
		Prec[c]	**Sens**[d]	**Prec**[e]	**Sens**[f]	
Our (t - 0.0)	3.97	100	100	100	100	0.007415
Our (t - 1×10^{-6})	0	99.93	100	99.76	99.98	0.008160
Our (t - 1×10^{-5})	0	100	100	100	100	0.003050
Our (t - 1×10^{-3})	0	100	100	100	100	0.003026
Our (t - 0.1)	0	100	100	100	100	0.001785
Our (t - 0.3)	0	100	100	100	100	0.001771
Our (t - 0.5)	0	100	100	100	100	0.001919
Our (t - 1.0)	0	100	100	100	100	0.001810
Notung	0	100	100	100	100	0.671599
Treerecs	0	100	100	100	100	0.037014

[a] Phylogenetic software with *"t"* as tolerance setting.
[b] Percentage of gene trees without a solution.
[c] Precision of inferred duplications.
[d] Sensitivity of inferred duplications.
[e] Precision of inferred gene losses.
[f] Sensitivity of inferred gene losses.
[g] Average runtime of computing the reconciliation for each gene tree in seconds.

100%. With the remaining tolerance setting, we infer reconciliation for all gene trees from the dataset with perfect precision and sensitivity of duplications and gene losses. The other software, Notung and Treerecs, compute reconciliation for all gene trees with 100% precision and sensitivity of inferred duplications and gene losses. To compare the average running time for each gene tree, our software has the best average running time about 0.004006 for each tolerance setting. The Treerecs has the second-best average running time bigger by 93.11% from our average running time. The longest average running time has Notung, which is bigger by 91.8% from the Treerecs average running time and by 99.43% from our average running time.

Conclusion

To sum up, our software runs the best with tolerance 1×10^{-5}, which is by one decimal place shorter than the edge length of gene trees from both datasets. Thus, if the tolerance has less than or an equal number of decimal places, our software may not find reconciliation for all rooted gene trees or may find a reconciliation with wrongly detected duplications and gene losses, which decreases the precision and sensitivity of inferred duplications and gene losses. In terms of time, our software infers the reconciliation for each gene tree the fastest compared to the Notung and Treerecs.

Table 4.2: Fungi: results of phylogenetic software on simulated dataset without rerooting the gene tree

Software[a]	W/o sol[b]	Duplication		Gene loss		Runtime[g]
		Prec[c]	Sens[d]	Prec[e]	Sens[f]	
Our (t - 0.0)	9.03	100	100	100	100	0.007225
Our (t - 1×10^{-6})	0	99.99	100	99.98	100	0.005962
Our (t - 1×10^{-5})	0	100	100	100	100	0.003898
Our (t - 1×10^{-3})	0	100	100	100	100	0.004728
Our (t - 0.1)	0	100	100	100	100	0.003158
Our (t - 0.3)	0	100	100	100	100	0.002231
Our (t - 0.5)	0	100	100	100	100	0.002366
Our (t - 1.0)	0	100	100	100	100	0.002481
Notung	0	100	100	100	100	0.708665
Treerecs	0	100	100	100	100	0.058145

[a] Phylogenetic software with *"t"* as tolerance setting.
[b] Percentage of gene trees without a solution.
[c] Precision of inferred duplications.
[d] Sensitivity of inferred duplications.
[e] Precision of inferred gene losses.
[f] Sensitivity of inferred gene losses.
[g] Average runtime of computing the reconciliation for each gene tree in seconds.

4.1.2 With rerooting the gene tree

The software for isometric reconciliation takes the rooted gene tree as input with an argument to reroot the given gene tree. We forget the root of the rooted gene tree and transform it into the unrooted gene tree. On the obtained unrooted gene tree, we run the *getIntervals* function (Algorithm 1 in Chapter 2.1) to get a set of roots given by a pair of intervals for subdividing each edge of the unrooted gene tree, where the first interval from the pair signifies the edge length on the left from the new root and the second interval from the pair signifies the edge length on the right from the new root. Then we run a reconciliation with *countDL* function (Algorithm 5 in Chapter 2.2.2). We want to find a new root minimizing the number of inferred duplications and gene losses in reconciliation. The process is executed several times with different tolerance and step settings. The tolerance and ϵ setting are the same as in the first case without rerooting the gene tree (Chapter 4.1.1). The step settings are $0.0, 0.1, 0.3, 0.5, 1.0, 2.0$ since the edge lengths of species trees and gene trees from the dataset are in a range from about 1 to 78 expressed in million years.

Flies dataset

At first, we run our software with all tolerance settings and step set to 0.0, as we can see in Table 4.3, where for each edge $(u, v) \in E(G)$, the *getIntervals* function returns a set of roots subdividing the edge right above the vertices u and v. Because of that, our software is not able to infer reconciliation for all gene trees at all tolerance settings. It not finds any reconciliation with tolerance set to $0.0, 1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-3}$. With the tolerance of 0.1, it recognizes the first isometric reconciliations for 82.37% of gene trees from the dataset. The percentage of gene trees without computed reconciliation decreases with the increasing tolerance. With the tolerance of 1.0, only 0.48% of gene trees does not have inferred reconciliation. As our software does not find solutions for the reconciliation with the first 4 tolerance settings, we can not calculate and compare the precision of inferred root and precision and sensitivity of inferred duplications or gene losses. For the remaining tolerance settings, the precision of correctly inferred root is highest with tolerance set to 1.0. It rises with increasing tolerance. A slight decrease is between tolerances 0.3 and 0.5 caused by newly inferred reconciliations for gene trees that have no solution with tolerance 0.3 but find a solution with tolerance 0.5. The precision of inferred duplications has a growing tendency for all remaining tolerance settings. The sensitivity of inferred duplications is sensitive to the rooting of a gene tree on a different edge than the original edge, so when the precision of correctly inferred root drops between tolerances 0.3 and 0.5, the sensitivity drops by 0.09% too. Finding the root on different edge induces duplications and gene losses on different nodes. Apart from that, it has an increasing tendency. The precision of inferred gene losses is increasing for the remaining tolerance settings with a slight drop of 0.01% between tolerances 0.3 and 0.5 also caused by the wrongly inferred roots. The sensitivity of inferred gene losses is highest with the tolerance of 0.1 and then it drops by 18.56%. It is induced by a decrease in the percentage of gene trees without solutions, where we infer reconciliation for more gene trees with tolerance 0.3 than with the tolerance 0.1, but some of them has wrongly inferred gene losses, where the gene loss is not supposed to be. Besides this drop, it has a growing tendency from tolerance 0.3 to tolerance 1.0.

With step set to 0.1, we find reconciliation solutions already at tolerance set to 0.0 for 1.65% gene trees from the dataset. The percentage of gene trees without solution decreases as the tolerance increase and it is 0% from tolerance 0.1 and more. The precision of correctly inferred root is 100% for tolerance settings from 0.0 till 0.1 include. It does not even drop with the big increase of inferred reconciliations between tolerances 1×10^{-5} and 1×10^{-3} for gene trees that have not solution at tolerance 1×10^{-5}, but the solution exists at tolerance 1×10^{-3}. It starts decreasing with tolerance 0.3 and bigger induced by increasing tolerance, allowing finding a solution on another edge than the original edge with a better reconciliation score. The precision of inferred duplications has similar development as the precision of correctly inferred root with one exception at the tolerance set to 1×10^{-3}. The high increase of gene trees with solution caused a big drop, where a lot of these solution contains wrongly inferred duplications which correct the increase in tolerance to 0.1. The decrease from tolerance set to 0.3 and more is caused by a change in finding a different root, which induces duplications on different nodes. The sensitivity of inferred duplications is perfect for tolerances from 0.0 to 0.1 include. It starts decreasing between tolerances 0.1 and 0.3 induced for the same reason as the decrease in precision of inferred duplications. The precision of inferred gene losses is perfect except tolerance set to 1×10^{-3}, where the cause of the drop is the same as for the precision of inferred duplications. The sensitivity of inferred gene losses is perfect until it starts decreasing between tolerances 0.1 and 0.3 caused by finding a reconciliation solution with different root and fewer gene losses, where the inferred gene losses are correct, as shown by the precision of inferred gene losses, but we do not find all expected gene losses.

The next tested step setting is 0.3, where the results are almost the same as the result with the step setting 0.1 with the exception for tolerances set to 1×10^{-3} and 0.1. At the tolerance 1×10^{-3}, we are not able to infer solutions for as many gene trees as with step set to 0.1. Still, the 4.27% increase in inferred reconciliation solutions for gene trees between tolerances 1×10^{-5} and 1×10^{-3} induces a decrease in precision of inferred duplications and gene losses. The second exception is at the tolerance 0.1, where the precision of inferred duplications and gene losses are 99.89% and 99.72%, respectively, again caused by a big increase of inferred solutions for gene trees with no solution at tolerance 1×10^{-3} and a found solution at tolerance 0.1, where some of these solutions have incorrectly detected duplications and gene losses.

The step set to 0.5 is big enough that our software is not capable to find a solution for gene trees with tolerance from 0.0 till 1×10^{-5} include. It finds solutions for 81.97% of the gene trees from the dataset at tolerance 1×10^{-3}. For the tolerance 0.1 and more, the percentage of gene trees without a solution is 0%. The precision of correctly inferred root is best at tolerances set to 1×10^{-3} and 0.1 as in the step setting

Table 4.3: Flies: results of phylogenetic software on simulated dataset with rerooting the gene tree

Software[a]	**W/o sol**[b]	**Root**[c]	**Duplication**[d]		**Gene loss**[e]		**Runtime**[f]
			Prec	**Sens**	**Prec**	**Sens**	
Our (t - 0.0; s - 0.0)	100	-	-	-	-	-	0.004377
Our (t - 1×10^{-6}; s - 0.0)	100	-	-	-	-	-	0.002373
Our (t - 1×10^{-5}; s - 0.0)	100	-	-	-	-	-	0.002402
Our (t - 1×10^{-3}; s - 0.0)	100	-	-	-	-	-	0.002352
Our (t - 0.1; s - 0.0)	17.63	98.64	25.42	98.58	8.99	99.42	0.004743
Our (t - 0.3; s - 0.0)	2.47	99.47	45.56	99.44	17.61	80.86	0.005165
Our (t - 0.5; s - 0.0)	2.32	99.39	45.63	99.35	17.60	80.99	0.005745
Our (t - 1.0; s - 0.0)	0.48	99.98	46.37	99.98	23.23	91.98	0.005148
Our (t - 0.0; s - 0.1)	98.35	100	100	100	100	100	0.213234
Our (t - 1×10^{-6}; s - 0.1)	98.3	100	100	100	100	100	0.315081
Our (t - 1×10^{-5}; s - 0.1)	98.3	100	100	100	100	100	0.237645
Our (t - 1×10^{-3}; s - 0.1)	0.28	100	34.6	100	17.03	100	0.460089
Our (t - 0.1; s - 0.1)	0	100	100	100	100	100	0.500895
Our (t - 0.3; s - 0.1)	0	99.93	99.95	99.95	100	99.94	0.622866
Our (t - 0.5; s - 0.1)	0	99.81	99.86	99.86	100	99.83	0.777619
Our (t - 1.0; s - 0.1)	0	99.65	99.76	99.76	100	99.70	0.914920
Our (t - 0.0; s - 0.3)	98.35	100	100	100	100	100	0.042934
Our (t - 1×10^{-6}; s - 0.3)	98.3	100	100	100	100	100	0.046455
Our (t - 1×10^{-5}; s - 0.3)	98.3	100	100	100	100	100	0.047223
Our (t - 1×10^{-3}; s - 0.3)	94.03	100	25.77	100	17.15	100	0.047767
Our (t - 0.1; s - 0.3)	0	100	99.91	100	99.76	100	0.100275
Our (t - 0.3; s - 0.3)	0	99.96	99.96	99.96	100	99.95	0.121550
Our (t - 0.5; s - 0.3)	0	99.89	99.91	99.91	100	99.88	0.144801
Our (t - 1.0; s - 0.3)	0	99.77	99.82	99.82	100	99.77	0.172164
Our (t - 0.0; s - 0.5)	100	-	-	-	-	-	0.027602
Our (t - 1×10^{-6}; s - 0.5)	100	-	-	-	-	-	0.027832
Our (t - 1×10^{-5}; s - 0.5)	100	-	-	-	-	-	0.028455
Our (t - 1×10^{-3}; s - 0.5)	18.03	100	31.96	100	12.33	100	0.052290
Our (t - 0.1; s - 0.5)	0	100	100	100	100	100	0.059033
Our (t - 0.3; s - 0.5)	0	99.96	99.96	99.96	100	99.95	0.072567
Our (t - 0.5; s - 0.5)	0	99.92	99.93	99.93	100	99.91	0.088120
Our (t - 1.0; s - 0.5)	0	99.83	99.86	99.86	100	99.81	0.109294
Our (t - 0.0; s - 1.0)	100	-	-	-	-	-	0.014088
Our (t - 1×10^{-6}; s - 1.0)	100	-	-	-	-	-	0.017554
Our (t - 1×10^{-5}; s - 1.0)	100	-	-	-	-	-	0.015989
Our (t - 1×10^{-3}; s - 1.0)	18.32	100	31.91	100	12.26	100	0.028652
Our (t - 0.1; s - 1.0)	0	100	99.77	100	99.42	100	0.031903
Our (t - 0.3; s - 1.0)	0	99.97	99.95	99.97	99.96	99.96	0.038490
Our (t - 0.5; s - 1.0)	0	99.94	99.94	99.94	100	99.92	0.045830
Our (t - 1.0; s - 1.0)	0	99.89	99.90	99.90	100	99.87	0.053317
Our (t - 0.0; s - 2.0)	100	-	-	-	-	-	0.008342
Our (t - 1×10^{-6}; s - 2.0)	100	-	-	-	-	-	0.009238
Our (t - 1×10^{-5}; s - 2.0)	100	-	-	-	-	-	0.009383
Our (t - 1×10^{-3}; s - 2.0)	99.97	100	50	100	60	100	0.009198
Our (t - 0.1; s - 2.0)	0	99.29	94.95	99.26	87.47	100	0.018753
Our (t - 0.3; s - 2.0)	0	99.87	99.17	99.87	98.02	99.96	0.022397
Our (t - 0.5; s - 2.0)	0	99.95	99.95	99.95	100	99.94	0.025403
Our (t - 1.0; s - 2.0)	0	99.93	99.93	99.93	100	99.91	0.028850
Notung	0	92.63	99.98	99.98	93.32	93.3	1.689122
Treerecs	0	81.07	99.09	99.09	96.12	95.14	0.081646
TreeFix	0	99.98	98.98	98.94	100	98.68	4.065790

[a] Phylogenetic software with "*t*" as tolerance setting and "*s*" as step setting.
[b] Percentage of gene trees without a solution.
[c] Precision of correctly inferred root.
[d] Precision ("*Prec*") and sensitivity ("*Sens*") of inferred duplications.
[e] Precision ("*Prec*") and sensitivity ("*Sens*") of inferred gene losses.
[f] Average runtime of computing the reconciliation for each gene tree in seconds.

before. It begins decreasing with tolerance 0.3 and more, where it finds solutions with better reconciliation score while rooting the gene trees at other edges. The precision of inferred duplications is low at tolerance 1×10^{-3}, where the gene trees are not able to fit right into the species tree and create more duplications and gene losses than are correct. The precision of inferred duplications decreases with increasing tolerance as the software finds different root and also different duplications on different nodes. The sensitivity of inferred duplications is perfect until tolerance set to 0.3 and more, where our software finds a root on a different edge than expected causing us not to find all correct duplications. The precision of inferred gene losses is low at tolerance 1×10^{-3} induces by the same cause as with the precision of inferred duplications. Besides, it is perfect for the rest of the tolerance settings. The sensitivity of inferred gene losses has the same development as the sensitivity of inferred duplications. At the tolerance 0.1, we see the same scenario as with the step set to 0.1 with the same tolerance, where our software infers reconciliation with correct root and perfect precision and sensitivity of duplications and gene losses.

The results with the step set to 1.0 have the same development in all categories as the results for the step setting 0.5 with exceptions for the precision of inferred duplications and gene losses at tolerance setting 0.1 and 0.3. With the tolerance of 0.1, the precision of duplications and gene losses is 99.77% and 99.42%, respectively, not 100% like with the step 0.5. This is caused by a big step, where we skip the perfect root which induces duplications and gene losses at wrong nodes. The precision of duplications and gene losses rises a little bit with tolerance 0.3 because the duplications and gene losses can find the right nodes, but it also decreases the precision of correctly inferred root as the higher tolerance allows to find rooting at other edges with better reconciliation score. From the tolerance set to 0.5, the precision of inferred gene losses is again 100%.

