Algorithms for Genome-Scale Phylogenetics Using Gene Tree Parsimony

Mukul S. Bansal and Oliver Eulenstein

Abstract— **The use of genomic datasets for phylogenetics is complicated by the fact that evolutionary processes such as gene duplication and loss, or incomplete lineage sorting (deep coalescence) cause incongruence among gene trees. One well-known approach that deals with this complication is gene tree parsimony, which, given a collection of gene trees, seeks a species tree that requires the smallest number of evolutionary events to explain the incongruence of the gene trees. However, a lack of efficient algorithms has limited the use of this approach.**

Here, we present efficient algorithms for SPR and TBR based local search heuristics for gene tree parsimony under the (i) duplication, (ii) loss, (iii) duplication-loss, and (iv) deep coalescence reconciliation costs. These novel algorithms improve upon the time complexities of previous algorithms for these problems by a factor of n, where n is the number of species in the collection of gene trees. Our algorithms provide a substantial improvement in runtime and scalability compared to previous implementations and enable large-scale gene tree parsimony analyses using any of the four reconciliation costs. Our algorithms have been implemented in the software packages DupTree and iGTP, and have already been used to perform several compelling phylogenetic studies.

Index Terms— **Gene Tree Parsimony, Gene Duplication, Gene Loss, Incomplete Lineage Sorting, Minimizing Deep Coalescences (MDC), Phylogenomics, Phylogenetics.**

I. Introduction

Genomic datasets provide a wealth of new information for accurate phylogenetic analyses. Traditional phylogenetic approaches make use of either a single gene or a small set of genes taken from the chosen set of species and assume that the evolutionary history of the chosen gene(s) represents the evolutionary history of the species themselves. In contrast, genomic datasets allow us to study the evolution of thousands of genes from across the genomes of the chosen species and make it possible to infer a truly genome-scale phylogeny. However, in order to properly incorporate information from thousands of genes (gene families) from many taxa, one must account for the various evolutionary phenomena like gene duplication and loss, incomplete lineage sorting (deep coalescence), lateral gene transfer, and recombination that affect the evolution of gene families and confound species relationships [1]. One well-studied approach for dealing with this complication is *gene tree parsimony* (GTP) which provides a framework for inferring species phylogenies, or *species trees*, from a collection of gene trees that are confounded by complex evolutionary processes. Given a collection of gene trees, GTP seeks a species tree that contains all taxa represented in the gene trees and implies the minimum reconciliation cost; that is, the fewest number of evolutionary events that explain the incongruence among the gene trees. The reconciliation cost for a given gene tree and species tree is computed by a process known as *reconciliation*, which involves comparing the gene tree with the species tree and invoking evolutionary events to explain the evolution of that gene tree inside the species tree; we refer the reader to [1] for an introductory discussion on reconciliation. The most widely used reconciliation costs are based either on the gene duplication model [2], [3], where the incongruence between gene trees and species trees is assumed to be due to gene duplication and gene loss events, or on the deep coalescence model [1], which assumes that the incongruence is due to incomplete lineage sorting. The GTP problem has been extensively studied under both of these reconciliation models; see, for example, [4]–[23] for GTP under the gene duplication model, and [24]–[33] for GTP under the deep coalescence model. Even though GTP has been shown to be an effective approach for phylogenetic analyses [11], [13], [14], [17], [26], [34], [35], previous algorithms for GTP are too slow to be applied to large-scale datasets, which has limited its use in practice. In this paper, we present novel algorithms that enable, for the first time, GTP analyses of even very large genomic datasets based on both the gene duplication and deep coalescence models.

Under the gene duplication model, two types of reconciliation costs have been traditionally studied in the literature: The duplication cost, and the duplication-loss cost. Specifically, given a gene tree and a species tree, the duplication cost is defined to be the minimum number of duplications required for the reconciliation, while the duplication-loss is defined to be the minimum number of duplications and losses required. Note that the duplication cost ignores the number of inferred losses; this is because, in datasets with incomplete gene sampling, it can be difficult to distinguish gene loss from the absence of sequence data. Thus, the choice of the cost model to be used is based on the nature of the dataset. More recently, a third type of reconciliation cost, the loss cost, which only counts the minimum number of required losses, has also been studied [36]. Using these costs, GTP makes it possible to perform truly genome-scale analyses by making it possible to use phylogenetic information from even those gene families that have a history of gene duplication and loss. Conventional phylogenetic analyses are generally unable to make use of such gene families. Under the deep coalescence model, the deep coalescence cost is defined to be the minimum number of extra lineages (defined formally later) required for the reconciliation. This version of the GTP

M. S. Bansal is with the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology, Cambridge, USA. O. Eulenstein is with the Department of Computer Science, Iowa State University, