國立臺灣大學電機資訊學院資訊工程學系

博士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Doctoral Dissertation

在基因複製模型下之多重基因複製與親緣樹建立相關的最佳化問題

Some Optimization Problems Related to Multiple Gene Duplications and Phylogenetic Tree Construction under the Gene Duplication Model

羅正偉

Cheng-Wei Luo

指導教授：趙坤茂 博士

Advisor: Kun-Mao Chao, Ph.D.

中華民國 99 年 7 月

July, 2010

誌謝

時間過的很快，不知不覺六年過去了，許多事情都改變了，然而唯一不變的是一直陪伴著我的趙坤茂老師以及 ACB 這個大家庭。

回想起六年前，我何其有幸接受了趙老師的指導，讓我從懵懂的碩士班新生，變成了偉哥偉教授，甚至到現在的偉博士，這一切都得感謝趙老師如慈父般的鼓勵與指導，而正如這六年的旅程無法在這一篇短短的誌謝完全表達一般，我心中對老師的感激之意也無法只是在三言兩語內訴盡，但那樣的感激之情卻一直在我心中迴盪不去，詞拙的我此刻能表達出來的只有：老師謝謝您，這六年您辛苦了！

感謝所有曾經陪伴過我的 ACB 夥伴們：耀廷、佳熒、容任、效飛、弘倫、冠宇、志亘、小錦、明江、怡靜、大秉、爽哥、榮哥、JuJu、一軒、謦儀、帥朋、Roger、Popple、晏禕、智鐸、韋仰、煜樟、小孟、安強、陳琨、韋霖、佑任、彥緯、霈君、蔚茵、秋芸、稚穎、峻偉、聖耀。你們不論何時總是陪在我身旁，是我這六年能撐下去的最大動力，與你們在一起的回憶是最美好也最甘甜的，若沒有你們的參與，我想這段旅程應該是黑白的吧。

感謝呂育道教授在論文計劃審查與口試期間，花了許多時間閱讀並修改我的論文，同時也指出了許多缺失。感謝歐陽彥正教授在論文計劃審查時勉勵我在研究上要勇於成爲先驅，而不僅僅是改進前人的結果。感謝傅楸善教授、呂學一教授以及曾宇鳳教授在百忙之中撥冗來參與我的論文計劃審查與口試，同時也提出了許多寶貴的建議，使我的論文能更臻完美。

此外也感謝這段時間伴我成長的人：俊德、勇慶、經略、昱豪以及荻嘉，你們使得我貧乏的心靈富足了起來，在我開心的時候一起分享，在我無助的時候熱心幫助我，我能夠順利畢業你們絕對是背後的無名英雄。

感謝這段時光女友玥伶的陪伴，給予我精神上無比的支持，她的存在如同夏日朝陽般耀眼明亮，使我對未來的人生又重拾起許多希望。

最後要感謝我的家人，沒有你們我絕對沒辦法完成這段艱苦的求學歷程，你們給我的支持與鼓勵是我這輩子都算不清也還不清的，這樣的恩情我永生難報，只盼未來我能夠將所學回報給這國家社會，而不辜負你們對我的期望。

今天是八月五號父親節前夕，爸！我終於拿到您期盼許久的博士學位，不能

讓您在世時見到披著博士袍的我，這一直是我心中最大的遺憾。如今我終於拿到了博士學位，我想這份榮耀最終得歸於您，若您九泉之下有知，也請替我高興吧！您的恩情我今生無法回報，來生縱使是當犬馬也必定回報您的恩情。最後請讓我再對您說一聲：父親節快樂！

最後祝福所有在我這段旅程中出現過的人們都能夠平安幸福！也祝福所有爸爸父親節快樂！

摘要

本論文探討在基因演化模型下之多重基因複製與親緣樹建立相關的最佳化問題。針對多重基因複製，我們探討了事件叢集問題 (Episode-Clustering Problem) 與最小事件問題 (Minimum Episodes Problem)。對於事件叢集問題，我們將 Burleigh 等人的結果改進到最佳的線性時間演算法；而對於最小事件問題，以 Bansal 與 Eulenstein 所提出的演算法爲基礎下，我們也提出了最佳的線性時間演算法。針對親緣樹的建立，我們探討了複製－遺失問題 (Duplication-Loss Problem)。由於複製－遺失問題是 NP 困難，因此在實際應用上都使用啓發式方法來解此問題。標準的啓發式方法是在樹狀空間上反覆執行局部搜尋直到局部最小值被算出。以最近鄰居交換 (NNI) 的局部搜尋爲基礎下，本論文探討了複製－遺失問題的啓發式方法，同時也對相應的局部搜尋問題提出了一個線性演算法。

Abstract

This dissertation studies several optimization problems related to multiple gene duplications and phylogenetic tree construction under the Gene Duplication model. For multiple gene duplications, we study the Episode-Clustering (EC) problem and the Minimum Episodes (ME) problem. For the EC problem, we improve the results of Burleigh *et al.* with an optimal linear-time algorithm. For the ME problem, on the basis of the algorithm presented by Bansal and Eulenstein, we propose an optimal linear-time algorithm. For the phylogenetic tree construction, we study the Duplication-Loss problem. Since the Duplication-Loss problem is NP-hard, heuristics are developed to solve it in practice. A standard heuristic is to perform the stepwise local search on the tree space until a local minimum is reached. In this dissertation, we study the heuristic for the Duplication-Loss problem based on NNI local searches and propose a linear-time algorithm for the corresponding Local Search problem.

Contents

List of Figures

Chapter 1

Introduction

1.1 Biological Background

In evolutionary molecular biology, phylogenetic analysis helps to realize the evolutionary relationship among various organisms. As genomic sequences are easier to obtain, these data provide sufficient material to conduct large-scale phylogenetic analysis among different species. One approach for using these genomic data is to employ the idea of gene trees. Given a gene family[*] for a set of species, a *gene tree* is a tree that depicts the phylogeny among the genes of the gene family sampled from the set of species, while a *species tree* is a tree that represents the phylogeny of a given set of species. Given a gene tree, the homologous genes in this gene tree are assumed to evolve in the same way as those species from which these homologous genes are sampled. In other words, the gene tree and the corresponding species tree are assumed to have the same topology. However, complicated evolutionary processes, such as gene duplication, loss, recombination, and horizontal gene transfer, generate gene trees that differ from species trees [36, 43, 53, 56]. That is to say, gene trees and species trees are *inconsistent*. In order to reconstruct the evolutionary history, it is important for evolutionary biologists to explain the inconsistency between gene trees and species trees.

Gene duplication is an evolutionary process during which one or more genes of a genome are duplicated, while loss is an evolutionary process during which one or more genes are eliminated. Many evidences have shown that gene duplications play a major role in the evolution of species

[*]A gene family is a set of homologous genes assumed to derive from a common ancestor.

on Earth [12, 13, 29, 37, 38, 44, 45, 47, 48, 51]. Therefore, utilizing gene duplication and loss is an applicable approach to explaining the inconsistency between gene trees and species trees. Goodman *et al.* proposed the Gene Duplication (GD) model to explain the inconsistency between gene trees and species trees by postulating gene duplications and losses [25]. The GD model has been well studied [14, 18, 26, 27, 35, 39, 57, 58]. Goodman *et al.* [25] used the concept of a reconciled tree to reconcile the inconsistent gene trees and a species tree based on the GD model. The reconciled tree provides the mapping between genes trees and a species tree, and explains the inconsistency in the evolutionary history by postulating gene duplications and losses. More details about the GD model are formally described in Section 1.2.2. In this dissertation, we mainly study some optimization problems under the GD model. According as the species tree is given or unknown, these optimization problems, for a given set of gene trees, can be categorized into two parts respectively: (1) multiple gene duplication problems, and (2) phylogenetic tree construction problems.

1.1.1 Multiple Gene Duplication Problems

For a large-scale genome duplication, many gene duplications are parts of large multiple gene duplication events during which a large portion of an organism's genome is duplicated. In order to distinguish gene duplication events and genome duplications, Page and Cotton [42] introduced the term "episode" for gene duplications in different gene trees, explainable by a single gene duplication event. That is, the gene duplications of different gene trees are not necessarily independent events since they may result from the same gene duplication event (episode). Unfortunately, since gene losses occur following gene duplications, it is difficult to detect the number and location of multiple gene duplication episodes.

A series of studies, based on the GD model, have focused on the Multiple Gene Duplication (MGD) problem in order to further understand the number and location of the multiple gene duplication episodes in the evolutionary history (e.g. [7, 16, 17, 22, 27, 42]). Given a set of gene trees and a species tree, the goal of the MGD problem is to map each gene duplication in gene trees onto a species tree and to locate the multiple gene duplication episode such that the total number of multiple gene duplication episodes is minimized. Guigó *et al.* [27] investigated the phylogenetic issues on multiple gene duplications on the basis of the GD model, and addressed the MGD problem. They set the range on the location of each gene duplication according to the mapping between gene trees and a species tree. Each gene duplication in the gene trees can be placed on any species in the species tree within a path between the two most recent species containing the duplication and its parent, respectively. If the parent does not exist, the path is located between the most recent species for the duplication and the root of the species tree. With the definitions of the location of each gene duplication, they proposed one formulation of the MGD problem: the EPISODE-CLUSTERING (EC) problem [27]. Given a set of gene trees and a species tree, the EC problem is to find a minimum number of locations in the species tree for placing all gene duplications in the gene trees. For the EC problem, however, Guigó *et al.* only gave some hints on how to solve this problem.

Page and Cotton [42] defined the EC problem introduced by Guigó *et al.* more formally, and presented a heuristic for this problem. Fellows *et al.* [22] also proposed another version of the MGD problem, and proved this version to be NP-hard. Recently, Burleigh *et al.* [17] revisited the EC problem and gave an exact algorithm for this problem. Afterwards, Bansal and Eulenstein [7] also gave comprehensive explorations of the MGD problem and defined a new version of the MGD problem: the MINIMUM EPISODES (ME) problem. Given a set of gene trees and a species tree, the ME problem is to assign all gene duplications to nodes in a species tree such that the total number of multiple gene duplication episodes is minimized. Note that Guigó *et al.* [27] and Page and Cotton [42] also attempted to solve the ME problem but turned out to solve the EC problem essentially. In this dissertation, we study the EC and ME problems and present linear-time algorithms for the two problems.

1.1.2 Phylogenetic Tree Construction Problems

When no preliminary knowledge about the species tree is given, a natural problem in evolutionary molecular biology is to construct a species tree among various species from a set of gene trees. One possible approach is to reconcile gene trees with species trees under the parsimonious criterion of minimizing the number of gene duplications and losses (also known as the *mutation cost*) based on the GD model proposed by Goodman *et al.* [25]. The corresponding problem is called the Duplication-Loss problem [27]. Other approaches use probabilistic models to reconcile gene trees with species trees [2, 3].

Given a set of gene trees, the Duplication-Loss problem is to infer a comparable species tree minimizing the mutation cost. A special case of the Duplication-Loss problem, the Gene Duplication problem, is to infer a comparable species tree only minimizing the number of gene duplications (also called the *duplication cost*) [27]. The decision versions of the two problems are NP-hard [34], and some parameterized problems are fixed-parameter tractable [30, 50]. Recently, Bansal and Shamir [10] showed that the Gene Duplication problem cannot be approximated to within a logarithmic factor unless P = NP. In practice, therefore, heuristics are applied to solving these problems to conduct large-scale species tree construction.

Commonly-used heuristics for the Duplication-Loss and Gene Duplication problems are to search all possible species trees by solving a series of instances of the Local Search problem [40, 54]. Given a tree edit operation and a cost criterion, the Local Search problem is to find an optimal tree S^* in the neighborhood of S under the given cost criterion. Given a tree S and a tree edit operation, the *neighborhood* of S is a set of trees which can be transformed by performing the given tree edit operations on S. Given a set of gene trees $\mathcal{G}$ and an initial species tree S, the heuristic for the Duplication-Loss and Gene Duplication problems proceeds as follows. For the first local search, we solve an instance of the Local Search problem to find an optimal tree S' in the neighborhood of the initial species tree S. Then the optimal tree S' is used as the initial species tree in the second local search, and the process repeats until a local minimum is obtained. Many studies have proven that these heuristics have much potential for constructing correct species trees (e.g. [20, 41, 42, 46, 49]).

There are several tree edit operations, such as the nearest neighbor interchange (NNI) operation [1, 8, 15], the subtree pruning and regrafting (SPR) operation [1, 4, 5, 15, 52, 55], and the tree bisection and reconnection (TBR) operation [1, 6, 19, 52]. For the Gene Duplication problem, efficient algorithms for the corresponding local search problems have been proposed based on the NNI, SPR, and TBR operations [5, 6, 8]. To the best of our knowledge, however, the corresponding local search problems for the DUPLICATION-LOSS problem only have been considered based on the SPR and TBR operations by Bansal *et al.* [4]. In this dissertation, we study the heuristic for the DUPLICATION-LOSS problem based on the NNI operation and present a linear-time algorithm for the corresponding local search problem.

1.2 Preliminaries

In the following, we introduce necessary definitions and notation based on those from [7] for later discussions. Since the EC, ME, and DUPLICATION-LOSS problems are all based on the GD model, we review some definitions and concepts related to the GD model.

1.2.1 Basic Definitions and Notations

A *tree* T is a connected, acyclic graph consisting of a node set $V(T)$ and an edge set $E(T)$. T is *rooted* if it has exactly one distinguished node called the *root*, denoted by $Ro(T)$. Given a rooted tree T, we denote by $\leq_T$ the partial order on $V(T)$, and say $x \leq_T y$ if y is a node on the path from $Ro(T)$ to x. A node with no children is called a *leaf*, and $Le(T)$ denotes the set of all leaves in T. If $(x, y) \in E(T)$ and $x \leq_T y$, then y is the *parent* of x, denoted by $Pa(x)$, and x is a *child* of y. Let the set of children of y be $Ch(y)$, and the left and right children of y are denoted by $Left(y)$ and $Right(y)$ respectively if T is a rooted binary tree. The length of the path from $Ro(T)$ to a node x, denoted by $d_T(x)$, is the *depth* of x in T. We denote by $l_T(x, y)$ the length of the unique path between x and y. The *least common ancestor* (LCA) of a node subset $L \subseteq V(T)$, denoted by $lca(L)$, is the node that is the ancestor of all nodes in L with the greatest depth. The *subtree* of T rooted at x, denoted by T_x, is the tree induced by all descendants of x. The *height* of a tree T, $h(T)$, is the number of nodes on a maximum-length path from $Ro(T)$ to a leaf node of T. A tree T is a *full binary* tree if each node in T is either a leaf or has two children. Unless specified otherwise, the tree refers to a rooted full binary tree

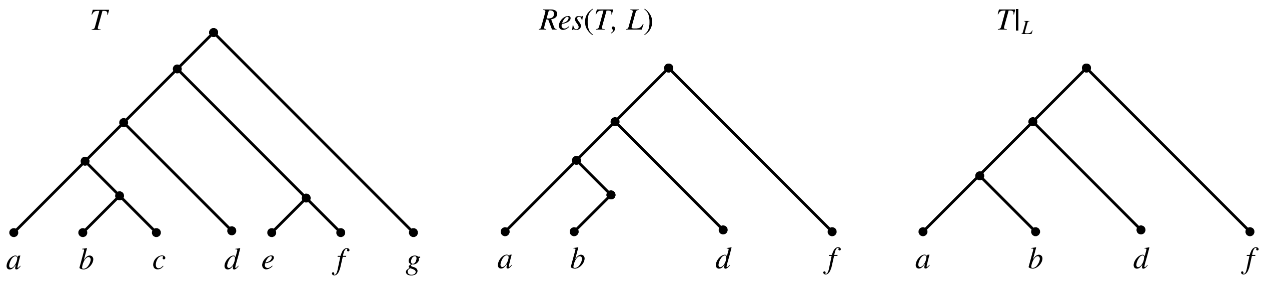

Figure 1.1: An illustration of the restriction and the homomorphic subtree of the tree T on the leaf set $L = \{a, b, d, f\}$.

throughout this dissertation.

Given a tree T and a set $L \subseteq Le(T)$, the *restriction* of T on L is the minimal subtree containing L as the leaf set, denoted by $Res(T, L)$. We define the *homomorphic subtree* $T|_L$ of T on the leaf set L to be the tree resulting from $Res(T, L)$ by contracting all nodes of degree two except $Ro(Res(T, L))$. See Figure 1.1 for an illustration. Given $x \leq_T y$, we define the interval $[x, y] = \{u \in V(T) | x \leq_T u \leq_T y\}$, and x and y are the *starting terminal* and the *ending terminal* of the interval $[x, y]$, respectively. Let $\mathcal{I}$ be a collection of intervals under the partial order $\leq_T$. A node set $U \subseteq V(T)$ is called a *cover* of $\mathcal{I}$ if for each interval $I \in \mathcal{I}$, there exists at least one node $v \in U$ such that $v \in I$. If U is a cover of minimum cardinality, we call U a *minimum cover* of $\mathcal{I}$. The *intersection graph* of a collection of intervals $\mathcal{I}$, denoted by $int(\mathcal{I})$, is the graph where $V(int(\mathcal{I})) = \mathcal{I}$ and $E(int(\mathcal{I})) = \{(I, I') | I, I' \in \mathcal{I} \text{ and } I \cap I' \neq \emptyset\}$.

1.2.2 The Gene Duplication Model

The Gene Duplication (GD) model was first introduced by Goodman *et al.* [25]. The model hypothesized that the inconsistency of gene trees and corresponding species tree is caused by a series of gene duplications and losses, and that each gene duplication can be placed on a specified interval on the species tree [25, 27, 57]. Given a set of n taxa, a *species tree* is a full binary tree, using these n taxa as leaves, which describes their evolutionary history. Given a gene family for a set of n taxa, a *gene tree* is a full binary tree that depicts the evolutionary history among the sequences of the gene family.

Let G and S be a gene tree and species tree, respectively. G and S are biologically related only if all genes in $Le(G)$ are sampled from the species in $Le(S)$. A *leaf-mapping* $\mathcal{L}_{G,S} : Le(G) \rightarrow Le(S)$ maps a gene $g \in Le(G)$ to a species $s \in Le(S)$. That is, a leaf-mapping specifies the species from which the gene was sampled. G and S are *comparable* if such a leaf-mapping $\mathcal{L}_{G,S}$ exists. Let $\mathcal{G}$ be a set of gene trees. $\mathcal{G}$ and S are comparable if each gene tree $G \in \mathcal{G}$ is comparable with S. For convenience, we define the set $\mathcal{L}^{-1}_{G,S}(s) = \{g | g \in Le(G)$ and $\mathcal{L}_{G,S}(g) = s\}$ for each leaf $s \in Le(S)$. For a set of gene trees $\mathcal{G}$ comparable to the species tree S, let $\mathcal{L}_{\mathcal{G},S} = \bigcup_{G \in \mathcal{G}} \mathcal{L}_{G,S}$ and $\mathcal{L}^{-1}_{\mathcal{G},S}(s) = \bigcup_{G \in \mathcal{G}} \mathcal{L}^{-1}_{G,S}(s)$ for each node $s \in Le(S)$. Unless specified otherwise, we assume that all given gene trees are comparable with S and denote by $\mathcal{G}$ a set of gene trees, where $G \in \mathcal{G}$ throughout this dissertation. To correlate a gene tree G with a species tree S, we require a function to map each gene g in $V(G)$ to the most recent species in S where g is involved.

Definition 1: Let G and S be a gene tree and species tree, respectively. Given a leaf-mapping $\mathcal{L}_{G,S}$ for G and S, the *LCA-mapping* $\mathcal{M}_{G,S} : V(G) \rightarrow V(S)$ of $\mathcal{L}_{G,S}$ is defined as $\mathcal{M}_{G,S}(g) = lca(\mathcal{L}_{G,S}(Le(G_g)))$ for each node $g \in V(G)$.

For the convenience, we define the set $\mathcal{M}^{-1}_{G,S}(s)$ to be $\{g | g \in V(G)$ and $\mathcal{M}_{G,S}(g) = s\}$ for each node $s \in V(S)$. For a set of gene trees $\mathcal{G}$ comparable to the species tree S, let $\mathcal{M}_{\mathcal{G},S} = \bigcup_{G \in \mathcal{G}} \mathcal{M}_{G,S}$ and $\mathcal{M}^{-1}_{\mathcal{G},S}(s) = \bigcup_{G \in \mathcal{G}} \mathcal{M}^{-1}_{G,S}(s)$ for each node $s \in V(S)$.

Definition 2: A node $y \in V(G)$ is a *gene duplication* if there exists a child x of y such that $\mathcal{M}_{G,S}(x) = \mathcal{M}_{G,S}(y)$. We denote by $Dup(G,S)$ the set of gene duplications in G with respect to S. Let $Dup(\mathcal{G},S) = \bigcup_{G \in \mathcal{G}} Dup(G,S)$ for a set of gene trees $\mathcal{G}$.

Definition 3: For each gene duplication $g \in Dup(G,S)$, the interval $I(g)$ specifies all possible placements of the gene duplication g onto the species tree and is defined as follows.

1. If $g = Ro(G)$, $I(g)$ is set to $[\mathcal{M}_{G,S}(g), Ro(S)]$.

2. If $\mathcal{M}_{G,S}(g) = \mathcal{M}_{G,S}(Pa(g))$, $I(g)$ is set to $[\mathcal{M}_{G,S}(g), \mathcal{M}_{G,S}(g)]$.

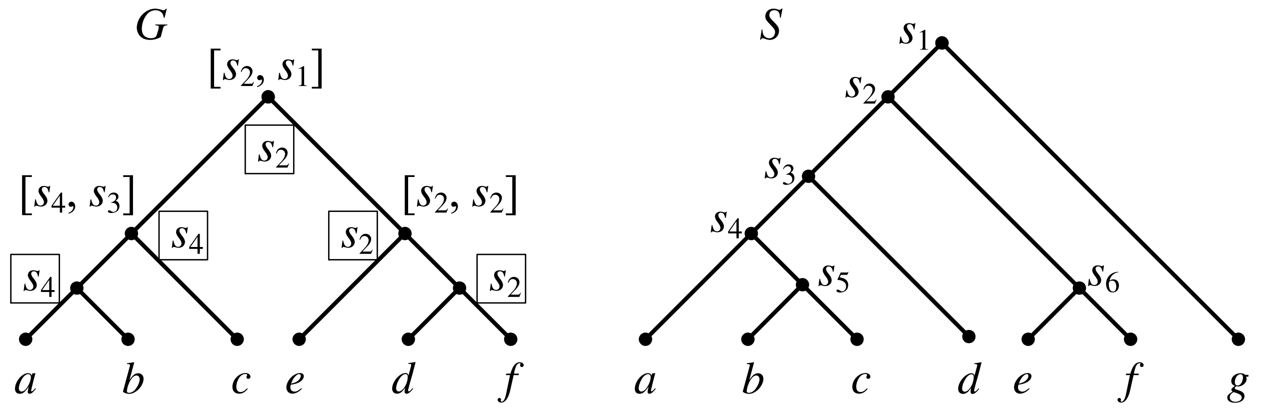

Figure 1.2: An illustration of a gene tree G and a comparable species tree S. For simplicity, the labels of leaves of G are replaced with the corresponding leaf-mapping. For each internal node in G, the boxed value denotes the LCA-mapping of the internal nodes of G. If a node of G is a gene duplication, the interval is also shown.

3. Otherwise, $I(g)$ is set to $[\mathcal{M}_{G,S}(g), z]$, where $z \in Ch(\mathcal{M}_{G,S}(Pa(g))) \cap [\mathcal{M}_{G,S}(g), \mathcal{M}_{G,S}(Pa(g))]$.

See Figure 1.2 for an illustration.

According to [27], we define the number of losses and the mutation cost as follows.

Definition 4: Let y be an internal node of the gene tree G and $\hat{S} = S|_{\mathcal{L}_{G,S}(Le(G))}$. The number of *losses* $Loss(G, S, y)$ associated to y is defined as follows.

$$Loss(G,S,y) = \begin{cases} 0 & \text{if } \mathcal{M}_{G,\hat{S}}(y) = \mathcal{M}_{G,\hat{S}}(Left(y)) \\ & \qquad\qquad = \mathcal{M}_{G,\hat{S}}(Right(y)); \\ \sum\limits_{z \in Ch(y)} |l_{\hat{S}}(\mathcal{M}_{G,\hat{S}}(y), \mathcal{M}_{G,\hat{S}}(z)) - 1| & \text{otherwise.} \end{cases}$$

Let $Loss(G, S) = \sum_{y \in V(G) \setminus Le(G)} Loss(G, S, y)$ be the total number of losses of G with respect to S, and $Loss(\mathcal{G}, S) = \sum_{G \in \mathcal{G}} Loss(G, S)$.

Definition 5: For a gene tree G and a species tree S, $Mut(G, S) = |Dup(G, S)| + Loss(G, S)$ is the *mutation cost* of G with respect to S. We also let $Mut(\mathcal{G}, S) = \sum_{G \in \mathcal{G}} Mut(G, S)$ for a set of gene trees $\mathcal{G}$ and a species tree S.

An example is shown in Figure 1.3 to describe how the GD model explains the inconsistency between a gene tree G and a species tree S by postulated gene duplications and losses. R is

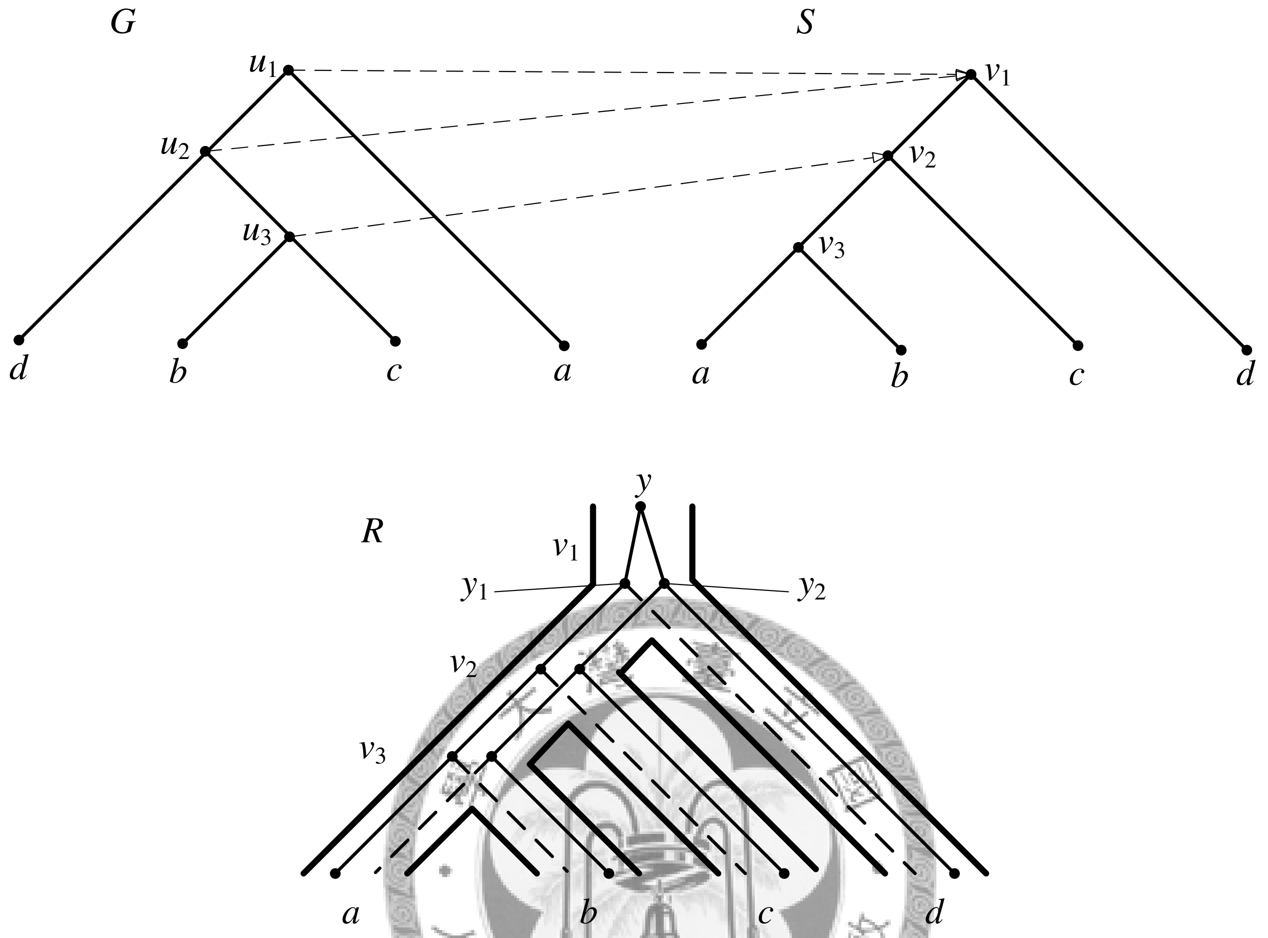

Figure 1.3: An illustration for describing how the GD model explains the inconsistency between a gene tree G and a species tree S. R is the reconciled tree for G and S. For simplicity, the labels of leaves of G are replaced with the corresponding leaf-mapping.

the reconciled tree for G and S. The LCA-mappings $\mathcal{M}_{G,S}(u_1)$, $\mathcal{M}_{G,S}(u_2)$, and $\mathcal{M}_{G,S}(u_3)$ are v_1, v_1, and v_2, respectively. In the species v_1 of R, the gene y duplicates into two copies y_1 and y_2, and both copies speciate according to the topology of the species tree S. The solid lines in R represent the embedding of G into R, while the dashed lines in R represent the losses of G. Thus, the inconsistency between G and S can be explained by postulating one gene duplication and four losses.

1.3 Problem Definition and Results

In this section, we define the problems discussed in this dissertation, and state our results for these problems. These results are summarized in Figure 1.4.

1. The Episode-Clustering (EC) Problem. Given a set of gene tree $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a species tree S, the EC problem is to find a minimum cover for the collection of intervals $\mathcal{I} = \bigcup_{g \in Dup(\mathcal{G},S)} \{I(g)\}$ under the partial order $\leq_S$. For the EC problem, Burleigh *et al.* [17] presented an exact algorithm rather than heuristic approaches used previously. The time complexity of the exact algorithm is $O((\sum_{i=1}^{k} m_i)^2 + p\sum_{i=1}^{k} m_i + n)$ where $m_i = |Le(G_i)|$ for all $1 \leq i \leq k$, $p = |E(int(\mathcal{I}))|$, and $n = |Le(S)|$. In this dissertation, we propose an optimal $O(\sum_{i=1}^{k} m_i + n)$-time algorithm.

2. The Tree Interval Cover (TIC) Problem. Given a tree T and a collection of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_\lambda\}$, the TIC problem is to find a minimum cover C for $\mathcal{I}$ under the partial order $\leq_T$. Burleigh *et al.* [17] showed that the EC problem is linear-time reducible to the TIC problem. Thus, Burleigh *et al.* solved the EC problem by proposing an $O(\lambda^2 + p\lambda + n)$-time algorithm for the TIC problem, where $p = |E(int(\mathcal{I}))|$ and $n = |Le(T)|$. In this dissertation, we give an optimal $O(\lambda + n)$-time algorithm for this problem.

3. The Minimum Episodes (ME) Problem. Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a species tree S, the ME problem is to assign duplications to nodes in S such that the total number of episodes is minimized, where each duplication g is associated with an interval $I(g)$ in S describing the locations in which g can be placed. The formal definition of episodes will be discussed in Chapter 3. For the ME problem, the problem had been open for a long time. Bansal and Eulenstein [7] were the first to solve it by an exact algorithm with the time complexity $O(\sum_{i=1}^{k} m_i n)$, where $m_i = |Le(G_i)|$ for all $1 \leq i \leq k$ and $n = |Le(S)|$. In this dissertation, we give an optimal $O(\sum_{i=1}^{k} m_i + n)$-time algorithm for the ME problem.

4. The DL-NNI Local Search Problem. We consider the heuristic for the Duplication-Loss problem based on the NNI operation and the corresponding local search problem, called the DL-NNI Local Search problem. Given a set of gene trees and a species tree S, the DL-NNI Local Search problem is to find a tree S^* with the minimum mutation cost among the neighborhood of S, where the neighborhood of S is the set of trees transformed from S by performing an NNI operation on any node of S. For a tree S and a node $x \in V(S)$, an NNI operation performed on x is to swap the subtree rooted at x and the subtree rooted at the sibling of the parent of x. Note that there are $\Theta(n)$ trees in the neighborhood of S based on the NNI operation, where n is the number of leaves in S. Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a species tee S, Zhang presented a linear-time algorithm for computing the mutation cost in [57]. Given $\mathcal{G}$ and S, the naïve algorithm for the DL-NNI Local Search problem computes the mutation costs for each tree in the neighborhood of S and takes total $O(\sum_{i=1}^{k} m_i n + n^2)$ time, where $m_i = |Le(G)|$ for all $1 \leq i \leq k$ and $n = |Le(S)|$. In this dissertation, we propose a linear-time algorithm for the DL-NNI Local Search problem. In addition, Bansal *et al.* [8] proposed a near-linear time algorithm for the NNI Local Search problem under the duplication cost (the D-NNI Local Search problem), and the problem is a special case of the DL-NNI Local Search problem if we set the number of losses zero. Therefore, our linear-time algorithm for the DL-NNI Local Search problem also improves the result in [8].

1.4 Organization of the Dissertation

There are five chapters in this dissertation. In Chapter 2, we discuss the EC problem and propose an optimal linear-time algorithm for the problem. Chapter 3 describes the ME problem and provides an optimal linear-time algorithm for the problem. We study the heuristic for the Duplication-Loss problem and propose a linear-time algorithm for the DL-NNI Local

Problem	Previous results	Our results
The Episode-Clustering Problem	$O((\sum_{i=1}^{k} m_i)^2 + p\sum_{i=1}^{k} m_i + n)$ [17]	$O(\sum_{i=1}^{k} m_i + n)$
The Tree Interval Cover Problem	$O(\lambda^2 + p\lambda + n)$ [17]	$O(\lambda + n)$
The Minimum Episodes Problem	$O(\sum_{i=1}^{k} m_i n)$ [7]	$O(\sum_{i=1}^{k} m_i + n)$
The DL-NNI Local Search Problem	$O(\sum_{i=1}^{k} m_i n + n^2)$[†]	$O(\sum_{i=1}^{k} m_i + n)$

Figure 1.4: Summary of results in this dissertation.

Search problem in Chapter 4. Finally, concluding remarks appear in Chapter 5.

[†]Only naïve algorithms were known for this problem. It should be noted that Bansal *et al.* proposed a near-linear time algorithm for the D-NNI Local Search problem [8]. In the near-linear time algorithm, the first instance of the D-NNI Local Search problem can be solved in $O(\sum_{i=1}^{k} m_i n + n^2)$ time, while the following instances can be solved in $O(\sum_{i=1}^{k} m_i + n)$ time.

Chapter 2

The Episode-Clustering Problem

2.1 A Linear-Time Algorithm for the Episode-Clustering Problem

In this chapter, we study the Episode-Clustering (EC) problem. Burleigh *et al.* [17] introduced the Tree Interval Cover (TIC) problem and showed that the EC problem is a special case of the TIC problem. In the following, we give the definition of the EC problem and propose a linear-time algorithm for the TIC problem.

2.1.1 A Linear-Time Algorithm for the Tree Interval Cover Problem

Given a collection of gene trees $\mathcal{G}$ and a species tree S, the EC problem is to find a minimum cover for the collection of intervals $\mathcal{I} = \bigcup_{g \in Dup(\mathcal{G},S)} \{I(g)\}$ under the partial order $\leq_S$. Now we turn to introduce the TIC problem. Given a tree T and a collection of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_\lambda\}$ where $I_i = [a_i, b_i]$ and $a_i, b_i \in V(T)$ for $i = 1, 2, \ldots, \lambda$, the TIC problem is to find a minimum cover C for $\mathcal{I}$ under the partial order $\leq_T$. Note that given $\mathcal{G}$ and S, the collection of intervals $\mathcal{I}$ can be computed in linear time using the efficient algorithm for finding the least common ancestor [11, 57]. As a result, it is not hard to see that the EC problem is a special case of the TIC problem [17]. Next, we state the linear-time algorithm for the TIC problem.

The algorithm proceeds as follows. First, we traverse the tree T from $Ro(T)$ using the breadth-first search to compute $d(v)$ for each node $v \in V(T)$, i.e., the distance between v to

$Ro(T)$. With the value of $d(v)$, we can derive the length, $l(I_i)$, of each interval $I_i = [a_i, b_i]$ in $\mathcal{I}$, by calculating the value of $d(a_i) - d(b_i)$. For each node v in T, we maintain a value $min\_len(v)$ defined as follows. Let $\mathcal{I}(v)$ be the set of intervals that pass through the node v. The $min\_len(v)$ is used to keep the minimum length of intervals from v to all *ending terminals* among the intervals in $\mathcal{I}(v)$. Initially, we set the value of $min\_len(v)$ to be the minimum among the lengths of the intervals using v as the *starting terminal.* If such value does not exist for the node v, the value of $min\_len(v)$ is set to infinity.

Then we traverse the tree T in a bottom-up fashion. When we visit a node v in T, we have the following two cases:

1. If $min\_len(v) = 0$, there exists at least one interval whose *ending terminal* is v. Hence, we must add the node v into the cover C.

2. If $min\_len(v) \neq 0$, there are no intervals using v as an *ending terminal.* We just upload the value $min\_len(v) - 1$ to $Pa(v)$ and compare the value $min\_len(v) - 1$ with $min\_len(Pa(v))$. Then we take the smaller value as the value of $min\_len(Pa(v))$.

We call the above method Algorithm TIC, and an example of executing Algorithm TIC is given in Figure 2.2. In G_1, the boxed value of each internal node u_i denotes the LCA-mapping, and the interval $I(u_i)$ is marked on the left side of node u_i if u_i is a gene duplication, where $1 \leq i \leq 5$. The same usage applies to G_2. In S, the gray-colored value of each internal node s_j denotes the initial value of $min\_len(s_j)$ computed by Algorithm TIC, where $1 \leq j \leq 7$. When Algorithm TIC traverses S in a bottom-up fashion, s_7 is the first internal node to be visited, and we do nothing because $min\_len(s_7) = \infty$. When s_6 is visited, the value of $min\_len(s_5)$ does not change since $min\_len(s_6) - 1 > min\_len(s_5)$. When s_5 is visited, s_5 is added to the cover C since $min\_len(s_5) = 0$. When s_4 is visited, we also do nothing because $min\_len(s_4) = \infty$. When s_3 is visited, s_3 is added to the cover C since $min\_len(s_3) = 0$. When s_1 and s_2 are visited, we do nothing because their min_len values are infinity. Finally, the returned cover $C = \{s_3, s_5\}$.

```
Algorithm TIC($T, \mathcal{I}$)
Input: A rooted tree $T$, where $|V(T)| = N$; a collection of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_\lambda\}$
        under the partial order $\leq_T$, where $I_i = [a_i, b_i]$ for $i = 1, \ldots, \lambda$.
Output: Return a minimum cover $C$ of $\mathcal{I}$.
 1 Perform the breadth-first search to calculate $d(v)$, the distance between $v$ and $Ro(T)$
   for each $v \in V(T)$.
 2 for each node $v \in V(T)$ do
 3     $min\_len(v) \leftarrow \infty$.
 4 for $i = 1$ to $\lambda$ do
 5     $l(I_i) \leftarrow d(a_i) - d(b_i)$.
 6     if $l(I_i) < min\_len(a_i)$ then
 7         $min\_len(a_i) \leftarrow l(I_i)$.
 8 Apply the postorder traversal to $T$ and let $v_1, v_2, \ldots, v_N$ be the visiting order of nodes.
 9 for $i = 1$ to $N$ do
10     if $min\_len(v_i) = 0$ then
11         Insert $v$ into the cover $C$.
12     else if $min\_len(v) - 1 < min\_len(Pa(v))$ then
13         $min\_len(Pa(v)) \leftarrow min\_len(v) - 1$.
14 return $C$.
```

Figure 2.1: The algorithm for the Tree Interval Cover problem.

2.1.2 Correctness and Complexity

Now we show the correctness and time complexity of this algorithm.

Lemma 1: Let $C = \{c_1, c_2, \ldots, c_p\}$ be the cover found by Algorithm TIC. The cover C must be a feasible interval cover, i.e., a cover covering all intervals in $\mathcal{I}$.

Proof: For the purpose of contradiction, assume that $I' = [s_1, s_i]$ be an interval that is not covered by the cover C. Let the path of I' be $(s_1, s_2, \ldots, s_i)$. When initializing the value $min\_len(s_1)$, it holds that $min\_len(s_1) \leq i - 1$ by Algorithm TIC. Then Algorithm TIC traverses the tree T in a bottom-up fashion. When visiting a node s_j where $1 \leq j \leq i$, it is clear that $min\_len(s_j) \leq i - j$. Before we encounter the node $Pa(s_i)$, there must exist a node $s' \in I'$ such that $min\_len(s') = 0$, and the node s' is inserted into our cover C. Therefore, the cover I' is covered by the node s' and our assumption is a contradiction. Hence, the cover C found by Algorithm TIC is a feasible interval cover. $\square$

Theorem 1: Algorithm TIC solves the TREE INTERVAL COVER problem correctly.

Proof: Let $C = \{c_1, c_2, \ldots, c_p\}$ be the cover found by Algorithm TIC. By Lemma 1, C is a feasible interval cover, i.e., a cover covering all intervals in $\mathcal{I}$. Therefore, the rest is to show that the cardinality of C is equal to that of an optimal cover.

For $i = 1, 2, \ldots, p$, there exists an interval, I_{c_i}, having c_i as an *ending terminal* and setting $min\_len(c_i) = 0$. We claim that $I_{c_i} \cap I_{c_j} = \emptyset$ for all $i \neq j$ where $1 \leq i, j \leq p$. For the purpose of contradiction, assume that $I_{c_i} \cap I_{c_j} \neq \emptyset$ for some $i \neq j$, and c_i is an ancestor of c_j without loss of generality. Since Algorithm TIC traverses the tree T in a bottom-up fashion, the node c_j would be inserted into the cover C earlier than the node c_i. When inserting c_j into C, the value of $min\_len(c_j)$ is equal to zero, and $min\_len(c_j) - 1$ would not be uploaded to compare with $min\_len(Pa(c_j))$ by Algorithm TIC. In other words, any interval passing through the node c_j can not affect the value of $min\_len(c_i)$. Due to $I_{c_i} \cap I_{c_j} \neq \emptyset$, I_{c_i} also pass through

c_j. Therefore, it is impossible that the interval I_{c_i} is the interval such that $min\_len(c_i) = 0$ and our assumption is a contradiction. Hence, for all $i \neq j$ where $1 \leq i, j \leq p$, the claim that $I_{c_i} \cap I_{c_j} = \emptyset$ holds.

By the above claim, we obtain that an optimal cover requires at least p nodes to cover these non-overlapped intervals $I_{c_1}, I_{c_2}, \ldots, I_{c_p}$. Thus, the cardinality of an optimal cover is equal to that of the cover C found by Algorithm TIC. $\square$

Theorem 2: Given a tree T and a collection of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_\lambda\}$, Algorithm TIC finds the minimum cover of $\mathcal{I}$ in $O(N + \lambda)$ time, where $|V(T)| = N$.

Proof: Algorithm TIC takes $O(N)$ time to compute $d(v)$ for each node $v \in V(T)$ using the breadth-first search. For each node $v \in V(T)$, the initial value of $min\_len(v)$ can be computed in $O(\lambda)$ time, and then Algorithm TIC traverses the tree T in a bottom-up fashion in $O(N)$ time. Totally, the time complexity for Algorithm TIC is $O(N + \lambda)$. $\square$

In the following corollary, we conclude the time complexity of the EC problem. The corollary can be easily derived by Theorem 2 and we omit the proof here.

Corollary 1: Given a set of gene tree $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a comparable species tree S, the EC problem can be solved in $O(\sum_{i=1}^{k} m_i + n)$ time, where $m_i = |Le(G_i)|$ for $i = 1, 2, \ldots, k$, and $n = |Le(S)|$.

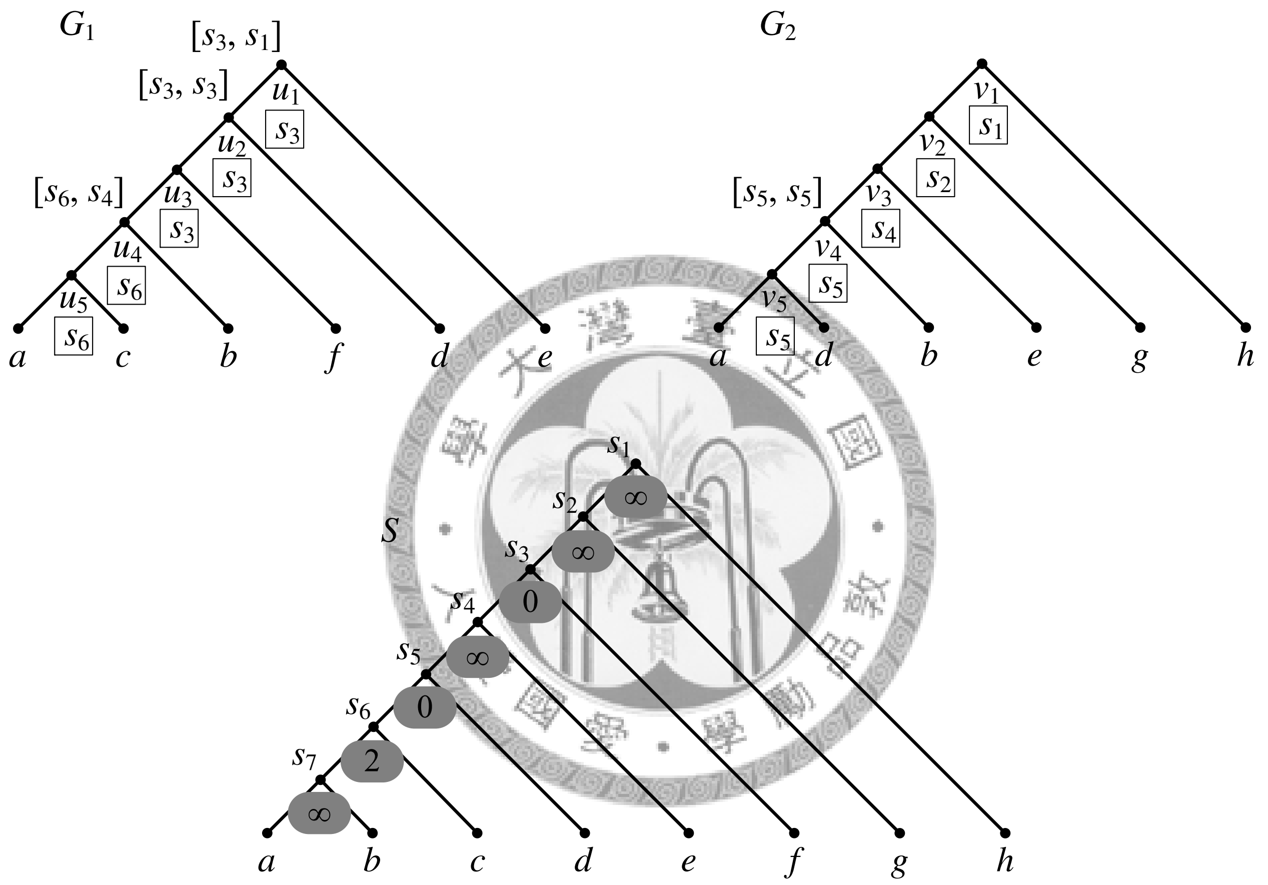

Figure 2.2: An example of executing Algorithm TIC for the EC problem with gene trees G_1, G_2 and a comparable species tree S. For simplicity, the labels of leaves of G are replaced with the corresponding leaf-mapping.

Chapter 3

The Minimum Episodes Problem

To study the MINIMUM EPISODES (ME) problem, we first introduce some definitions and notation proposed by Bansal and Eulenstein [7].

Given a gene tree G and a species tree S, let $\mathcal{F}_{G,S} : V(G) \rightarrow V(S)$. $\mathcal{F}_{G,S}$ is *valid* if for each node $g \in V(G)$, its mapping satisfies the following:

1. If $g \in Dup(G, S)$, g is mapped to any node in the interval $I(g)$.
2. Otherwise, $\mathcal{F}_{G,S}(g)$ is the same as $\mathcal{M}_{G,S}(g)$.

Let the unions of the mappings $\mathcal{F} = \bigcup_{G \in \mathcal{G}} \mathcal{F}_{G,S}$ and $\mathcal{F}_{\mathcal{M}} = \bigcup_{G \in \mathcal{G}} \mathcal{M}_{G,S}$ for a set of gene trees $\mathcal{G}$. $\mathcal{F}$ is *valid* if $\mathcal{F}_{G,S}$ is valid for each gene tree $G \in \mathcal{G}$.

Given a set of gene trees $\mathcal{G}$, a species tree S, and a valid mapping $\mathcal{F}$, let $\mathcal{F}^{-1}(s)$ denote the node set $\{g : \mathcal{F}(g) = s\}$ and $H(\mathcal{F}, s)$ denote the subgraph of $\mathcal{G}$ induced by the node set $\mathcal{F}^{-1}(s)$, where $s \in V(S)$. Note that $H(\mathcal{F}, s)$ must be a forest.

Definition 6: Given a set of gene trees $\mathcal{G}$, a species tree S, and a valid mapping $\mathcal{F}$, $\Delta(\mathcal{F}, s) = \max\{h(T) : T \text{ is a tree in } H(\mathcal{F}, s)\}$, i.e., the number of episodes at s caused by $\mathcal{F}$, where $s \in V(S)$. Also let $\Delta(\mathcal{F}) = \sum_{s \in V(S)} \Delta(\mathcal{F}, s)$.

Let T be a tree in $H(\mathcal{F}, s)$ such that $h(T) = \Delta(\mathcal{F}, s)$, where $s \in V(S)$. A node $g \in \mathcal{F}^{-1}(s)$ is a *leading node* if and only if g is the root of T. A node $g \in \mathcal{F}^{-1}(s)$ is *free* if and only if $Pa(s)$ is in the interval $I(g)$, where s is not the root of S.

The ME problem is, given a set of gene trees $\mathcal{G} = \{G_1, G_2, , \ldots, G_k\}$ and a species tree S, to find a valid mapping $\mathcal{F}_{opt} : \bigcup_{G \in \mathcal{G}} V(G) \to V(S)$ such that $\Delta(\mathcal{F}_{opt}) = \min\{\Delta(\mathcal{F}) : \mathcal{F}$ is any valid mapping$\}$.

3.1 Algorithm ME by Bansal and Eulenstein

Algorithm ME [7] first computes the mapping $\mathcal{F}_{\mathcal{M}}$ from $\mathcal{G}$ to S and all intervals $I(g)$ for each node $g \in Dup(\mathcal{G}, S)$. Let $\mathcal{F} : \bigcup_{G \in \mathcal{G}} V(G) \to V(S)$ record the mapping in each step. $\mathcal{F}$ is initialized with $\mathcal{F}_{\mathcal{M}}$ and is modified step by step as follows. S is traversed in postorder, and each visited node $s \in V(S)$ is checked whether $\mathcal{F}^{-1}(s) \neq \emptyset$ and all leading nodes in $\mathcal{F}^{-1}(s)$ are free. If both conditions hold, $\mathcal{F}$ is updated by changing the mappings of all leading nodes in $\mathcal{F}^{-1}(s)$ from s to $Pa(s)$. When the postorder traversal is terminated, $\Delta(\mathcal{F})$ is minimum, i.e., $\mathcal{F}$ is an optimal valid mapping.

Bansal and Eulenstein [7] gave an analysis of Algorithm ME as follows. Let $n = |Le(S)|$ and $m_i = |Le(G_i)|$ for all $1 \leq i \leq k$. The mapping $\mathcal{F}_{\mathcal{M}}$ is computed in $O(\sum_{i=1}^{k} m_i n)$ time. All intervals of the nodes in $Dup(\mathcal{G}, S)$ are calculated in $O(\sum_{i=1}^{k} m_i)$ time. For each node $s \in V(S)$, it takes $O(\sum_{i=1}^{k} m_i)$ time for each step of finding all leading nodes in $\mathcal{F}^{-1}(s)$, checking if these leading nodes are free, and updating the mapping $\mathcal{F}$. Since there are $O(n)$ nodes in the species tree, each of the above three steps takes $O(\sum_{i=1}^{k} m_i n)$ time, we have the following theorem.

Theorem 3: [7] Given a set of gene trees $\mathcal{G}$ and a species tree S, Algorithm ME computes an optimal valid mapping from the gene trees to the species tree in $O(\sum_{i=1}^{k} m_i n)$ time.

3.2 A Linear-Time Algorithm for the Minimum Episodes Problem

The time complexity of Algorithm ME is dominated by four steps: (1) computing the LCA-mapping, (2) finding all leading nodes, (3) checking if these leading nodes are free, and (4) updating the mapping. We present a linear-time algorithm for the ME problem by separately improving these steps in the following.

3.2.1 Computing the LCA-Mapping

Given a gene tree and a species tree, Zhang [57] proposed a linear-time algorithm for computing the LCA-mapping. We conclude the result in the following.

Theorem 4: [57] Given a gene tree G and a species tree S, computing the LCA-mapping from G to S takes $O(m+n)$ time, where $|Le(G)| = m$ and $|Le(S)| = n$.

By Theorem 7, the following corollary can be easily derived and we omit the proof here.

Corollary 2: Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$ and a species tree S, computing the LCA-mapping from $\mathcal{G}$ to S takes $O(\sum_{i=1}^{k} m_i + n)$ time, where $|Le(S)| = n$ and $|Le(G_i)| = m_i$ for all $1 \leq i \leq k$.

3.2.2 Finding All Leading Nodes

We present an efficient approach, named Algorithm LeadingNode, to finding all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ for each node $s \in V(S)$ in the following. For each node $s \in V(S)$, we maintain a value $\delta(s)$ and a linked list $lead[s]$ with two pointers $head[lead[s]]$ and $tail[lead[s]]$, which point to the head and tail of $lead[s]$, respectively. We use $\delta(s)$, initially zero, to keep $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ and use $lead[s]$ to store the leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$.

We traverse each gene tree $G \in \mathcal{G}$ by performing the breadth-first search from $Ro(G)$. For each node $g \in V(G)$, we keep a value $r(g)$ to store $Ro(T)$, where T is a tree in $H(\mathcal{F}_{\mathcal{M}}, \mathcal{F}_{\mathcal{M}}(g))$ with $g \in V(T)$. When g is visited, $r(g)$ is determined according the following three rules:

1. If $g = Ro(G)$, then $r(g) = g$ since g must be $Ro(T)$.

2. If $\mathcal{F}_{\mathcal{M}}(g) \neq \mathcal{F}_{\mathcal{M}}(Pa(g))$, it follows that $Pa(g) \notin V(T)$. Thus, $r(g) = g$.

3. Otherwise, $r(g) = r(Pa(g))$ since $Pa(g)$ is also a node in T.

For each node $g \in V(G)$, we also maintain a boolean value $flag(g)$, initially zero, to indicate whether g is a leading node. When visiting g, we let $s = \mathcal{F}_{\mathcal{M}}(g)$ and consider the following two cases:

1. If the distance between g and $r(g)$ plus one is greater than $\delta(s)$, then the elements in $lead[s]$ are not the leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and $r(g)$ is a new leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$. We delete each node $g' \in lead[s]$ and set $flag(r(g')) = 0$. We also insert $r(g)$ into $lead[s]$, set $flag(r(g)) = 1$, and update $\delta(s)$ to the distance between g and $r(g)$ plus one.

2. If the distance between g and $r(g)$ plus one is equal to $\delta(s)$ and $flag(r(g)) = 0$, then $r(g)$ is also a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$. We insert $r(g)$ into $lead[s]$ and set $flag(r(g)) = 1$.

After processing all gene trees in $\mathcal{G}$, we find all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and compute $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ for each node $s \in V(S)$. Algorithm LeadingNode is given in Figure 3.1. The correctness and time analysis of Algorithm LeadingNode are shown as follows.

Lemma 2: Given a set of gene trees $\mathcal{G}$, a species tree S, and a valid mapping $\mathcal{F}_{\mathcal{M}}$, Algorithm LeadingNode finds all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and computes $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ for each node $s \in V(S)$.

Proof: We need to show that (1) the elements in $lead[s]$ are all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$, and (2) $\Delta(\mathcal{F}_{\mathcal{M}}, s) = \delta(s)$ for each node $s \in V(S)$ after Algorithm LeadingNode terminates.

Part (1): Assume that g is a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and $g \notin lead[s]$. Let $g = Ro(T)$ for some tree T in $H(\mathcal{F}_{\mathcal{M}}, s)$. Since $g \notin lead[s]$, there exists another tree T' in $H(\mathcal{F}_{\mathcal{M}}, s)$ with $g' = Ro(T')$ such that $h(T') = \delta(s) > h(T)$ by lines 14 and 21 of Algorithm LeadingNode. Thus, it follows that $\Delta(\mathcal{F}_{\mathcal{M}}, s) \geq h(T') > h(T)$ and g is not a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$, which is a contradiction.

On the other hand, assume that $g \in lead[s]$ and g is not a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$. Let $g = Ro(T)$ for some tree T in $H(\mathcal{F}_{\mathcal{M}}, s)$. Since g is not a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$, there must exist a tree T' in $H(\mathcal{F}_{\mathcal{M}}, s)$ with $g' = Ro(T')$ such that $h(T') > h(T)$. By lines 14 and 21 of Algorithm LeadingNode, g is not in $lead[s]$, which is a contradiction. Therefore, g is a leading node in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ if and only if $g \in lead[s]$ for each node $s \in V(S)$.

Part (2): Let T be a tree in $H(\mathcal{F}_{\mathcal{M}}, s)$ with $h(T) = \delta(s)$, and $g = Ro(T)$. For the purpose of contradiction, assume that there exists a tree T' in $H(\mathcal{F}_{\mathcal{M}}, s)$ with $h(T') = \Delta(\mathcal{F}_{\mathcal{M}}, s)$ such

```
Algorithm LeadingNode($\mathcal{G}, S, \mathcal{F}_{\mathcal{M}}$)
Input: A set of gene trees $\mathcal{G}$, a species tree $S$, and a valid mapping $\mathcal{F}_{\mathcal{M}}$.
Output: Return all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ for each node $s \in V(S)$.
 1 for each node $s \in V(S)$ do
 2     $\delta(s) = 0$.
 3     $head[lead[s]] \leftarrow$ NIL.
 4 for each gene tree $G \in \mathcal{G}$ do
 5     for each node $g \in V(G)$ visited in the breadth-first search from $Ro(G)$ do
 6         if $g = Ro(G)$ then
 7             $r(g) \leftarrow g$.
 8         else if $\mathcal{F}_{\mathcal{M}}(g) \neq \mathcal{F}_{\mathcal{M}}(Pa(g))$ then
 9             $r(g) \leftarrow g$.
10         else
11             $r(g) \leftarrow r(Pa(g))$.
12         $flag(g) \leftarrow 0$.
13         Calculate $d(g)$, the distance between $g$ and $Ro(G)$.
14         if $\delta(\mathcal{F}_{\mathcal{M}}(g)) < d(g) - d(r(g)) + 1$ then
15             for each element $g'$ in $lead[\mathcal{F}_{\mathcal{M}}(g)]$ do
16                 Delete $g'$ from $lead[\mathcal{F}_{\mathcal{M}}(g)]$.
17                 $flag(g') \leftarrow 0$.
18             Insert $r(g)$ into $lead[\mathcal{F}_{\mathcal{M}}(g)]$.
19             $flag(r(g)) \leftarrow 1$.
20             $\delta(\mathcal{F}_{\mathcal{M}}(g)) \leftarrow d(g) - d(r(g)) + 1$.
21         else if $\delta(\mathcal{F}_{\mathcal{M}}(g)) = d(g) - d(r(g)) + 1$ and $flag(g) = 0$ then
22             Insert $r(g)$ into $lead[\mathcal{F}_{\mathcal{M}}(g)]$.
23             $flag(r(g)) \leftarrow 1$.
24 return $lead[s]$ and $\delta(s)$ for each $s \in V(S)$.
```

Figure 3.1: The algorithm for finding the leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ and for computing $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ for each node $s \in V(S)$.

that $h(T') > h(T)$. Let $g' = Ro(T')$. According to line 16 of Algorithm LeadingNode, $\delta(s)$ must be set to $h(T')$, which contradicts the assumption. Thus, we have $h(T') \leq h(T) = \delta(s)$ and $\Delta(\mathcal{F}_{\mathcal{M}}, s) = \delta(s)$ for each node $s \in V(S)$. $\square$

Lemma 3: Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$, a species tree S, and a valid mapping $\mathcal{F}_{\mathcal{M}}$, the time complexity of Algorithm LeadingNode is $O(\sum_{i=1}^{k} m_i + n)$, where $|Le(S)| = n$ and $|Le(G_i)| = m_i$ for all $1 \leq i \leq k$.

Proof: Algorithm LeadingNode initializes $\delta(s)$ and $lead[s]$ for all nodes $s \in V(S)$ in $O(n)$ time. All gene trees in $\mathcal{G}$ are traversed, so there are in total $O(\sum_{i=1}^{k} m_i)$ visited nodes. The two for-loops in lines 4 and 5 contain $O(\sum_{i=1}^{k} m_i)$ iterations and each operation in the two for-loops takes $O(1)$ time except the for-loop in lines 15–17. The time complexity of the two for-loops in lines 4 and 5 is dominated by the for-loop in lines 15–17. According to lines 18 and 22 of Algorithm LeadingNode, however, a visited node $g \in V(G)$ brings at most one insertion. It implies that $O(\sum_{i=1}^{k} m_i)$ insertions are totally executed. That is, the total number of deletions in line 16 of Algorithm LeadingNode is $O(\sum_{i=1}^{k} m_i)$, and the for-loop in lines 15–17 takes totally $O(\sum_{i=1}^{k} m_i)$ time. Consequently, the time complexity of Algorithm LeadingNode is $O(\sum_{i=1}^{k} m_i + n)$. $\square$

3.2.3 Checking If All Leading Nodes Are Free

In the following, we check if all leading nodes in $\mathcal{F}^{-1}(s)$ are free, where $\mathcal{F}$ is an arbitrary valid mapping. For each gene tree $G \in \mathcal{G}$, the interval $I(g)$ and the length of $I(g)$ for each node $g \in Dup(G, S)$ are computed in linear time [57] by applying an efficient algorithm for the Least Common Ancestor problem [11]. For each node $s \in V(S)$, we maintain a value $min\_len(s)$, which is the minimum length of intervals from s to all *ending terminals* among all intervals passing through s. Initially, we set $min\_len(s)$ to be the minimum among the lengths of the intervals of all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ using s as the *starting terminal.* If

```
Algorithm LinearME($\mathcal{G}, S$)
Input: A set of gene trees $\mathcal{G}$ and a species tree $S$.
Output: Return an optimal mapping $\mathcal{F} : \bigcup_{G \in \mathcal{G}} V(G) \rightarrow V(S)$ such that $\Delta(\mathcal{F})$
    $= \min\{\Delta(\widehat{\mathcal{F}}) : \widehat{\mathcal{F}}$ is any valid mapping$\}$.
 1 Compute the LCA-mapping $\mathcal{F}_{\mathcal{M}}$ from $\mathcal{G}$ to $S$.
 2 Find all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ for each node $s \in V(S)$.
 3 for each gene tree $G \in \mathcal{G}$ do
 4     Compute the interval $I(g) = [a_g, b_g]$ for each $g \in Dup(G, S)$ using the algorithm in [11].
 5 Traverse $S$ by the breadth-first search from $Ro(S)$ to calculate $d(s)$, the distance
   between $s$ and $Ro(S)$ for each node $s \in V(S)$.
 6 for each node $s \in V(S)$ do
 7     $min\_len(s) \leftarrow \infty$.
 8 for each node $s \in V(S)$ do
 9     for each node $g$ in $lead[s]$ do
10         if $g \in Dup(\mathcal{G}, S)$ then
11             $l(I(g)) \leftarrow d(a_g) - d(b_g)$.
12         else
13             $l(I(g)) \leftarrow 0$.
14         if $l(I(g)) < min\_len(\mathcal{F}_{\mathcal{M}}(g))$ then
15             $min\_len(\mathcal{F}_{\mathcal{M}}(g)) \leftarrow l(I(g))$.
16 for each node $s \in V(S)$ visited in postorder do
17     if $head[lead[s]] \neq$ NIL then
18         if $min\_len(s) > 0$ then
19             if $\delta(Pa(s)) \leq 1$ then
20                 $tail[lead[Pa(s)]] \leftarrow head[lead[s]]$.
21                 $min\_len(Pa(s)) \leftarrow \min\{min\_len(Pa(s)), min\_len(s) - 1\}$.
22             else
23                 $tail[leadfree[Pa(s)]] \leftarrow head[lead[s]]$.
24             $head[lead[s]] \leftarrow$ NIL.
25 Construct a mapping $\mathcal{F} : \bigcup_{G \in \mathcal{G}} V(G) \rightarrow V(S)$ as follows:
       For each node $g \in \bigcup_{G \in \mathcal{G}} V(G)$,
           $\mathcal{F}(g) = \begin{cases} s, & \text{if } g \in lead[s] \text{ or } g \in leadfree[s], \\ \mathcal{F}_{\mathcal{M}}(g), & \text{otherwise.} \end{cases}$
26 return $\mathcal{F}$.
```

Figure 3.2: The algorithm for the Minimum Episodes problem.

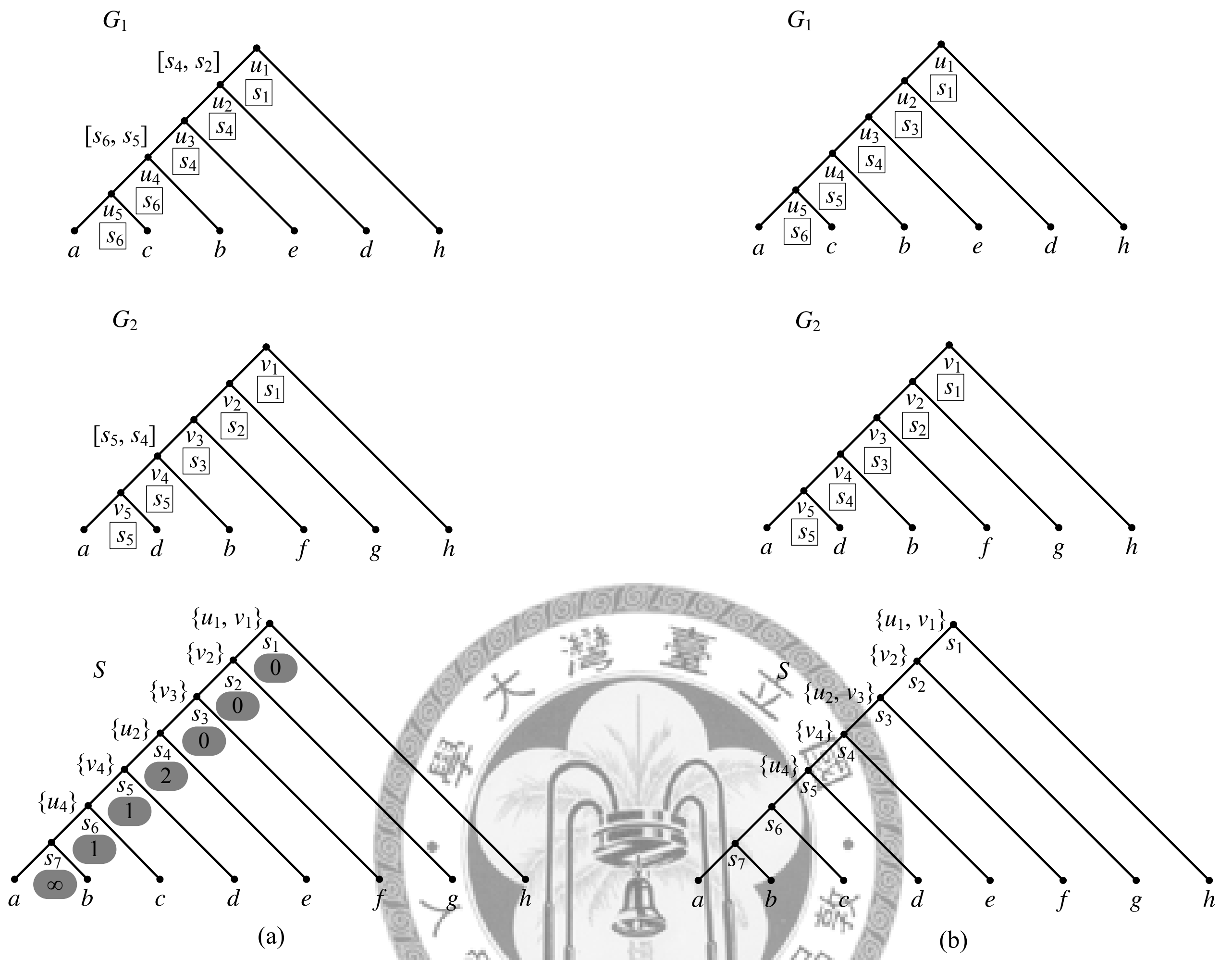

Figure 3.3: An example of executing Algorithm LinearME for the ME problem with gene trees G_1, G_2 and a comparable species tree S. For simplicity, the labels of leaves of G_1 and G_2 are replaced with the corresponding leaf-mapping. (a) Two gene trees G_1, G_2 and a comparable species tree S. (b) The valid mapping $\mathcal{F}$ returned by Algorithm LinearME.

$min\_len(s)$ is greater than zero, all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$ are free. According to Algorithm ME [7], we change the mapping in this condition and update $min\_len(Pa(s))$ to the value of $\min\{min\_len(Pa(s)), min\_len(s) - 1\}$.

3.2.4 Updating the Mapping from the Gene Trees to the Species Tree

We now show how to efficiently update the mapping from $\mathcal{G}$ to S and, together with the above three improved approaches, present a linear-time algorithm for the ME problem. First, we traverse the species tree S in postorder. When visiting a node $s \in V(S)$, we let $\mathcal{F}_1$ denote the mapping just before visiting s and check whether $\mathcal{F}_1^{-1}(s) \neq \emptyset$, i.e., $head[lead[s]] \neq$ NIL, and whether all leading nodes $\mathcal{F}_1^{-1}(s)$ are free, i.e., $min\_len(s) > 0$. If both conditions hold, the mappings of all nodes in $lead[s]$ from s to $Pa(s)$ are changed by moving all nodes in $lead[s]$ to $lead[Pa(s)]$. Let $\mathcal{F}_2$ be the mapping just after the modification of the mappings of all nodes in $lead[s]$. In $H(\mathcal{F}_2, Pa(s))$, the nodes originally in $lead[s]$ become nodes of degree zero, i.e., trivial trees. In other words, in $H(\mathcal{F}_2, s)$, all trees induced by the nodes originally in $lead[s]$ are trivial trees. This implies that the modified mapping does not increase the total number of episodes. This observation is shown in Lemma 4.

Lemma 4: Let $s \in V(S)$ be the node being visited, and assume that $\mathcal{F}_{\mathcal{M}}^{-1}(s) \neq \emptyset$ and all nodes in $lead[s]$ are free leading nodes. Let $\mathcal{F}_1$ be the mapping just before visiting s, and $\mathcal{F}_2$ be the mapping just after the modification of the mappings of all nodes in $lead[s]$ from s to $Pa(s)$. For all nodes $g \in lead[s]$, g is a node of degree zero, i.e., a trivial tree, in $H(\mathcal{F}_2, Pa(s))$.

Proof: We know that $\mathcal{F}_1(g) = s$ and $\mathcal{F}_2(g) = Pa(s)$. Since g is a free leading node under the mapping $\mathcal{F}_1$, it follows that $\mathcal{F}_1(g) \neq \mathcal{F}_1(Pa(g)) = \mathcal{F}_2(Pa(g)) = \mathcal{M}_{G,S}(Pa(g))$, and that both s and $Pa(s)$ are in $I(g)$. By Definition 3, $\mathcal{M}_{G,S}(Pa(g))$ does not belong to $I(g)$. Thus, $\mathcal{F}_2(Pa(g))$ is not equal to $\mathcal{F}_2(g)$ and $Pa(g)$ is not the parent of g under the mapping $\mathcal{F}_2$. Let a and b be the left and right children of g in G, respectively. Since $\mathcal{M}_{G,S}(a) \leq_S \mathcal{M}_{G,S}(g)$ and $\mathcal{M}_{G,S}(b) \leq_S \mathcal{M}_{G,S}(g)$, g is not the parent of a and b under the mapping $\mathcal{F}_2$. Therefore, g is a trivial tree in $H(\mathcal{F}_2, Pa(s))$. □

According to Lemma 4, we only need to compare the maximum height among the trees in $H(\mathcal{F}_2, Pa(s))$ with the height of these trivial trees induced by all nodes in $lead[s]$ as follows. If $\delta(Pa(s)) \leq 1$, then all nodes in $lead[s]$ are also the leading nodes of $Pa(s)$. Therefore, $tail[lead[Pa(s)]]$ is changed by pointing to $head[lead[s]]$, and $min\_len(Pa(s))$ is updated to the value of $\min\{min\_len(Pa(s)), min\_len(s) - 1\}$. Otherwise, we use a linked list $leadfree[s]$ to collect those nodes whose mappings are changed from s to $Pa(s)$ but not the leading nodes in

$\mathcal{F}_2^{-1}(Pa(s))$. The procedure is repeated until all nodes in S are visited. Finally, we construct a new mapping $\mathcal{F} : \bigcup_{G\in\mathcal{G}} V(G) \to V(S)$ in a way that for each node $g \in \bigcup_{G\in\mathcal{G}} V(G)$, $\mathcal{F}(g) = s$ if $g \in lead[s]$ or $g \in leadfree[s]$, and $\mathcal{F}(g) = \mathcal{F}_{\mathcal{M}}(g)$ otherwise. The construction is performed as follows. First, we traverse S in postorder and set $\mathcal{F}(g) = s$ for all nodes g in $lead[s]$ and $leadfree[s]$. Next, all gene trees $G \in \mathcal{G}$ are also traversed in postorder. For each node $g \in V(G)$, if $flag(g) = 0$ then we set $\mathcal{F}(g) = \mathcal{F}_{\mathcal{M}}(g)$ since its mapping has never been changed. $\mathcal{F}$ is the final solution to the ME problem.

Algorithm LinearME for the ME problem is shown in Figure 3.2, and an example of executing Algorithm LinearME is given in Figure 3.3. Let G_1, G_2 be two gene trees and S be a comparable species tree. For Figure 3.3(a), in G_1, the boxed value of each internal node u_i denotes the LCA-mapping $\mathcal{F}_{\mathcal{M}}(u_i)$, and the interval $I(u_i)$ is marked on the left side of node u_i if u_i is a gene duplication, where $1 \le i \le 5$. The same usage applies to G_2. In S, the gray-colored value of each internal node s_j denotes the value of $min\_len(s_j)$ computed by lines 6–15 of Algorithm LinearME, and the linked list of leading nodes $lead[s_j]$ is shown on the left side of s_j, where $1 \le j \le 7$. $\Delta(\mathcal{F}_{\mathcal{M}}) = 17$. For Figure 3.3(b), in G_1, the boxed value of each internal node u_i denotes the returned mapping $\mathcal{F}(u_i)$, and the same usage applies to G_2. In S, the computed linked list $lead[s_j]$ or $leadfree[s_j]$ is shown on the left side of s_j, where $1 \le j \le 7$. At the execution of the for-loop in line 16, s_7 is the first internal node to be visited, and this iteration is terminated because $lead[s_7] = \emptyset$. When s_6 is visited, line 23 is executed since $min\_len(s_6) > 0$ and $\delta(s_5) > 1$. That is, $leadfree[s_5] = \{u_4\}$ and $lead[s_6] = \emptyset$. The procedure of visiting s_5 is similar to s_6. Thus, $leadfree[s_4] = \{v_4\}$ and $lead[s_5] = \emptyset$. When s_4 is visited, lines 20 and 21 are executed since $min\_len(s_4) > 0$ and $\delta(s_3) \le 1$. In other words, $lead[s_3] = \{v_3, u_2\}$, $min\_len(s_3) = \min\{0, 2-1\} = 0$, and $lead[s_4] = \emptyset$. After s_1, s_2, and s_3 are visited, the iterations are terminated because the min_len values of the three nodes are all zero. Finally, a valid mapping $\mathcal{F}$ is computed and $\Delta(\mathcal{F}) = 14$.

3.3 Correctness and Complexity

Since Algorithm LinearME is based on Algorithm ME [7], the correctness of Algorithm LinearME follows the proof shown in [7]. We conclude the correctness of Algorithm LinearME in Theorem 5.

Theorem 5: Given a set of gene trees $\mathcal{G}$ and a species tree S, Algorithm LinearME computes a valid mapping $\mathcal{F} : \bigcup_{G \in \mathcal{G}} V(G) \to V(S)$ such that $\Delta(\mathcal{F}) = \min \{\Delta(\widehat{\mathcal{F}}) : \widehat{\mathcal{F}}$ is any valid mapping$\}$.

The time complexity of Algorithm LinearME is analyzed in Theorem 6, and we conclude that Algorithm LinearME is a linear-time algorithm.

Theorem 6: Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a species tree S, the time complexity of Algorithm LinearME is $O(\sum_{i=1}^{k} m_i + n)$, where $|Le(S)| = n$ and $|Le(G_i)| = m_i$ for all $1 \leq i \leq k$.

Proof: By Corollary 3 and Lemma 3, it takes $O(\sum_{i=1}^{k} m_i + n)$ time to compute the LCA mapping for all nodes in $\bigcup_{G \in \mathcal{G}} V(G)$, to find all leading nodes in $\mathcal{F}_{\mathcal{M}}^{-1}(s)$, and to compute the value of $\Delta(\mathcal{F}_{\mathcal{M}}, s)$ for all $s \in V(S)$. For all $g \in Dup(\mathcal{G}, S)$, all intervals $I(g)$ are computed in $O(\sum_{i=1}^{k} m_i + n)$ time [11, 57]. The executions of lines 5–15 run in $O(\sum_{i=1}^{k} m_i + n)$ time. The for-loop in line 16 executes $O(n)$ times of iterations and each iteration can be completed in $O(1)$ time. Constructing a new mapping $\mathcal{F}$ takes $O(\sum_{i=1}^{k} m_i + n)$ time. Therefore, Algorithm LinearME solves the ME problem in $O(\sum_{i=1}^{k} m_i + n)$ time. □

Chapter 4

The DL-NNI Local Search Problem

In this chapter, we study the heuristic for the Duplication-Loss problem based on the NNI operation and the corresponding local search problem, called the DL-NNI Local Search problem. We propose a linear-time algorithm for the DL-NNI Local Search problem.

4.1 Preliminaries

First we review the nearest neighbor interchange (NNI) operation [1, 15] and define the DL-NNI Local Search problem.

Definition 7: For a tree S, we define the *valid nodes* of S, $val(S)$, to be $V(S) \setminus (\{Ro(S)\} \cup Ch(Ro(S)))$.

Definition 8: Let S be a tree. For each node $x \in val(S)$, we denote by $NNI_S(x)$ the resulting tree by swapping the two subtrees S_x and S_y, where y is the sibling of $Pa(x)$. The tree $NNI_S(x)$ is the tree transformed from S by performing the NNI operation on the node x.

For the NNI operation, see Figure 4.1 for an illustration.

Given a set of gene trees $\mathcal{G}$ and a species tree S, the DL-NNI Local Search problem is to find a tree $S^* \in \{NNI_S(x) : x \in val(S)\}$ such that $Mut(\mathcal{G}, S^*) = \min_{T \in \{NNI_S(x):x \in val(S)\}} Mut(\mathcal{G}, T)$.

We present a linear-time algorithm for the DL-NNI Local Search in the following. Our algorithm contains two main steps: (1) initializing the LCA-mapping, the mutation cost, and

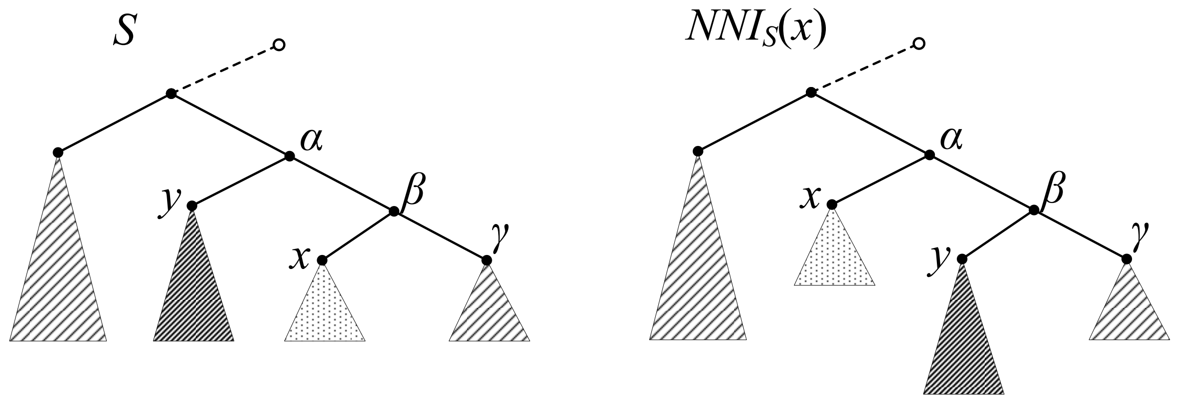

Figure 4.1: The trees S and $NNI_S(x)$, where $NNI_S(x)$ is obtained by swapping the subtrees S_x and S_y.

necessary values, and (2) computing the new gene duplications and losses separately after performing an NNI operation. It should be noted that we only focus on an individual gene tree $G \in \mathcal{G}$ in the following discussion, and it can be straightforward extended to all gene trees in $\mathcal{G}$. For simplicity, we also assume that $\mathcal{L}_{G,S}(Le(G)) = Le(S)$ in the following discussion. If $\mathcal{L}_{G,S}(Le(G)) \neq Le(S)$, we can set S to be $S|_{\mathcal{L}_{G,S}(Le(G))}$. After preprocessing S in linear time [11], the internal nodes of $S|_{\mathcal{L}_{G,S}(Le(G))}$ can be constructed in $O(|Le(G)|)$ time. By traversing S in postorder, we can construct $E(S|_{\mathcal{L}_{G,S}(Le(G))})$ in $O(|Le(G)|+|Le(S)|)$ time. The time complexity of our algorithm for the DL-NNI Local Search problem are not affected. The details of the algorithm for constructing $S|_{\mathcal{L}_{G,S}(Le(G))}$ for all $G \in \mathcal{G}$ will be presented in Section 4.5.

4.2 Initializing the LCA-mapping and the Mutation Cost

Given a gene tree and a species tree, Zhang [57] proposed a linear-time algorithm for computing the LCA-mapping and the mutation cost. We conclude the result in the following theorem.

Theorem 7: [57] Given a gene tree G and a species tree S, computing the LCA-mapping $\mathcal{M}_{G,S}$, the number of gene duplications $|Dup(G,S)|$, and the number of losses $Loss(G,S)$ takes $O(m+n)$ time, where $|Le(G)| = m$ and $|Le(S)| = n$.

By Theorem 7, the following corollary can be easily derived and we omit the proof here.

Corollary 3: Given a set of gene trees $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$ and a species tree S, computing the LCA-mapping $\mathcal{M}_{\mathcal{G},S}$, the number of gene duplications $|Dup(\mathcal{G}, S)|$, and the number of losses $Loss(\mathcal{G}, S)$ takes $O(\sum_{i=1}^{k} m_i + n)$ time, where $|Le(S)| = n$ and $|Le(G_i)| = m_i$ for all $1 \leq i \leq k$.

In addition to the LCA-mapping and the mutation cost, we also maintain some values $f(g, i)$ for each node $g \in V(G)$, where $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$. The definition of $f(g, i)$ is shown in the following.

$$f(g,i) = \begin{cases} 1 & \text{if } \mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_i) \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases}$$

When $f(g, i) = 1$, there exists at least one leaf z of G_g such that $\mathcal{M}_{G,S}(z)$ belongs to the leaf set of S_i. The information will be useful when we compute the new LCA-mapping after an NNI operation is performed. For a node $g \in V(G)$, now we discuss how to compute the value of $f(g, i)$ for each $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$. By the algorithm proposed by Bender and Farach-Colton [11], the least common ancestor of any two nodes in the species tree S can be answered in constant time after S is preprocessed in linear time. For each node $g \in V(G)$, we compute $f(g, i)$ by traversing the gene tree G in a bottom-up fashion. For each node $g \in V(G)$, we initially set the value $f(g, i)$ zero for all $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$. We do nothing to all nodes g where $Ch(Ch(\mathcal{M}_{G,S}(g))) = \emptyset$. When we visit a node g where $Ch(Ch(\mathcal{M}_{G,S}(g))) \neq \emptyset$, we check all nodes in $\mathcal{M}_{G,S}(Ch(g))$ to compute the value $f(g, i)$. Let z be a node in $Ch(g)$. There are the following four cases to be considered.

1. If $\mathcal{M}_{G,S}(z) = \mathcal{M}_{G,S}(g)$, then we set $f(g, i) = 1$ when $f(z, i) = 1$ for each $i \in Ch(Ch(\mathcal{M}_{G,S}(z)))$.

2. Let j be a node in $Ch(\mathcal{M}_{G,S}(g))$. If $\mathcal{M}_{G,S}(z) = j$, then we set the value $f(g, k) = 1$ for each $k \in Ch(j)$.

3. Let j be a node in $Ch(Ch(\mathcal{M}_{G,S}(g)))$. If $\mathcal{M}_{G,S}(z) = j$, then we set the value $f(g, j) = 1$.

4. If $\mathcal{M}_{G,S}(z) \notin \{\mathcal{M}_{G,S}(g)\} \cup Ch(\mathcal{M}_{G,S}(g)) \cup Ch(Ch(\mathcal{M}_{G,S}(g)))$, we check whether the least common ancestor of $\mathcal{M}_{G,S}(z)$ and j is equal to j, for each $j \in Ch(Ch(\mathcal{M}_{G,S}(g)))$.

If $lca(\mathcal{M}_{G,S}(z), j) = j$, it follows that $\mathcal{M}_{G,S}(Le(G_g)) \cap \{Le(S_j)\} \neq \emptyset$. Thus, we set the value $f(g, j) = 1$.

For the technical reasons, we also maintain the value $d_S(j)$, i.e., the distance between j and $Ro(S)$, for each node $j \in V(S)$. For each node $j \in V(S)$, the value $d_S(j)$ can be easily computed by traversing the species tree S using the breadth-first search. With the value $d_S(j)$, we can derive the distance between any two nodes u and v of S in $O(1)$ time by computing the value $|d_S(u) - d_S(v)|$.

Algorithm InitializeLocalSearch is shown in Figure 4.2. Now we analyze the time complexity of the algorithm. Note that we assume that $\mathcal{L}_{G,S}(Le(G)) = Le(S)$. Let $|Le(G)| = |Le(S)| = m$. In Algorithm InitializeLocalSearch, the execution of lines 1 and 2 runs in $O(m)$ time by Theorem 3 and [11]. Since visiting all nodes in G, the for-loop in line 3 executes $O(m)$ times totally. Since lines 4–20 take only $O(1)$ time, each iteration of the for-loop in line 3 can be computed in $O(1)$ time. Line 21 performs the breadth-first search on the species tree S and can be done in $O(m)$ time. Thus, the time complexity of Algorithm InitializeLocalSearch is $O(m)$.

Theorem 8: Given a gene tree G and a species tree S, let $|Le(G)| = |Le(S)| = m$. Algorithm InitializeLocalSearch computes the LCA-mapping, the mutation cost, and the value $f(g, i)$, where $g \in V(G)$, for each node $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$ in $O(m)$ time.

4.3 Gene Duplications in $NNI_S(x)$

In the following, we show how to compute all gene duplications $Dup(G, S')$, where $S' = NNI_S(x)$, after an NNI operation is performed on a node $x \in V(S)$. Before the discussion, we review the result proven by Bansal *et al.* [8].

Lemma 5: [8] Let g be a node of G and x be a node in $val(S)$. Assume that $Pa(x) = \beta$ and $Pa(\beta) = \alpha$. If $\mathcal{M}_{G,S}(g) \notin \{\alpha, \beta\}$, then $\mathcal{M}_{G,S'}(g)$ is the same as $\mathcal{M}_{G,S}(g)$, where $S' = NNI_S(x)$.

```
Algorithm InitializeLocalSearch($(G, S)$)
Input: A gene tree $G$ and a species tree $S$.
Output: Compute the LCA-mapping $\mathcal{M}_{G,S}$, the mutation cost, and the value $f(g, i)$ for
        each node $g \in V(G)$, where $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$.
 1 Compute the LCA-mapping $\mathcal{M}_{G,S}$, $|Dup(G, S)|$, and $Loss(G, S)$.
 2 Preprocess the species tree $S$ using the algorithm for computing the LCA in [11].
 3 for each node $g \in V(G)$ visited in the postorder do
 4     for each node $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$ do
 5         $f(g, i) \leftarrow 0$.
 6     for each node $z \in Ch(g)$ do
 7         if $\mathcal{M}_{G,S}(z) = \mathcal{M}_{G,S}(g)$ then
 8             for each node $j \in Ch(Ch(\mathcal{M}_{G,S}(z)))$ do
 9                 $f(g, j) \leftarrow f(z, j)$.
10         for each node $j \in Ch(\mathcal{M}_{G,S}(g))$ do
11             if $\mathcal{M}_{G,S}(z) = j$ then
12                 for each node $k \in Ch(j)$ do
13                     $f(g, k) \leftarrow 1$.
14         for each node $j \in Ch(Ch(\mathcal{M}_{G,S}(g)))$ do
15             if $\mathcal{M}_{G,S}(z) = j$ then
16                 $f(g, j) \leftarrow 1$.
17         if $\mathcal{M}_{G,S}(z) \notin \{\mathcal{M}_{G,S}(g)\} \cup Ch(\mathcal{M}_{G,S}(g)) \cup Ch(Ch(\mathcal{M}_{G,S}(g)))$ then
18             for each node $j \in Ch(Ch(\mathcal{M}_{G,S}(g)))$ do
19                 if $lca(\mathcal{M}_{G,S}(z), j) = j$ then
20                     $f(g, j) \leftarrow 1$.
21 Perform the breadth-first search to compute $d_S(j)$ for each node $j \in V(S)$.
```

Figure 4.2: The algorithm for computing the LCA-mapping from G to S, the mutation cost, and the values $f(g, i)$ for each node $g \in V(G)$, where $i \in Ch(Ch(\mathcal{M}_{G,S}(g)))$.

When we perform an NNI operation on a node x, except the nodes mapped to α or β, the LCA-mapping of other nodes in G remains the same by Lemma 5. Thus, we focus on the nodes in G mapped to α and β.

Lemma 6: Let g be a node of G and x be a node in $val(S)$. Assume that $Pa(x) = \beta$ and $Pa(\beta) = \alpha$. If $\mathcal{M}_{G,S}(g) = \beta$, then $\mathcal{M}_{G,S'}(g)$ is equal to α, where $S' = NNI_S(x)$.

Proof: Let the siblings of x and β be γ and y, respectively. Since $\mathcal{M}_{G,S}(g) = \beta$, we know that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) \neq \emptyset$, $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_\gamma) \neq \emptyset$, and $\mathcal{M}_{G,S}(Le(G_g)) \subseteq Le(S_x) \cup Le(S_\gamma)$. After performing an NNI operation on the node x, the two subtrees S_x and S_y are swapped. By Definition 1, it is clear that $\mathcal{M}_{G,S'}(g)$ is equal to α. □

Lemma 7: Let $x \in val(S)$, $Pa(x) = \beta$, and $Pa(\beta) = \alpha$. Assume that $g \in V(G)$ and $\mathcal{M}_{G,S}(g) = \alpha$. After performing an NNI operation on x, there are two cases to be considered. (Let $S' = NNI_S(x)$.)

1. If $f(g, x) = 1$, then $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S}(g) = \alpha$.
2. Otherwise, $\mathcal{M}_{G,S'}(g)$ is equal to β.

Proof: Let the siblings of x and β be γ and y, respectively. For Case 1, since $\mathcal{M}_{G,S}(g) = \alpha$, we know that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_y) \neq \emptyset$. Due to $f(g, x) = 1$, it follows that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) \neq \emptyset$. After we perform an NNI operation on x, the two subtrees S_x and S_y are swapped, and the least common ancestor of $Le(S_x)$ and $Le(S_y)$ is also α. Therefore, by Definition 1, $\mathcal{M}_{G,S'}(g)$ is equal to α.

For Case 2, since $\mathcal{M}_{G,S}(g) = \alpha$ and $f(g, x) \neq 1$, we know that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) = \emptyset$ and $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_\gamma) \neq \emptyset$. After performing an NNI operation on x, it follows that $\mathcal{M}_{G,S'}(Le(G_g)) \subseteq Le(S'_y) \cup Le(S'_\gamma)$. Therefore, $\mathcal{M}_{G,S'}(g)$ is equal to β. □

By the above two lemmas, we know the new LCA-mapping $\mathcal{M}_{G,S'}$ after an NNI operation is performed on a nodes x. It is clear that the difference between $Dup(G,S)$ and $Dup(G,S')$ results from those nodes of G, whose LCA-mapping is α or β. Now we turn to discuss the difference of gene duplications after performing an NNI operation.

Lemma 8: Let $x \in val(S)$, $Pa(x) = \beta$, and $Pa(\beta) = \alpha$. For a node $g \in V(G)$, assume that $\mathcal{M}_{G,S}(g) = \beta$. We have that $g \in Dup(G,S)$ if and only if $g \in Dup(G,S')$, where $S' = NNI_S(x)$.

Proof:

($\Rightarrow$) Let the siblings of x and β be γ and y, respectively. If $g \in Dup(G,S)$, there exists a child g' of g such that $\mathcal{M}_{G,S}(g) = \mathcal{M}_{G,S}(g') = \beta$. Thus, the two sets $\mathcal{M}_{G,S}(Le(G_{g'})) \cap Le(S_x)$ and $\mathcal{M}_{G,S}(Le(G_{g'})) \cap Le(S_\gamma)$ are not empty, and $\mathcal{M}_{G,S}(Le(G_{g'})) \subseteq Le(S_x) \cup Le(S_\gamma)$. After performing an NNI operation on x, the two subtrees S_x and S_y are swapped. Since $\mathcal{M}_{G,S}(Le(G_{g'})) \cap Le(S_x) \neq \emptyset$, $\mathcal{M}_{G,S}(Le(G_{g'})) \cap Le(S_\gamma) \neq \emptyset$, and $\mathcal{M}_{G,S}(Le(G_{g'})) \subseteq Le(S_x) \cup Le(S_\gamma)$, we know that $\mathcal{M}_{G,S'}(g')$ is α. By Lemma 6, $\mathcal{M}_{G,S'}(g)$ is equal to α. Due to $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = \alpha$, it follows that $g \in Dup(G,S')$.

($\Leftarrow$) By Lemma 6, we have that $\mathcal{M}_{G,S'}(g) = \alpha$. Since $g \in Dup(G,S')$, there exists a child g' of g such that $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = \alpha$. Due to $\mathcal{M}_{G,S}(g) = \beta$, we have that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_y) = \emptyset$ and $\mathcal{M}_{G,S}(Le(G_{g'})) \cap Le(S_y) = \emptyset$. Thus, we know that $\mathcal{M}_{G,S'}(Le(G_{g'})) \cap Le(S'_x) \neq \emptyset$, $\mathcal{M}_{G,S'}(Le(G_{g'})) \cap Le(S'_\gamma) \neq \emptyset$, and $\mathcal{M}_{G,S'}(Le(G_{g'})) \subseteq Le(S'_x) \cup Le(S'_\gamma)$. Since the NNI operation performed on the node x is to swap the two subtrees S_x and S_y, it is easy to verify that $\mathcal{M}_{G,S}(g') = \beta$. Due to $\mathcal{M}_{G,S}(g) = \mathcal{M}_{G,S}(g') = \beta$, it follows that $g \in Dup(G,S)$. $\square$

Lemma 9: Let $x \in val(S)$, $Pa(x) = \beta$, and $Pa(\beta) = \alpha$. For a node $g \in V(G)$, assume that $\mathcal{M}_{G,S}(g) = \alpha$ and $\mathcal{M}_{G,S'}(g) = \beta$. We have that $g \in Dup(G,S)$ if and only if $g \in Dup(G,S')$, where $S' = NNI_S(x)$.

Proof:

($\Rightarrow$) Since $\mathcal{M}_{G,S}(g) = \alpha$ and $\mathcal{M}_{G,S'}(g) = \beta$, it follows that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) = \emptyset$ by Lemma 7. Since $g \in Dup(G,S)$, there exists a child g' of g such that $\mathcal{M}_{G,S}(g) = \mathcal{M}_{G,S}(g') = \alpha$. The fact that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) = \emptyset$ and $\mathcal{M}_{G,S}(g') = \alpha$ implies $\mathcal{M}_{G,S'}(g') = \beta$ according to Lemma 7. Thus, we have that $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = \beta$ and obtain that $g \in Dup(G,S')$.

($\Leftarrow$) Since $\mathcal{M}_{G,S}(g) = \alpha$ and $\mathcal{M}_{G,S'}(g) = \beta$, it follows that $\mathcal{M}_{G,S}(Le(G_g)) \cap Le(S_x) = \emptyset$ by Lemma 7. Due to $g \in Dup(G,S')$, there exists a child g' of g such that $\mathcal{M}_{G,S'}(g) = \mathcal{M}_{G,S'}(g') = \beta$. The fact that $\mathcal{M}_{G,S'}(g') = \beta$ implies $\mathcal{M}_{G,S'}(Le(G_{g'})) \cap Le(S'_y) \neq \emptyset$, $\mathcal{M}_{G,S'}(Le(G_{g'})) \cap Le(S'_\gamma) \neq \emptyset$, and $\mathcal{M}_{G,S'}(Le(G_{g'})) \subseteq Le(S'_y) \cup Le(S'_\gamma)$. Thus, it is easy to verify that $\mathcal{M}_{G,S}(g') = \alpha$ before swapping the two subtrees S_x and S_y. We have $\mathcal{M}_{G,S}(g) = \mathcal{M}_{G,S}(g') = \alpha$ and obtain that $g \in Dup(G,S)$. □

According to Lemmas 8 and 9, the number of gene duplications is not affected by the nodes in $\mathcal{M}^{-1}_{G,S}(\beta)$ and $\{g : \mathcal{M}_{G,S}(g) = \alpha \text{ and } \mathcal{M}_{G,S'}(g) = \beta\}$. Therefore, the only nodes changing the number of gene duplications are in the set $GD\_Diff = \{g : \mathcal{M}_{G,S}(g) = \alpha \text{ and } \mathcal{M}_{G,S'}(g) = \alpha\}$. For a node $g \in GD\_Diff$, we must recompute the LCA-mapping of the children of g with respect to $NNI_S(x)$ to check whether g is a gene duplication or not. By Lemmas 6 and 7, the re-computation of the LCA-mapping of $Ch(g)$ can be done in $O(1)$ time. Therefore, after performing an NNI operation, we can determine if a node $g \in GD\_Diff$ is a gene duplication in $O(1)$ time.

Algorithm ComputeGD is shown in Figure 4.3. Now we analyze the time complexity of the algorithm. In Algorithm ComputeGD, the for-loop in line 3 executes $O(|\mathcal{M}^{-1}_{G,S}(\alpha)|)$ times totally. Since lines 4–15 take only $O(1)$ time, each iteration of the for-loop in line 3 can be computed in $O(1)$ time. Thus, the time complexity of Algorithm ComputeGD is $O(|\mathcal{M}^{-1}_{G,S}(\alpha)|)$.

Theorem 9: Let G be a gene tree, S be a species tree, and $x \in val(S)$. Given G, S, x, $|Dup(G,S)|$, and $\mathcal{M}_{G,S}$, Algorithm ComputeGD computes the number of gene duplications $|Dup(G, NNI_S(x))|$ in $O(|\mathcal{M}^{-1}_{G,S}(\alpha)|)$ time, where $\alpha = Pa(Pa(x))$.

```
Algorithm ComputeGD$(G, S, x, |Dup(G,S)|, \mathcal{M}_{G,S})$
Input: A gene tree $G$, a species tree $S$, a node $x \in V(S)$, the number of gene
        duplications $|Dup(G,S)|$, and the LCA-mapping $\mathcal{M}_{G,S}$.
Output: Return the number of gene duplications $|Dup(G, NNI_S(x))|$.
 1 Let $Pa(x) = \beta$, $Pa(Pa(x)) = \alpha$, and $S' = NNI_S(x)$.
 2 $|Dup(G,S')| \leftarrow |Dup(G,S)|$.
 3 for each node $g \in \mathcal{M}^{-1}_{G,S}(\alpha)$ do
 4     if $f(g,x) = 1$ then /* $\mathcal{M}_{\mathcal{G},S}(g) = \alpha$. */
 5         for each node $z \in Ch(g)$ do
 6             if $\mathcal{M}_{G,S}(z) = \beta$ then
 7                 $\mathcal{M}_{G,S'}(z) \leftarrow \alpha$.
 8             else if $\mathcal{M}_{G,S}(z) = \alpha$ and $f(z,x) = 1$ then
 9                 $\mathcal{M}_{G,S'}(z) \leftarrow \alpha$.
10             else $\mathcal{M}_{G,S'}(z) \leftarrow \beta$.
11         if $\mathcal{M}_{G,S}(Left(g)) = \alpha$ or $\mathcal{M}_{G,S}(Right(g)) = \alpha$ then /* $g \in Dup(\mathcal{G}, S)$. */
12             if $\mathcal{M}_{G,S'}(Left(g)) \neq \alpha$ and $\mathcal{M}_{G,S'}(Right(g)) \neq \alpha$ then
13                 $|Dup(G,S')| \leftarrow |Dup(G,S')| - 1$.
14         else if $\mathcal{M}_{G,S'}(Left(g)) = \alpha$ or $\mathcal{M}_{G,S'}(Right(g)) = \alpha$ then
15             $|Dup(G,S')| \leftarrow |Dup(G,S')| + 1$.
16 return $|Dup(G,S')|$.
```

Figure 4.3: The algorithm for computing the number of gene duplications $|Dup(G, NNI_S(x))|$.

4.4 Losses in $NNI_S(x)$

In this section, we show how to compute $Loss(G, S')$, where $S' = NNI_S(x)$, after performing an NNI operation on a node $x \in val(S)$. Unless specified otherwise, we assume that (1) $x \in val(S), S' = NNI_S(x)$, (2) $Pa(x) = \beta, Pa(\beta) = \alpha$, and (3) the siblings of x and β are γ and y, respectively, throughout this section. See the left part of Figure 4.1 for an illustration.

Let g be a node in G. We will consider the node g in the following five cases: (1) $\mathcal{M}_{G,S}(g) \in \{\alpha, \beta\}$ (Lemma 10), (2) $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in \{\alpha, \beta\}$ for some $g' \in Ch(g)$ (Lemma 11), (3) $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in V(S_y) \cup V(S_x)$ for some $g' \in Ch(g)$ (Lemma 12), (4) $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in (V(S)\backslash V(S_\alpha)) \cup V(S_\gamma)$ for each $g' \in Ch(g)$, and (5) $\mathcal{M}_{G,S}(g) \in V(S_y) \cup V(S_x) \cup V(S_\gamma)$.

Observation 1: Let g be a node in G. If $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in (V(S)\backslash V(S_\alpha)) \cup V(S_\gamma)$ for each $g' \in Ch(g)$, then $Loss(G, S', g) = Loss(G, S, g)$.

Observation 2: Let g be a node in G. If $\mathcal{M}_{G,S}(g) \in V(S_y) \cup V(S_x) \cup V(S_\gamma)$, then $Loss(G, S', g) = Loss(G, S, g)$.

Observations 1 and 2 show that Cases (4) and (5) do not change the number of losses after an NNI operation is performed. Except Cases (4) and (5), the remaining cases all change the number of losses. In the following, we discuss how to compute the number of losses for the remaining cases separately.

Lemma 10: Let g be a node in G. If $\mathcal{M}_{G,S}(g) \in \{\alpha, \beta\}$, then $Loss(G, S', g)$ can be computed in $O(1)$ time.

Proof: By Lemmas 5, 6, and 7, the mappings $\mathcal{M}_{G,S'}(g)$ and $\mathcal{M}_{G,S'}(g')$ can be derived for each node $g \in V(G)$ and for each node $g' \in Ch(g)$ in $O(1)$ time. With $\mathcal{M}_{G,S'}(g)$ and $\mathcal{M}_{G,S'}(g')$ for each node $g' \in Ch(g)$, we can compute $Loss(G, S', g)$ by Definition 4. Note that we compute the distance $l_S(\mathcal{M}_{G,S}(g), \mathcal{M}_{G,S}(g'))$ by computing the value $|d_S(\mathcal{M}_{G,S}(g)) - d_S(\mathcal{M}_{G,S}(g'))|$, and it can be done in $O(1)$ time. However, after performing an NNI operation on the node

$x \in V(S)$, the two subtrees S_x and S_y are swapped and the value $d_{S'}(\mathcal{M}_{G,S'}(g'))$ may differ from $d_S(\mathcal{M}_{G,S}(g'))$, where $g' \in Ch(g)$. There are the following two conditions to be considered.

- If $\mathcal{M}_{G,S}(g') \in V(S_x)$, then $d_{S'}(\mathcal{M}_{G,S'}(g')) = d_S(\mathcal{M}_{G,S}(g')) - 1$.
- If $\mathcal{M}_{G,S}(g') \in V(S_y)$, then $d_{S'}(\mathcal{M}_{G,S'}(g')) = d_S(\mathcal{M}_{G,S}(g')) + 1$.

It should be mentioned that deciding whether $\mathcal{M}_{G,S}(g') \in V(S_x)$ (or $\mathcal{M}_{G,S}(g') \in V(S_y)$) can be done by checking if $lca(\mathcal{M}_{G,S}(g'), x) = x$ (or $lca(\mathcal{M}_{G,S}(g'), y) = y$) in $O(1)$ time. Therefore, $Loss(G, S', g)$ can be computed in $O(1)$ time. □

Lemma 11: Let g be a node in G. If $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in \{\alpha, \beta\}$ for some $g' \in Ch(g)$, then $Loss(G, S', g)$ can be computed in $O(1)$ time.

Proof: We consider the LCA-mapping of $g' \in Ch(g)$ after performing an NNI operation on x. Note that there exists exactly one child $g' \in Ch(g)$ such that $\mathcal{M}_{G,S}(g') \in \{\alpha, \beta\}$ since $\mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$. There are two conditions to be considered.

- If $\mathcal{M}_{G,S}(g') = \beta$, $Loss(G, S', g)$ is equal to $Loss(G, S, g) - 1$ since $\mathcal{M}_{G,S'}(g') = \alpha$ by Lemma 6. For the computation, we only check each node $i \in \mathcal{M}^{-1}_{G,S}(\beta)$. If $\mathcal{M}_{G,S}(Pa(i)) \notin \{\alpha, \beta\}$, then $Loss(G, S', Pa(i)) = Loss(G, S, Pa(i)) - 1$.
- If $\mathcal{M}_{G,S}(g') = \alpha$, we check whether $\mathcal{M}_{G,S'}(g')$ is equal to α or β in $O(1)$ time according to Lemma 7. If $\mathcal{M}_{G,S'}(g') = \alpha$, then $Loss(G, S', g) = Loss(G, S, g)$. If $\mathcal{M}_{G,S'}(g') = \beta$, then $Loss(G, S', g) = Loss(G, S, g) + 1$. For the computation, we only check each node $i \in \mathcal{M}^{-1}_{G,S}(\alpha)$. If $\mathcal{M}_{G,S}(Pa(i)) \neq \alpha$, we decide the value $Loss(G, S', Pa(i))$ by checking the new mapping $\mathcal{M}_{G,S'}(i)$ according to Lemma 7.

Thus, $Loss(G, S', g)$ can be computed in $O(1)$ time. □

Before discussing Case (3), we define the values $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$. We can compute the number of losses associated to those nodes in Case (3) by using Δ_2 and Δ_3.

Definition 9: For each node $j \in V(S)$, define $\Delta_2(j)$ to be the number of nodes $i \in V(G)$ such that $\mathcal{M}_{G,S}(i) \in V(S_j)$ and $\mathcal{M}_{G,S}(Pa(i)) \in V(S) \backslash V(S_{Pa(j)})$. In other words, $\Delta_2(j)$ is equal to $|\{i \in V(G) : i \in \mathcal{M}^{-1}_{G,S}(V(S_j))$ and $l_S(\mathcal{M}_{G,S}(Pa(i)), \mathcal{M}_{G,S}(i)) - l_S(j, \mathcal{M}_{G,S}(i)) \geq 2\}|$. Similarly, define $\Delta_3(j)$ to be the number of nodes $i \in V(G)$ such that $\mathcal{M}_{G,S}(i) \in V(S_j)$ and $\mathcal{M}_{G,S}(Pa(i)) \in V(S) \backslash V(S_{Pa(Pa(j))})$. In other words, $\Delta_3(j)$ is equal to $|\{i \in V(G) : i \in \mathcal{M}^{-1}_{G,S}(V(S_j))$ and $l_S(\mathcal{M}_{G,S}(Pa(i)), \mathcal{M}_{G,S}(i)) - l_S(j, \mathcal{M}_{G,S}(i)) \geq 3\}|$.

Now we present an efficient algorithm to compute the values Δ_2 and Δ_3 for each node $j \in V(S)$. We compute the value Δ_2 by traversing the species tree S in a bottom-up fashion. For each leaf $z \in Le(S)$, the value $\Delta_2(z)$ can be obtained by computing the number of nodes $i \in \mathcal{M}^{-1}_{G,S}(z)$ satisfying $l_S(\mathcal{M}_{G,S}(Pa(i)), z) \geq 2$. When visiting an internal node $j \in V(S) \backslash Le(S)$, we compute the number of nodes $i \in \mathcal{M}^{-1}_{G,S}(j)$ satisfying $l_S(\mathcal{M}_{G,S}(Pa(i)), j) \geq 2$, and denote the number by $\pi(j)$. Let the value $\delta(j) = |\{i \in V(G) : i \in \mathcal{M}^{-1}_{G,S}(V(S_{Left(j)}) \cup V(S_{Right(j)}))$ and $\mathcal{M}_{G,S}(Pa(i)) = Pa(j)\}|$. It is easy to verify that the value $\Delta_2(j)$ is equal to $\Delta_2(Left(j)) + \Delta_2(Right(j)) + \pi(j) - \delta(j)$.

The remainder is to compute the value $\delta(j)$. For each node $z \in \mathcal{M}^{-1}_{G,S}(Pa(j))$, we only check if there exists some child z' of z satisfying that $\mathcal{M}_{G,S}(z') \in V(S_{Left(j)}) \cup V(S_{Right(j)})$. The checking can be done in $O(1)$ time by verifying whether $lca(\mathcal{M}_{G,S}(z'), Left(j)) = Left(j)$ or $lca(\mathcal{M}_{G,S}(z'), Right(j)) = Right(j)$.

The computation of Δ_3 is similar to that of Δ_2. To compute Δ_3, we also traverse the species tree S in a bottom-up fashion. For each leaf $z \in Le(S)$, the value $\Delta_3(z)$ can be obtained by computing the number of nodes $i \in \mathcal{M}^{-1}_{G,S}(z)$ satisfying $l_s(\mathcal{M}_{G,S}(Pa(i)), z) \geq 3$. When visiting an internal node $j \in V(S) \backslash Le(S)$, we first compute the number of nodes $i \in \mathcal{M}^{-1}_{G,S}(j)$ satisfying $l_s(\mathcal{M}_{G,S}(Pa(i)), j) \geq 3$, and denote the number by $\pi'(j)$. Let the value $\delta'(j) = |\{i \in V(G) : i \in \mathcal{M}^{-1}_{G,S}(V(S_{Left(j)}) \cup V(S_{Right(j)}))$ and $\mathcal{M}_{G,S}(Pa(i)) = Pa(Pa(j))\}|$. It is easy to verify that

$\Delta_3(j) = \Delta_3(Left(j)) + \Delta_3(Right(j)) + \pi'(j) - \delta'(j)$. The computation of $\delta'(j)$ is also similar to that of $\delta(j)$. For each node $z \in \mathcal{M}^{-1}_{G,S}(j)$, we check whether there exists a node $z' \in Ch(z)$ satisfying that $\mathcal{M}_{G,S}(z') \in V(S_{Left(j)}) \cup V(S_{Right(j)})$. This checking can be done in $O(1)$ time by verifying if $lca(\mathcal{M}_{G,S}(z'), Left(j)) = Left(j)$ or $lca(\mathcal{M}_{G,S}(z'), Right(j)) = Right(j)$.

After computing the values $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$, we can cope with Case (3) easily. Let $\Gamma \subseteq V(G)$ be the node set in Case (3), i.e., $\Gamma = \{g \in V(G) : \mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in V(S_y) \cup V(S_x)$ for some $g' \in Ch(g)\}$. After performing an NNI operation on the node $x \in V(S)$, it is easy to see that the total difference of the number of losses caused by the node set Γ is equal to $\Delta_2(y) - \Delta_3(x)$. We conclude the result in the following lemma.

Lemma 12: Let Γ be the node set $\{g \in V(G) : \mathcal{M}_{G,S}(g) \in V(S)\backslash V(S_\alpha)$ and $\mathcal{M}_{G,S}(g') \in V(S_y) \cup V(S_x)$ for some $g' \in Ch(g)\}$. We have that $\sum_{g\in\Gamma}(Loss(G, S', g) - Loss(G, S, g)) = \Delta_2(y) - \Delta_3(x)$.

Algorithm ComputeDelta is shown in Figure 4.4. Now we analyze the time complexity of the algorithm. We assume that $\mathcal{L}_{G,S}(Le(G)) = Le(S)$, and let $|Le(G)| = |Le(S)| = m$. Since visiting all nodes of S, the for-loop in line 1 executes $O(m)$ times. For each iteration of the for-loop in line 1, the time is dominated by the for-loops in lines 3, 11, and 16. Since we visit all nodes of S, the for-loops in lines 3, 11, and 16 totally execute at most $3 \cdot |V(G)| + |V(S)| = O(m)$ times, and each iteration can be done in $O(1)$ time. Therefore, the time complexity of Algorithm ComputeDelta is $O(m)$, and we conclude the result in the following.

Theorem 10: Given a gene tre G, a species tree S, and the LCA-mapping $\mathcal{M}_{G,S}$, Algorithm ComputeDelta computes the values $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$ in $O(m)$ time, where $|Le(G)| = |Le(S)| = m$.

Algorithm ComputeLoss is shown in Figure 4.5. Before we close this section, we analyze the time complexity of the algorithm. In Algorithm ComputeLoss, the time complexity is dominated by the for-loops in lines 3 and 21. The for-loops in lines 3 and 21 execute at most

```
Algorithm ComputeDelta($G, S, \mathcal{M}_{G,S}$)
Input: A gene tree $G$, a species tree $S$, and the LCA-mapping $\mathcal{M}_{G,S}$.
Output: Compute the values $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$.
1  for each node $j \in V(S)$ visited in the postorder do
2      $\Delta_2(j) \leftarrow 0$, $\Delta_3(j) \leftarrow 0$, $\pi(j) \leftarrow 0$, $\delta(j) \leftarrow 0$, $\pi'(j) \leftarrow 0$, $\delta'(j) \leftarrow 0$.
3      for each node $i \in \mathcal{M}^{-1}_{G,S}(j)$ do
4          if $|d_S(\mathcal{M}_{G,S}(i)) - d_S(\mathcal{M}_{G,S}(Pa(i)))| \geq 2$ then
5              $\pi(j) \leftarrow \pi(j) + 1$.
6          if $|d_S(\mathcal{M}_{G,S}(i)) - d_S(\mathcal{M}_{G,S}(Pa(i)))| \geq 3$ then
7              $\pi'(j) \leftarrow \pi'(j) + 1$.
8      if $j \in Le(S)$ then
9          $\Delta_2(j) \leftarrow \pi(j)$, $\Delta_3(j) \leftarrow \pi'(j)$.
10     else
11         for each node $z \in \mathcal{M}^{-1}_{G,S}(Pa(j))$ do
12             for each node $z' \in Ch(z)$ do
13                 if $lca(\mathcal{M}_{G,S}(z'), Left(j)) = Left(j)$ or $lca(\mathcal{M}_{G,S}(z'), Right(j)) = Right(j)$ then
14                     $\delta(j) \leftarrow \delta(j) + 1$.
15         $\Delta_2(j) \leftarrow \Delta_2(Left(j)) + \Delta_2(Right(j)) + \pi(j) - \delta(j)$.
16         for each node $z \in \mathcal{M}^{-1}_{G,S}(Pa(Pa(j)))$ do
17             for each node $z' \in Ch(z)$ do
18                 if $lca(\mathcal{M}_{G,S}(z'), Left(j)) = Left(j)$ or $lca(\mathcal{M}_{G,S}(z'), Right(j)) = Right(j)$ then
19                     $\delta'(j) \leftarrow \delta'(j) + 1$.
20         $\Delta_3(j) \leftarrow \Delta_3(Left(j)) + \Delta_3(Right(j)) + \pi'(j) - \delta'(j)$.
```

Figure 4.4: The algorithm for computing the values $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$.

$2 \cdot (|\mathcal{M}^{-1}_{G,S}(\alpha)| + |\mathcal{M}^{-1}_{G,S}(\beta)|) = O(|\mathcal{M}^{-1}_{G,S}(\alpha)| + |\mathcal{M}^{-1}_{G,S}(\beta)|)$ times. It is easy to verify that each iteration of the two for-loops can be done in $O(1)$ time. Thus, the time complexity of Algorithm ComputeLoss is $O(|\mathcal{M}^{-1}_{G,S}(\alpha)| + |\mathcal{M}^{-1}_{G,S}(\beta)|)$.

Theorem 11: Let G be a gene tree, S be a species tree, and x be a node in $val(S)$. Given G, S, x, $Loss(G,S)$, $\mathcal{M}_{G,S}$, Δ_2, and Δ_3, Algorithm ComputeLoss computes the number of losses $Loss(G, NNI_S(x))$ in $O(|\mathcal{M}^{-1}_{G,S}(\alpha)| + |\mathcal{M}^{-1}_{G,S}(\beta)|)$ time, where $\alpha = Pa(Pa(x))$ and $\beta = Pa(x)$.

```
Algorithm ComputeLoss($G, S, x, Loss(G,S), \mathcal{M}_{G,S}, \Delta_2, \Delta_3$)
Input: A gene tree $G$, a species tree $S$, a node $x \in val(S)$, the number of losses
        $Loss(G,S)$, the LCA-mapping $\mathcal{M}_{G,S}$, and $\Delta_2(j)$ and $\Delta_3(j)$ for each node $j \in V(S)$.
Output: Return the number of losses $Loss(G, NNI_S(x))$.
 1 Let $Pa(x) = \beta$, $Pa(Pa(x)) = \alpha$, $S' = NNI_S(x)$, and $y$ be the sibling of $\beta$.
 2 $Diff_1 \leftarrow 0$, $Diff_2 \leftarrow 0$, and $Diff_3 \leftarrow \Delta_2(y) - \Delta_3(x)$.
 3 for each node $g \in \mathcal{M}^{-1}_{G,S}(\alpha) \cup \mathcal{M}^{-1}_{G,S}(\beta)$ do
 4     Compute the number of losses $Loss(G,S,g)$.
 5     if $\mathcal{M}_{G,S}(g) = \alpha$ and $f(g,x) \neq 1$ then
 6         $\mathcal{M}_{G,S'}(g) \leftarrow \beta$.
 7     else $\mathcal{M}_{G,S'}(g) \leftarrow \alpha$.
 8     for each node $z \in Ch(g)$ do
 9         if $\mathcal{M}_{G,S}(z) \notin \{\alpha, \beta\}$ then
10             $\mathcal{M}_{G,S'}(z) \leftarrow \mathcal{M}_{G,S}(z)$.
11             if $lca(\mathcal{M}_{G,S}(z), x) = x$ then
12                 $d_{S'}(\mathcal{M}_{G,S'}(z)) \leftarrow d_S(\mathcal{M}_{G,S}(z)) - 1$.
13             else if $lca(\mathcal{M}_{G,S}(z), y) = y$ then
14                 $d_{S'}(\mathcal{M}_{G,S'}(z)) \leftarrow d_S(\mathcal{M}_{G,S}(z)) + 1$.
15             else $d_{S'}(\mathcal{M}_{G,S'}(z)) \leftarrow d_S(\mathcal{M}_{G,S}(z))$.
16         else if $\mathcal{M}_{G,S}(z) = \alpha$ and $f(z,x) \neq 1$ then
17             $\mathcal{M}_{G,S'}(z) \leftarrow \beta$.
18         else $\mathcal{M}_{G,S'}(z) = \alpha$.
19     Compute the number of losses $Loss(G,S',g)$.
20     $Diff_1 \leftarrow Diff_1 + Loss(G,S',g) - Loss(G,S,g)$.
21 for each node $i \in \mathcal{M}^{-1}_{G,S}(\beta) \cup \mathcal{M}^{-1}_{G,S}(\alpha)$ do
22     if $\mathcal{M}_{G,S}(i) = \beta$ and $\mathcal{M}_{G,S}(Pa(i)) \notin \{\alpha, \beta\}$ then
23         $Diff_2 \leftarrow Diff_2 - 1$.
24     if $\mathcal{M}_{G,S}(i) = \alpha$ and $\mathcal{M}_{G,S}(Pa(i)) \neq \alpha$ and $i^x \neq 1$ then
25         $Diff_2 \leftarrow Diff_2 + 1$.
26 return $Loss(G,S') \leftarrow Loss(G,S) + Diff_1 + Diff_2 + Diff_3$.
```

Figure 4.5: The algorithm for computing the number of losses $Loss(G, NNI_S(x))$.

4.5 A Linear-Time Algorithm for the DL-NNI Local Search Problem

In this section, we present a linear-time algorithm for the DL-NNI LOCAL SEARCH problem and give the time analysis of this algorithm.

The algorithm proceeds as follows. First, we show how to construct $S|_{\mathcal{L}_{G,S}(Le(G))}$ for each gene tree $G \in \mathcal{G}$ in linear time. After preprocessing the species tree S according to [11], the least common ancestor of any two nodes of S can be answered in $O(1)$ time. For each gene tree G, we maintain a linked list $leaf[G]$. Then we scan the leaves of S in postorder. When we scan a leaf $z \in V(S)$, we check each node $g \in \mathcal{M}^{-1}_{\mathcal{G},S}(z)$. If $g \in V(G)$ and $z \notin leaf[G]$, then we insert z into the tail of $leaf[G]$. After scanning all leaves of S, we complete $leaf[G]$ for all $G \in \mathcal{G}$. Since we scan the leaves of S in postorder, the elements in $leaf[G]$ are also stored in postorder of S for each gene tree $G \in \mathcal{G}$. For each internal node $s \in V(S)$, we maintain a linked list $node[s]$. Let $leaf[G] = \{s_1, s_2, \ldots, s_j\}$. For each linked list $leaf[G]$, we query $lca(s_i, s_{i+1})$ and insert the label 'G' into $node[lca(s_i, s_{i+1})]$ for all $i = 1, \ldots, j-1$. Note that these least common ancestors are the internal nodes of $S|_{\mathcal{L}_{G,S}(Le(G))}$. Thus, the vertex set of $S|_{\mathcal{L}_{G,S}(Le(G))}$ is $V(S|_{\mathcal{L}_{G,S}(Le(G))}) = \{s_1, s_1, ..., s_j\} \cup \{lca(s_i, s_{i+1} : i = 1, \ldots, j-1)\}$. Finally, we traverse the species tree S in postorder. When visiting an internal node $s \in V(S)$, we check each element in the linked list $node[s]$. If a label 'G' is in $node[s]$, then we insert two edges $(s, Left(s))$ and $(s, Right(s))$ into the edge set $E(S|_{\mathcal{L}_{G,S}(Le(G))})$. After completing the traversal of S, we obtain $S|_{\mathcal{L}_{G,S}(Le(G))}$ for each gene tree $G \in \mathcal{G}$.

For each gene tree $G \in \mathcal{G}$ and $S|_{\mathcal{L}_{G,S}(Le(G))}$, we invoke Algorithm INITIALIZELOCALSEARCH and Algorithm COMPUTEDELTA to compute the LCA-mapping, the mutation cost, Δ_2, and Δ_3. For each node $x \in val(S)$, we apply Algorithm COMPUTEGD and Algorithm COMPUTELOSS to computing the numbers of gene duplications and losses after performing an NNI operation on the node x. Then we select the node x^* such that the mutation cost of $NNI_S(x^*)$ is the minimum and output the tree $NNI_S(x^*)$.

Algorithm LINEARNNI is shown in Figure 4.6. Finally, we give the time analysis of the algorithm. Let the set of gene trees $\mathcal{G}$ be $\{G_1, G_2, \ldots, G_k\}$ and the species tree be S, where $|Le(G_i)| = m_i$ for all $1 \le i \le k$ and $|Le(S)| = n$. By [11], line 1 can be done in $O(n)$ time. Lines 2–18 construct $S|_{\mathcal{L}_{G,S}(Le(G))}$ for all $G \in \mathcal{G}$, and it is easy to verify that the execution time of lines 2–18 is $O(\sum_{i=1}^{k} m_i + n)$ totally. The for-loop in line 20 executes total $O(k)$ times. By Theorem 8 and Theorem 10, lines 21 and 22 can be done in $O(|Le(G)|)$ time. Thus, the total execution time of the for-loop in line 20 is $O(\sum_{i=1}^{k} m_i)$. The for-loop in line 25 executes $O(|val(S)|)$ times. The time complexity of each iteration of the for-loop in line 25 is dominated by lines 27–29. By Theorem 9 and Theorem 11, the total execution time of lines 27–29 is $O(\sum_{x \in val(S)} |\mathcal{M}^{-1}_{\mathcal{G},S}(Pa(Pa(x)))| + |\mathcal{M}^{-1}_{\mathcal{G},S}(Pa(x))|) = O(\sum_{x \in V(S)} \mathcal{M}^{-1}_{\mathcal{G},S}(x)) = O(\sum_{i=1}^{k} m_i + n)$. Therefore, the time complexity of Algorithm LinearNNI is $O(\sum_{i=1}^{k} m_i + n)$ and we conclude the result in the following theorem.

Theorem 12: Given a set of gene tree $\mathcal{G}$ and a species tree S, the DL-NNI Local Search problem can be solved by Algorithm LinearNNI in linear time.

```
Algorithm LinearNNI($\mathcal{G}, S$)
Input: A set of gene trees $\mathcal{G}$ and a species tree $S$.
Output: A tree $S^*$ such that $Mut(\mathcal{G}, S^*) = \min_{T \in \{NNI_S(x): x \in val(S)\}} Mut(\mathcal{G}, T)$.
 1 Preprocess the species tree $S$ using the algorithm for computing the LCA in [11].
 2 for each gene tree $G \in \mathcal{G}$ do
 3     $head[leaf[G]] \leftarrow$ NIL, $V(S|_{\mathcal{L}_{G,S}(Le(G))}) \leftarrow \emptyset, E(S|_{\mathcal{L}_{G,S}(Le(G))}) \leftarrow \emptyset$.
 4 for each leaf $z \in Le(S)$ visited in postorder do
 5     for each node $g \in \mathcal{L}^{-1}_{\mathcal{G},S}(z)$ do
 6         if $g \in V(G)$ and $z \in leaf[G]$ then
 7             Insert $z$ into the tail of $leaf[G]$.
 8 for each internal node $s \in V(S)$ do
 9     $head[node[s]] \leftarrow$ NIL.
10 for each gene tree $G \in \mathcal{G}$ do
11     Let the linked list $leaf[G]$ be $\{s_1, s_2, \ldots, s_j\}$.
12     for $i \leftarrow 1$ to $j - 1$ do
13         $V(S|_{\mathcal{L}_{G,S}(Le(G))}) \leftarrow V(S|_{\mathcal{L}_{G,S}(Le(G))}) \cup \{s_i, lca(s_i, s_{i+1})\}$.
14         Insert $G$ into the tail of $node[lca(s_i, s_{i+1})]$.
15     $V(S|_{\mathcal{L}_{G,S}(Le(G))}) \leftarrow V(S|_{\mathcal{L}_{G,S}(Le(G))}) \cup \{s_j\}$.
16 for each internal node $s \in V(S)$ do
17     for each element $G$ in $node[s]$ do
18         $E(S|_{\mathcal{L}_{G,S}(Le(G))}) \leftarrow E(S|_{\mathcal{L}_{G,S}(Le(G))}) \cup \{(s, Left(s)), (s, Right(s))\}$.
19 $|Dup(\mathcal{G}, S)| \leftarrow 0, Loss(\mathcal{G}, S) \leftarrow 0$.
20 for each gene tree $G \in \mathcal{G}$ do
21     InitializeLocalSearch($G, S|_{\mathcal{L}_{G,S}(Le(G))}$).
22     ComputeDelta($G, S, \mathcal{M}_{G,S|_{\mathcal{L}_{G,S}(Le(G))}}$).
23     $|Dup(\mathcal{G}, S)| \leftarrow |Dup(\mathcal{G}, S)| + |Dup(G, S)|, Loss(\mathcal{G}, S) \leftarrow Loss(\mathcal{G}, S) + Loss(G, S)$.
24 $MIN \leftarrow |Dup(\mathcal{G}, S)| + Loss(\mathcal{G}, S)$.
25 for each node $x \in val(S)$ do
26     $GD \leftarrow 0, LOSS \leftarrow 0$.
27     for each gene tree $G \in \mathcal{G}$ do
28         $GD \leftarrow GD +$ ComputeGD($G, S|_{\mathcal{L}_{G,S}(Le(G))}, x, |Dup(G, S)|, \mathcal{M}_{G,S}$).
29         $LOSS \leftarrow LOSS +$ ComputeLoss($G, S|_{\mathcal{L}_{G,S}(Le(G))}, x, Loss(G, S), \mathcal{M}_{G,S}, \Delta_2, \Delta_3$).
30     if $MIN > GD + LOSS$ then
31         $MIN \leftarrow GD + LOSS, pivot \leftarrow x$.
32 return $NNI_S(pivot)$.
```

Figure 4.6: The algorithm for the DL-NNI Local Search problem.

Chapter 5

Concluding Remarks

In this chapter, we summarize the results and discuss the possible future work for the problems studied in this dissertation.

5.1 Summary

In this dissertation, we study two versions of the Multiple Gene Duplication problems: the Episode-Clustering (EC) problem and the Minimum Episodes (ME) problem. In Chapter 3, we give an optimal linear-time algorithm for the EC problem. As a byproduct, we solve the Tree Interval Cover (TIC) problem in linear time. In Chapter 4, we also improve the results in [7] and propose an optimal linear-time algorithm for the ME problem. In addition to the Multiple Gene Duplication problems, we study the heuristic for the Duplication-Loss problem based on the NNI local search and propose a linear-time algorithm for the DL-NNI Local Search problem in Chapter 5.

5.2 Future Work

5.2.1 The Weighted Episode-Clustering Problem

In the EC problem, we want to find a minimum number of locations in the species tree for placing all duplications in the gene trees. We also call this problem the Unweighted Episode-Clustering (Unweighted EC) problem since the locations in the species tree are unweighted. The Unweighted EC problem is based on the assumption that all species in the species tree have the same gene duplication rate. However, there are many studies showing that different species may have different gene duplication rates [21, 31, 32, 33]. For different gene duplication rates of different species in a species tree, therefore, it is reasonable to extend the UNWEIGHTED EC problem to the WEIGHTED EC problem.

Let $\mathcal{G}$ be a set of gene trees and S be a species tree. Let $W : V(S) \to \mathbb{R}^+$ be a weight function for each node $s \in V(S)$. Given $\mathcal{G}, S$, and W, the WEIGHTED EC problem is to find a set of nodes U in S for placing all duplications in $Dup(\mathcal{G}, S)$ such that the sum of weights $\sum_{s \in U} W(s)$ is the minimum. Guo and Niedermeier [28] studied the TREE-LIKE WEIGHTED SET COVER problem and showed that the problem is NP-complete. However, they also showed that the TREE-LIKE WEIGHTED SET COVER problem is fixed-parameter tractable with respect to the maximum subset size. The WEIGHTED EC problem is linear-time reducible to the TREE-LIKE WEIGHTED SET COVER problem, but the time complexity is exponential. To the best of our knowledge, so far there exist no efficient algorithms for the WEIGHTED EC problem. We would like to investigate the WEIGHTED EC problem and see if there exists any efficient algorithm for it.

5.2.2 The DL-k-NNI Local Search Problem

Bansal *et al.* [8] extended the neighborhood of the NNI operation to the k-NNI neighborhood. Given a tree S, the k-NNI neighborhood of S, denoted by $k\text{-}NNI_S$, is the set of trees transformed from S by performing at most k successive NNI operations on any node of S. Given a set of gene tree $\mathcal{G}$ and a initial comparable species tree S, the DL-k-NNI LOCAL SEARCH problem is to find a comparable species tree S^* among $k\text{-}NNI_S$ such that $Mut(\mathcal{G}, S^*) = \min_{T \in k\text{-}NNI_S} Mut(\mathcal{G}, T)$. Given a tree S, it has been shown that $2\text{-}NNI_S$ and $3\text{-}NNI_S$ have very little overlap with the SPR and TBR neighborhoods of S [23, 24]. Bansal *et al.* proposed near-linear time algorithms for the 2-NNI and 3-NNI LOCAL SEARCH problems under the duplication cost in [8]. We would like to devise linear-time algorithms for the DL-2-NNI and DL-3-NNI LOCAL SEARCH problems.

5.2.3 The SPR Local Search Problem

For the Gene Duplication problem, the local search heuristic based on the SPR operation has been considered by Bansal *et. al.* [5, 9]. They presented an $O(kn^2)$-time algorithm for the corresponding local search problem, where k is the number of gene trees and n is the size of the resulting species tree. As a result of the quadratic time complexity, their algorithm is unfavorable to construct the species tree for a large-scale phylogenetic analysis. It should be a challenge to devise a subquadratic time algorithm for the SPR-based local search problem.

Bibliography

[1] B.L. Allen and M. Steel. Subtree Transfer Operations and Their Induced Metrics on Evolutionary Trees. *Annals of Combinatorics*, vol. 5, pp. 1–15, 2001.

[2] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad. Bayesian Gene/Species Tree Reconciliation and Orthology Analysis Using MCMC. *ISMB* (*Supplement of Bioinformatics*), pp. 7–15, 2003.

[3] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad. Gene Tree Reconstruction and Orthology Analysis Based on an Integrated Model for Duplications and Sequence Evolution. *In Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology*, pp. 326–335, 2004.

[4] M.S. Bansal, J.G. Burleigh, and O. Eulenstein. Efficient Genome-Scale Phylogenetic Analysis under the Duplication-Loss and Deep Coalescence Cost Models. *BMC Bioinformatics*, 11(Suppl 1):S42, 2010.

[5] M.S. Bansal, J.G. Burleigh, O. Eulenstein, and A. Wehe. Heuristics for the Gene-Duplication Problem: A $\Theta(n)$ Speed-Up for the Local Search. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology*, pp. 238–252, 2007.

[6] M.S. Bansal and O. Eulenstein. An $\Omega(n^2/\log n)$ Speed-Up of TBR Heuristics for the Gene-Duplication Problem. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 5, no. 4, pp. 514–524, 2008.

[7] M.S. Bansal and O. Eulenstein, "The Multiple Gene Duplication Problem Revisited," *Bioinformatics*, vol. 24, no. 13, pp. i132–i138, 2008.

[8] M.S. Bansal, O. Eulenstein, and A. Wehe. The Gene-Duplication Problem: Near-Linear Time Algorithms for NNI-Based Local Searches. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 6, no. 2, pp. 221–231, 2009.

[9] M.S. Bansal. Algorithms for Efficient Phylogenetic Tree Construction. Ph.D. thesis, Iowa State University, 2009.

[10] M.S. Bansal and R. Shamir. A Note on the Fixed Parameter Tractability of the Gene-Duplication Problem. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, accepted, 2010.

[11] M.A. Bender and M. Farach-Colton. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pp. 88–94, 2000.

[12] G. Blanc, K. Hokamp, and K.H. Wolfe. A Recent Polyploidy Superimposed on Older Large-Scale Duplications in the Arabidopsis Genome. *Genome Research*, vol. 13, pp. 137–144, 2003.

[13] G. Blanc and K.H. Wolfe. Widespread Paleopolyploidy in Model Plant Species Inferred from Age Distributions of Duplicate Genes. *Plant Cell*, vol. 16, pp. 1093–1101, 2004.

[14] P. Bonizzoni, G.D. Vedova, and R. Dondi. Reconciling a Gene Tree to a Species Tree under the Duplication Cost Model. *Theoretical Computer Science*, vol. 347, no. 1–2, pp. 36–53, 2005.

[15] M. Bordewich and C. Semple. On the Computational Complexity of the Rooted Subtree Prune and Regraft Distance. *Annals of Combinatorics*, vol. 8, pp. 409–423, 2004.

[16] J.G. Burleigh, M.S. Bansal, O. Eulenstein, and T.J. Vision. Inferring Species Trees from Gene Duplication Episodes. *ACM-BCB*, accepted, 2010.

[17] J.G. Burleigh, M.S. Bansal, A. Wehe, and O. Eulenstein. Locating Large-Scale Gene Duplication Events through Reconciled Trees: Implications for Identifying Ancient Polyploidy Events in Plants. *Journal of Computational Biology*, vol. 16, no. 8, pp. 1071–1083, 2009.

[18] K. Chen, D. Durand, and M. Farach-Colton. NOTUNG: A Program for Dating Gene Duplications and Optimizing Gene Family Trees. *Journal of Computational Biology*, vol. 7, no. 3–4, pp. 429–447, 2000.

[19] D. Chen, O. Eulenstein, D. Fernández-Baca, and J.G. Burleigh. Improved Heuristics for Minimum-Flip Supertree Construction. *Evolutionary Bioinformatics*, vol. 2, pp. 347–356, 2006.

[20] J.A. Cotton and R.D.M. Page. Tangled Tales from Multiple Markers: Reconciling Conflict between Phylogenies to Build Molecular Supertrees. *Phylogenetic Supertrees: Combing Information to Reveal the Tree of Life*, Springer-Verlag, pp. 107–125, 2004.

[21] J.A. Cotton and R.D.M. Page. Rates and Patterns of Gene Duplication and Loss in the Human Genome. *Proceedings of the Royal Society B*, vol. 272, pp. 277–283, 2005.

[22] M. Fellows, M. Hallett, and U. Stege. On the Multiple Gene Duplication Problem. In *Proceedings of the 9the International Symposium on Algorithms and Computation*, pp. 347–357, 1998.

[23] G. Ganapathy, V. Ramachandran, and T. Warnow. Better Hill-Climbing Searches for Parsimony. In *Proceedings of the 3rd Workshop on Algorithms in Bioinformatics*, pp. 245–258, 2003.

[24] G. Ganapathy, V. Ramachandran, and T. Warnow. On Contract-and-Refine Transformations Between Phylogenetic Trees. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 900-909, 2004.

[25] M. Goodman, J. Czelusniak, G.W. Moore, A.E. Romero-Herrera, and G. Matsuda. Fitting the Gene Lineage into Its Species Lineage, a Parsimony Strategy Illustrated by Cladograms Constructed from Globin Sequences. *Systematic Zoology*, vol. 28, pp. 132–163, 1979.

[26] P. Górecki and J. Tiuryn. On the Structure of Reconciliations. In *Proceedings of the 2nd RECOMB Comparative Genomics Satellite Workshop*, pp. 42–54, 2004.

[27] R. Guigó, I. Muchnik, and T.F. Smith. Reconstruction of Ancient Molecular Phylogeny. *Molecular Phylogenetics and Evolution*, vol. 6, no. 2, pp. 189–213, 1996.

[28] J. Guo and R. Niedermeier. Exact Algorithms and Applications for Tree-Like Weighted Set Cover. *Journal of Discrete Algorithms*, vol. 4, pp. 608–622, 2006.

[29] R. Guyot and B. Keller. Ancestral Genome Duplication in Rice. *Genome*, vol. 47, pp. 610–614, 2004.

[30] M.T. Hallett and J. Lagergren. New Algorithms for the Duplication-Loss Model. In *Proceedings of the 4th Annual International Conference on Computational Molecular Biology*, pp. 138–146, 2000.

[31] M. Lynch and J.S. Conery. The Evolutionary Fate and Consequences of Duplicate Genes. *Science*, vol. 290, pp. 1151–1155, 2000.

[32] M. Lynch and J.S. Conery. Gene Duplication and Evolution. *Science*, vol. 293, pp. 1551, 2001.

[33] M. Lynch and J.S. Conery. The Evolutionary Demography of Duplicate Genes. *Journal of Structural and Functional Genomics*, vol. 3, pp. 35–44, 2003.

[34] B. Ma, M. Li, and L. Zhang. From Gene Trees to Species Trees. *SIAM Journal on Computing*, vol. 30, no. 3, pp. 729–752, 2000.

[35] B. Mirkin, I. Muchnik, and T.F. Smith. A Biologically Consistent Model for Comparing Molecular Phylogenies. *Journal of Computational Biology*, vol. 2, no. 4, pp. 493–507, 1995.

[36] J.E. Neigel and J.C. Avise. Phylogenetic Relationship of Mitochondrial DNA Under Various Demographic Models of Speciation. *Evolutionary Processes and Theory*, Academic Press, Orlando, FL, pp. 515–534, 1986.

[37] S. Ohno. *Evolution by Gene Duplication*, Springer-Verlag, Berlin, 1970.

[38] S. Ohno. Gene duplication and the uniqueness of vertebrate genomes circa 1970–1999. *Seminars in Cell & Developmental Biology*, vol. 10, no. 5, pp. 517–522, 1999.

[39] R.D.M. Page. Maps Between Trees and Cladistic Analysis of Historical Associations Among Genes, Organisms, and Areas. *System Biology*, vol. 43, no.1, pp. 58–77, 1994.

[40] R.D.M. Page. GeneTree: Comparing Gene and Species Phylogenies Using Reconciled Trees. *Bioinformatics*, vol. 14, no. 9, pp. 819–820, 1998.

[41] R.D.M. Page. Extracting Species Trees From Complex Gene Trees: Reconciled Trees And Vertebrate Phylogeny. *Molecular Phylogenetics and Evolution*, vol. 14, no. 1, pp. 89–106, 2000.

[42] R.D.M. Page and J.A. Cotton. Vertebrate Phylogenomics: Reconciled Trees and Gene Duplications. In *Proceedings of the 7th Pacific Symposium on Biocomputing*, pp. 536–547, 2002.

[43] P. Pamilo and M. Nei. Relationships Between Gene Trees and Species Trees. *Molecular Biology and Evolution*, vol. 5, pp. 568–583, 1988.

[44] A.H. Paterson, J.E. Bowers, and B.A. Chapman. Ancient Polyploidization Predating Divergence of the Cereals and its Consequences for Comparative Genomics. *Proc. Natl. Acad. Sci.*, vol. 101, pp. 9903-9908, 2004.

[45] S.A. Rensing, J. Ick, J.A. Fawcett, *et al.* An Ancient Genome Duplication Contributed to the Abundance of Metabolic Genes in the Moss Physcomitrella patens. *BMC Evolutionary Biology*, vol. 7, pp. 130, 2007.

[46] M.J. Sanderson and M.M. McMahon. Inferring Angiosperm Phylogeny from EST Data with Widespread Gene Duplication. *BMC Evolutionary Biology*, 7(Suppl 1):S3, 2007

[47] J. Schlueter, P. Dixon, C. Granger, D. Grant, L. Clark, J. Doyle, and R. Shoemaker. Mining EST Databases to Resolve Evolutionary Events in Major Crop Species. *Genome*, vol. 47, no. 5, pp. 868–876, 2004.

[48] M.E. Schranz and T. Mitchell-Olds. Independent Ancient Polyploidy Events in Sister Families Brassicaceae and Cleomaceae. *Plant Cell*, vol. 18, pp. 1152–1165, 2006.

[49] J.B. Slowinski, A. Knight, and A.P. Rooney. Inferring Species Trees from Gene Trees: A Phylogenetic Analysis of the Elapidae (Serpentes) Based on the Amino Acid Sequences of Venom Proteins. *Molecular Phylogenetics and Evolution*, vol. 8, no. 3, pp. 349–362, 1997.

[50] U. Stege. Gene Trees and Species Trees: The Gene-Duplication Problem is Fixed-Parameter Tractable. In *Proceedings of the 6th Workshop on Algorithms and Data Structures*, pp. 288–293, 1999.

[51] L. Sterck, S. Rombauts, S. Jansson, *et al.* EST Data Suggest that Poplar Is an Ancient Polyploidy. *New Phytol.*, vol. 167, pp. 165–170, 2005.

[52] D.L. Swofford, G.J. Olsen, P.J. Waddel, and D.M. Hillis. Phylogenetic Inference. *Molecular Systematics*, chapter 11, pp. 407–509, Sinauer Associates, 1996.

[53] N. Takahata. Gene Genealogy in Three Related Populations: Consistency Probability Between Gene and Population Trees. *Genetics*, vol. 122, no. 4, pp. 957–966, 1989.

[54] A. Wehe, M.B. Bansal, J.G. Burleigh, and O. Eulenstein. DupTree: a Program for Large-Scale Phylogenetic Analyses Using Gene Tree Parsimony. *Bioinformatics*, vol. 24, no. 13, pp. 1540–1541, 2008.

[55] A. Wehe, W.-C. Chang, O. Eulenstein, and S. Aluru. A Scalable Parallelization of the Gene Duplication Problem. *Journal of Parallel and Distributed Computing*, vol. 70, no. 3, pp. 237–244, 2010.

[56] C.I. Wu. Inferences of Species Phylogeny in Relation to Segregation of Ancient Polymorphisms. *Genetics*, vol. 127, no. 2, pp. 429–435, 1991.

[57] L. Zhang. On a Mirkin-Muchnik-Smith Conjecture for Comparing Molecular Phylogenies. *Journal of Computational Biology*, vol. 4, no. 2, pp. 177–187, 1997.

[58] L. Zhang. Inferring a Species Tree from Gene Trees under the Deep Coalescence Cost. In *Posters of the 4th Annual International Conference on Computational Molecular Biology*, pp. 192–192, 2000.