With the last step setting 2.0, our software finds reconciliation only for 0.03% of gene trees from the dataset. For tolerance 0.1 and more, we infer reconciliation for all gene trees. The precision of correctly inferred root is best with the tolerance 1×10^{-3} caused by a small amount of found reconciliation solutions. It decreases to 99.29 with finding the solution for all gene trees and slowly rises till tolerance 1.0, where it decreases because it finds a better rooting with a smaller reconciliation score. With smaller step settings, the decrease caused by better rooting starts with a tolerance of 0.3. The late decrease in precision of correctly inferred root is induced by a big step 2.0 because of which we skip a lot of possible roots. The similar progress is with the precision of inferred duplications, where it rises from tolerance 1×10^{-3} to 0.5 include and decreases with tolerance 1.0 by the same cause as the precision of correctly inferred root. The sensitivity of inferred duplications has the same progress as the precision of correctly inferred root as it is sensitive to the change in the rooting of the gene tree. The precision of inferred gene losses rises until it obtains 100% at tolerance 0.5. The sensitivity of inferred gene losses has the same development as in the previous step settings.

The common development of average running time of computing the reconciliation for each gene tree is the same for all steps and tolerances. It always increases with the increasing value of the tolerance and decreases with the increasing value of the step except for the running times of steps 0.0 and 0.1 as the average running times of step 0.0 are lower than the average running times of step 0.1.

To compare with other software, Notung, Treerecs and TreeFix are able to find a solution for all gene trees. We calculated our average values in all examined categories, where the percentage of gene trees without a solution is 0%. The average precision of correctly inferred root is 99.88%. The average precision and sensitivity of inferred duplications are 99.62% and 99.89%, respectively. The average precision and sensitivity of inferred gene losses are 99.23% and 99.92%, respectively. The average runtime of computing the reconciliation for each gene tree is 0.197452 seconds. Our average precision of correctly inferred root is better than Notung by 7.25% and Treerecs by 18.81%. The best results in the precision of correctly inferred root have TreeFix with 99.98%, which is better by 0.1% from our average. In the precision of inferred duplications, our average has better results than Treerecs by 0.53% and TreeFix by 0.64%. Our average is worse by 0.36% from Notung that has the best precision of inferred duplications. In the case of sensitivity of inferred duplications, our average is better than Treerecs by 0.8% and TreeFix by 0.95%. Our average is worse by 0.09% from Notung, which has the best sensitivity of inferred duplications. The precision of inferred gene losses is worse in Treerecs by 3.11% and in Notung by 5.91% from our average. The best is TreeFix with perfect precision better by 0.77% from our average precision of inferred gene losses. Our average sensitivity of inferred gene losses is the best compare to the other software. It is worse in TreeFix by 1.24%, in Treerecs by 4.78% and in Notung by 6.62%.Our average runtime of computing the reconciliation for each gene tree is better by 88.31% from Notung and by 95.14% from TreeFix. The fastest runtime has Treerecs, better by 58.65% from our average.

With the average values, our software is the best only in the category of sensitivity of inferred gene losses. However, we computed reconciliation for all gene trees from the dataset with the precision of correctly inferred root and precision and sensitivity of inferred duplications and gene losses to be 100% two times with tolerance set to 0.1 and step set to 0.1 and 0.5. This makes our software best in the precision of correctly inferred root, precision and sensitivity of inferred duplications and sensitivity of inferred gene losses. For the precision of correctly inferred gene losses, we split the first place with TreeFix.

Fungi dataset

The first step setting is 0.0, where we consider subdividing each edge $(u, v) \in E(G)$ right above the vertices u and v, so the size of the set of possible roots for each edge is 2. Our software is not able to find reconciliation with tolerance set to $0.0, 1 \times 10^{-6}, 1 \times 10^{-5}$ (Tab. 4.4). It finds solutions to the reconciliation for 0.02% of the gene trees at tolerance set to 1×10^{-3}. The percentage of gene trees without inferred reconciliation decreases with increasing tolerance. The precision of correctly inferred root is 100% with tolerance 1×10^{-3}, thus the percentage of gene trees without reconciliation solution is big. It drops with finding the reconciliation solution for more gene trees at tolerance 0.1, where most of the gene trees find root at the different edge. It increases with increasing tolerance. The precision of inferred duplications and gene losses has the same development. They are also perfect with tolerance 1×10^{-3} as the precision of correctly inferred root, because of the big percentage of gene trees without a solution. They drop with tolerance 0.1 as 3.33% of gene trees find a reconciliation solution and almost half of all solution do not find the correct rooting. The precision of duplications and gene losses is sensitive to the change in the percentage of gene trees without solution, caused by inferring reconciliation solutions for gene trees with no solution before. The gene trees, that have no solution before, do not fit correctly to the species tree inducing duplications and gene losses on wrong nodes. So when the percentage of gene trees without solution significantly drops between tolerance settings $0.1, 0.3$ and $0.5, 1.0$, the precision of inferred duplications and gene losses also drops. They increase between tolerances 0.3 and 0.5, where the decrease in the percentage of gene trees without a solution is not so big, only 7.18%. The sensitivity of inferred duplications has the same progress as the precision of correctly inferred root as it depends on changes in rooting of the gene trees. The sensitivity of inferred gene losses is perfect for tolerances 1×10^{-3} and 0.1, where the percentage of gene trees with a solution is small. It starts decreasing with tolerance set to 0.3 as the percentage of gene trees without solutions considerably decreases, where gene losses are inferred incorrectly due to incorrectly inferred duplications.

With the step set to 0.1, we do not find reconciliation solutions merely with a tolerance setting of 0.0. The percentage of gene trees without solutions considerably drops with tolerance 1×10^{-3} and is 0% from tolerance 0.1 and more. The precision of correctly inferred root is 100% for tolerances 1×10^{-6}, 1×10^{-5} and 1×10^{-3} and slowly decreases from tolerance 0.1 and more caused by increasing tolerance, where our software finds solutions to the reconciliation with a smaller number of duplications and gene losses on other edges than on the original edge. We could already observe this development of precision of correctly inferred root with the simulated flies dataset (Tab. 4.3). The precision of inferred duplications increases with increasing tolerance from 1×10^{-6} till 0.1 include causing more accurate mapping of duplications to the correct nodes. It decreases from 0.1 to 1.0 include induced by rooting at different edges, where the software finds reconciliation with a better score, but the duplications are on wrong nodes. The sensitivity of inferred duplications is perfect until tolerance 0.1, where it starts decreasing caused by the decrease in precision of correctly inferred root. The precision of inferred gene losses has similar progress as the precision of inferred duplications. It increases between tolerance settings 1×10^{-6} and 0.1. Unlike the precision of inferred duplications, it starts decreasing with tolerance between 0.5 and 1.0. The sensitivity of inferred gene losses is perfect for almost all tolerance settings. It only decreases with tolerance set to 1.0 caused by a quite big percentage of gene trees, that are rooted at a different edge than the original edge inducing different gene losses.

The results for step setting 0.3 has comparable development as results with step set to 0.1. The percentage of gene trees without a solution at tolerance 0.0 is also 100% as with the step 0.1 and in addition, our software does not find any solutions at tolerance 1×10^{-6} either. We find solutions at tolerance 1×10^{-5} and from tolerance 0.1 and more is the percentage of gene trees without solution 0%. The precision of correctly inferred root and precision and sensitivity of inferred duplications and gene losses has the same progress as with step 0.1. Only one exception rises, when the sensitivity of inferred gene losses slightly decrease by 0.01 for tolerance setting 1×10^{-3} induced by a bigger step, where we skip a solution containing correct gene loss.

For the step setting 0.5, our software is not able to find a solution for all gene trees with tolerance set to 0.0 and 1×10^{-6} like with the step 0.3. It finds a lot less solutions with tolerances 1×10^{-5} and 1×10^{-3} than with the previous step. The precision of correctly inferred root has the same development as with the step 0.3. To compare, we have higher precision of correctly inferred root as in the step 0.3 with the identical tolerance settings. The precision of inferred duplications with tolerance set to 1×10^{-5} is smaller than with the step before as the step is bigger and we skip solutions with correct duplications nodes. However, at tolerance 1×10^{-3}, it is bigger compare to step 0.3 because of the low percentage of gene trees with reconciliation solution. The sensitivity of inferred duplication is equal to the results from the previous step. The precision of inferred gene losses for tolerance 1×10^{-5} and 1×10^{-3} is bigger than with the step 0.3 because the percentage of gene trees without solutions is bigger, but it is smaller by 0.02 with tolerance 0.1 as the step is bigger and our software misses solutions with accurate gene losses locations. From tolerance 0.3 and more, the precisions of inferred gene losses values are alike with the step 0.3. The sensitivity of inferred gene losses is perfect for tolerance settings from 1×10^{-5} to 0.5 include. It decreases between tolerances 0.5 and 1.0 as with the step 0.3.

The percentage of gene trees without solution with step setting 1.0 and tolerance settings 1×10^{-5} and 1×10^{-3} is again bigger than with the previous step setting caused by an increased step. The values at other tolerances are not changed from step 0.5. The precision of correctly inferred root is generally higher compare to the step before caused by the increase of the step. The precision of inferred duplications is lower from the previous step with tolerance settings 1×10^{-5} and 1×10^{-3} and stays the same from 0.1 and more. The lower values are induced by a bigger percentage

Table 4.4: Fungi: results of phylogenetic software on simulated dataset with rerooting the gene tree

Software[a]	W/o sol[b]	Root[c]	Duplication[d]		Gene loss[e]		Runtime[f]
			Prec	Sens	Prec	Sens	
Our (t - 0.0; s - 0.0)	100	-	-	-	-	-	0.006002
Our (t - 1×10^{-6}; s - 0.0)	100	-	-	-	-	-	0.004688
Our (t - 1×10^{-5}; s - 0.0)	100	-	-	-	-	-	0.004670
Our (t - 1×10^{-3}; s - 0.0)	99.98	100	100	100	100	100	0.004678
Our (t - 0.1; s - 0.0)	96.65	56.94	42.27	92.28	17.59	100	0.005205
Our (t - 0.3; s - 0.0)	78.68	62.06	40.91	95.42	14.73	98.7	0.006203
Our (t - 0.5; s - 0.0)	71.50	88.68	48.59	95.45	20.14	95.22	0.006832
Our (t - 1.0; s - 0.0)	0.52	98.05	35.38	98.07	11.78	90.97	0.009312
Our (t - 0.0; s - 0.1)	100	-	-	-	-	-	1.001114
Our (t - 1×10^{-6}; s - 0.1)	99.95	100	56.67	100	27.78	100	1.031367
Our (t - 1×10^{-5}; s - 0.1)	99.65	100	61.14	100	30.30	100	1.057450
Our (t - 1×10^{-3}; s - 0.1)	0.02	100	86.57	100	72.07	100	2.106134
Our (t - 0.1; s - 0.1)	0	99.97	99.99	99.99	100	100	2.385887
Our (t - 0.3; s - 0.1)	0	99.53	99.81	99.81	100	100	2.947574
Our (t - 0.5; s - 0.1)	0	98.95	99.58	99.58	100	100	3.354926
Our (t - 1.0; s - 0.1)	0	94.97	98.18	98.18	99.99	99.61	4.346881
Our (t - 0.0; s - 0.3)	100	-	-	-	-	-	0.217207
Our (t - 1×10^{-6}; s - 0.3)	100	-	-	-	-	-	0.223558
Our (t - 1×10^{-5}; s - 0.3)	99.92	100	55.36	100	16.67	100	0.222740
Our (t - 1×10^{-3}; s - 0.3)	16.05	100	31.89	100	16.05	99.99	0.427555
Our (t - 0.1; s - 0.3)	0	99.97	99.99	99.99	100	100	0.491044
Our (t - 0.3; s - 0.3)	0	99.56	99.82	99.82	100	100	0.590541
Our (t - 0.5; s - 0.3)	0	98.98	99.58	99.58	100	100	0.688235
Our (t - 1.0; s - 0.3)	0	95.07	98.18	98.18	99.99	99.61	0.848628
Our (t - 0.0; s - 0.5)	100	-	-	-	-	-	0.125547
Our (t - 1×10^{-6}; s - 0.5)	100	-	-	-	-	-	0.134174
Our (t - 1×10^{-5}; s - 0.5)	99.95	100	51.35	100	23.4	100	0.131749
Our (t - 1×10^{-3}; s - 0.5)	90.27	100	64.06	100	42.17	100	0.153825
Our (t - 0.1; s - 0.5)	0	99.97	99.98	99.99	99.98	100	0.291047
Our (t - 0.3; s - 0.5)	0	99.58	99.82	99.82	100	100	0.355765
Our (t - 0.5; s - 0.5)	0	99.01	99.58	99.58	100	100	0.409451
Our (t - 1.0; s - 0.5)	0	95.17	98.18	98.18	99.99	99.61	0.535211
Our (t - 0.0; s - 1.0)	100	-	-	-	-	-	0.079327
Our (t - 1×10^{-6}; s - 1.0)	100	-	-	-	-	-	0.110613
Our (t - 1×10^{-5}; s - 1.0)	99.97	100	45.83	100	21.21	100	0.111556
Our (t - 1×10^{-3}; s - 1.0)	92.95	100	62.48	100	45.58	100	0.095454
Our (t - 0.1; s - 1.0)	0	99.96	99.98	99.98	99.99	100	0.257746
Our (t - 0.3; s - 1.0)	0	99.63	99.82	99.82	100	100	0.316109
Our (t - 0.5; s - 1.0)	0	99.09	99.58	99.58	100	100	0.219381
Our (t - 1.0; s - 1.0)	0	95.36	98.18	98.18	99.99	99.61	0.329572
Our (t - 0.0; s - 2.0)	100	-	-	-	-	-	0.064227
Our (t - 1×10^{-6}; s - 2.0)	100	-	-	-	-	-	0.052543
Our (t - 1×10^{-5}; s - 2.0)	99.97	100	45.83	100	21.21	100	0.047304
Our (t - 1×10^{-3}; s - 2.0)	97.05	100	62.72	100	42.06	100	0.051702
Our (t - 0.1; s - 2.0)	0	99.98	99.95	99.98	99.82	100	0.112863
Our (t - 0.3; s - 2.0)	0	99.66	99.80	99.82	99.77	100	0.128979
Our (t - 0.5; s - 2.0)	0	99.17	99.57	99.57	100	100	0.139579
Our (t - 1.0; s - 2.0)	0	95.69	98.19	98.19	99.99	99.61	0.168751
Notung	0	96.92	99.76	99.74	97.11	96.75	1.887500
Treerecs	0	60.75	92.77	92.75	98.8	94.86	0.099334
TreeFix	0	99.22	95.83	95.69	99.77	94.42	10.510925

[a] Phylogenetic software with *"t"* as tolerance setting and *"s"* as step setting.
[b] Percentage of gene trees without a solution.
[c] Precision of correctly inferred root.
[d] Precision (*"Prec"*) and sensitivity (*"Sens"*) of inferred duplications.
[e] Precision (*"Prec"*) and sensitivity (*"Sens"*) of inferred gene losses.
[f] Average runtime of computing the reconciliation for each gene tree in seconds.

of gene trees without solutions, where we find solutions with more duplications on incorrect nodes. The sensitivity is almost the same with the step 0.5 except small decrease by 0.01 at a tolerance of 0.1. The precision of inferred gene losses compare to the step 0.5 is also lower with tolerance 1×10^{-5}, but higher at tolerance 1×10^{-3} and 0.1. This irregularity is caused by the bigger step, when we infer fewer solutions per gene tree, thus we skip some wrongly inferred gene losses. The results for tolerances 0.3 and more are the same as the previous step. The sensitivity of inferred gene losses remains the same as with step 0.5.

With the last step setting 2.0, the development of all categories is the same as with the previous step, but the values are slightly different due to the higher step, where we skip a lot of possible solutions. Our software has the same results for the percentage of gene trees without solution as with the step before with one exception at tolerance 1×10^{-3}, where we find a solution for only 2.95% of all gene trees. The precision of correctly inferred root remains 100% for tolerances 1×10^{-5} and 1×10^{-3} and increases for the tolerances 0.1 and more. The precision and sensitivity of inferred duplications and gene losses stay the same for tolerance 1×10^{-5}, as the percentage of gene trees without solutions does not change either. The precision of inferred duplications increases with the tolerance increase till tolerance 0.1 and then decreases caused by rooting the gene trees at a different edge with a better reconciliation score. The decrease from tolerance 0.1 and more is also in the sensitivity of inferred duplications that is perfect for tolerances 1×10^{-5} and 1×10^{-3}. The precision of inferred gene losses is smaller from the previous step but has an increasing tendency from tolerance 1×10^{-5} to 0.5, where it is 100%. It decreases from the tolerance 0.5 and more. The sensitivity of inferred gene losses does not change compared to step 1.0.

The average running time of computing the reconciliation for each gene tree at all steps and tolerances is generally increasing with the increasing tolerance and decreases with increasing step except the average running times in steps 0.0 and 0.1, same as with the flies dataset.

As in the flies dataset, we calculated our average values for all categories, where the percentage of gene trees without a solution is 0% as Notung, Treerecs and TreeFix find solutions for all gene trees in the dataset. The average precision of correctly inferred root is 99.46%. The average precision and sensitivity of inferred duplications are 99.39%

for both. The average precision and sensitivity of inferred gene losses are 99.99% and 99.9%, respectively. The average running time of computing the reconciliation for each gene tree is 0.945909 seconds. Our average precision of correctly inferred root is the best among the tested software. It is better by 0.24% from TreeFix, by 2.54% from Notung and by 38.71% from Treerecs. In terms of precision of inferred duplications, our average has better results than TreeFix by 3.56% and Treerecs by 6.62%. The best precision of inferred duplications has Notung with 99.76%, which is better than our average by 0.37%. In the sensitivity of inferred duplications, our average is again better from Treefix by 3.7% and from Treerecs by 6.64%, while Notung has the best sensitivity of inferred duplications, where our average is worse by 0.35%. Our average is also best in the precision of inferred gene losses, where the TreeFix is worse by 0.22%, Treerecs is worse by 1.19% and Notung is worse by 2.88%. In the case of the sensitivity of inferred gene losses, our average is best too. It is better by 3.15% from Notung, by 5.04% from Treerecs and by 5.48% from TreeFix. Our average running time of computing the solution for each gene tree is better by 49.89% from Notung and by 99.05% from TreeFix. The fastest running time has Treerecs with 0.099334, which is better than our average by 89.5%.

Our average values are the best in the precision of correctly inferred root and the precision and sensitivity of inferred gene losses. However, our best result is with tolerance set to 0.1 and step settings 0.1 and 0.3, where the precision and sensitivity of inferred duplications is 99.99% for both, which is better than the results of Notung and makes our software the best in all categories.

Conclusion

In conclusion, for both datasets, the best tolerance setting to infer isometric reconciliation for all gene trees is between 1×10^{-3} and 0.1 as the percentage of gene trees without solutions is never 0% at tolerance 1×10^{-3} and is always 0% at tolerance 0.1. With the lower setting of tolerance, we do not find solutions for all gene trees and contrarily, with the higher setting of tolerance, we find solutions at a different edge than the original edge, but with a better reconciliation score. The best tolerance for inferring the gene trees with the correct root is also somewhere between tolerances 1×10^{-3} and 0.1. The precision of correctly inferred root is always 100% until tolerance 1×10^{-3} (or with flies in some cases until tolerance 0.1), where the percentage of gene trees with a solution is not 0% and it decreases with the increasing tolerance. The precision and sensitivity of inferred duplications depend mostly on the precision of correctly inferred root means that when the precision of correctly inferred root decreases, the precision and sensitivity of inferred duplications decrease with it. The precision and sensitivity of gene losses show a higher number of 100% values, but as they depend on the correctly inferred duplications, they also depend on the precision of correctly inferred root. The average running time for computing the reconciliation for each gene tree in both datasets increases as the step decreases and decreases as the tolerance increases. This rule does not apply to step 0.0 as for each edge, the set of possible roots is of size 2, thus the running time is quite fast.

The best settings to infer the correct reconciliation on these datasets is tolerance set between 1×10^{-3} and 0.1 as we explained and step 0.1 as our software finds the best solutions for tolerance 0.1 and steps 0.1 and 0.5 on the flies dataset and for the same tolerance and steps 0.1 and 0.3 on the fungi dataset. These best solutions are also best in comparison with the other software: Notung, Treerecs and Treefix.

4.2 Real dataset

The correct gene tree is unknown in the real dataset, thus we use different metrics as with the simulated dataset. We measure the number of inferred duplications and gene losses overall gene trees in the dataset and the consistency score of duplications.

The duplication consistency score, dcs, shows the plausibility of inferred duplications. For a duplication node u with children v and w, the duplication consistency score $dcs(u) = (L \cup R) \mid (L \cap R)$ is computed as the union of L and R over intersection of L and R, where L is the set of species represented in left child v and R is the set of species represented in right child w.

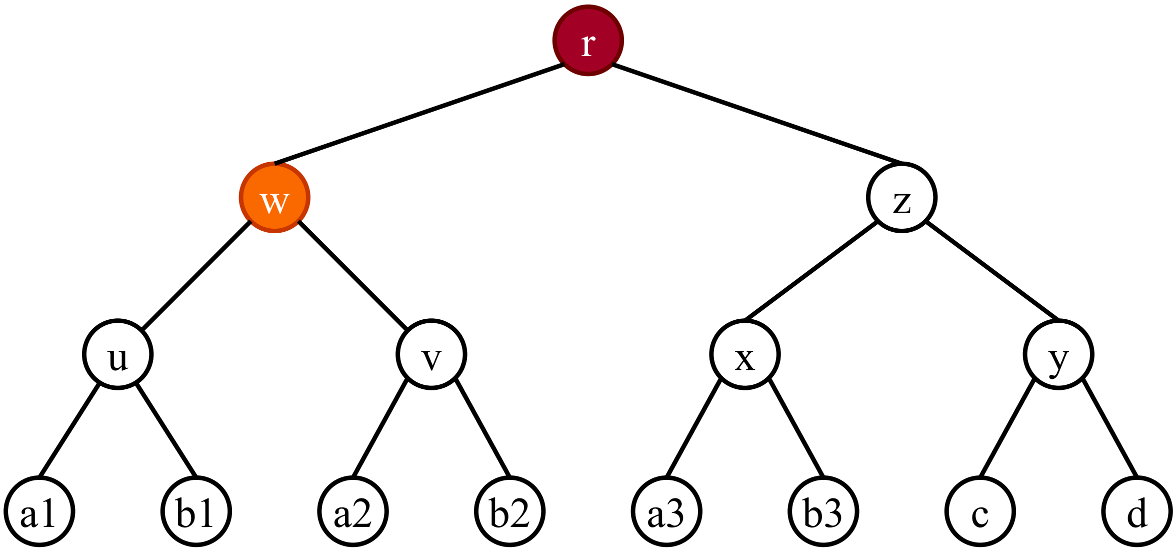

Figure 4.1: **Duplication consistency score for node w**
The set of species of left child u is $L = \{A, B\}$ and the set of species of right child v is $R = \{A, B\}$. The duplication consistency score is computed as: $dcs(w) = (L \cup R) \mid (L \cap R) = 2 \div 2 = 1$.
Duplication consistency score for node r
The set of species of left child w is $L = \{A, B\}$ and the set of species of right child z is $R = \{A, B, C, D\}$. The duplication consistency score is computed as: $dcs(r) = (L \cup R) \mid (L \cap R) = 2 \div 4 = 0.5$.

Conclusion

Appendix

Bibliography

[1] Mukul Bansal. Tutorial: Treefix and treefix-dtl. Available from: `http://compbio.mit.edu/treefix/tutorial.html`, 2014. [Accessed 12 Jan 2021].

[2] Bastien Boussau. Phyldog: joint reconstruction of species and gene phylogenies. Available from: `https://pbil.univ-lyon1.fr/software/phyldog/`, 2012. [Accessed 12 Jan 2021].

[3] Bastien Boussau et al. Genome-scale coestimation of species and gene trees. *Genome Research*, 23(2):323–330, Feb. 2013.

[4] Broňa Brejová et al. Isometric gene tree reconciliation revisited. *Algorithms for Molecular Biology*, 12(1):17, Jun. 2017.

[5] Radoslav Chládek. Algorithms for isometric gene tree reconciliation. Master's thesis, Comenius University in Bratislava, 2019.

[6] Radoslav Chládek et al. Isometric gene tree reconciliation with software.interval software.edge lengths.

[7] Nicolas Comte et al. Treerecs: an integrated phylogenetic tool, from sequences to reconciliations. *bioRxiv*, Oct. 2019.

[8] Dave Danicic et al. *Notung 2.8: A Manual.* Notung Development Team, Mar. 2015.

[9] Jean-Philippe Doyon, Cedric Chauve, and Sylvie Hamel. Algorithms for exploring the space of gene tree/species tree reconciliations. In *Nelson C.E., Vialette S. (eds) Comparative Genomics. RECOMB-CG 2008. Lecture Notes in Computer Science*, volume 5267, pages 1–13. Springer-Verlag, Berlin, Heidelberg, Oct. 2008.

[10] Joseph Felsenstein. *Inferring Phylogenies.* Sinauer, Sunderland, 2003.

[11] Morris Goodman et al. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology*, 28(2):132–163, Jun. 1979.

[12] Damir Hasić and Eric Tannier. Gene tree species tree reconciliation with gene conversion. *Journal of Mathematical Biology*, 78(6):1981–2014, May 2019.

[13] Li Heng. Treesoft: Treebest. [online]. Available from: `http://treesoft.sourceforge.net/treebest.shtml`. [Accessed 12 Jan 2021].

[14] INRIA. Tutorial – treerecs. Available from: `https://project.inria.fr/treerecs/tutorial/`, 2011. [Accessed 12 Jan 2021].

[15] Jian Ma et al. The infinite sites model of genome evolution. *Proceedings of the National Academy of Science*, 105(38):14254–14261, Sep. 2008.

[16] Matthew D. Rasmussen. Spimap documentation. Available from: `http://compbio.mit.edu/spimap/pub/spimap/doc/spimap-manual.html`, 2011. [Accessed 12 Jan 2021].

[17] Matthew D. Rasmussen and Manolis Kellis. A bayesian approach for fast and accurate gene tree reconstruction. *Molecular Biology and Evolution*, 28(1):273–290, Jan. 2011.

[18] Benjamin Vernot et al. Reconciliation with non-binary species trees. *Journal of Computational Biology*, 15(8):981–1006, Oct. 2008.

[19] Albert J. Vilella et al. Ensemblcompara genetrees: Complete, duplication-aware phylogenetic trees in vertebrates. *Genome Research*, 19(2):327–335, Feb. 2009.

[20] Wu Yi-Chieh et al. Treefix: Statistically informed gene tree error correction using species trees. *Systematic Biology*, 62(1):110–120, Jan. 2013.

[21] Wu Yi-Chieh et al. Treefix. Available from: `https://www.cs.hmc.edu/~yjw/software/treefix/`, 2014. [Accessed 12 Jan 2021].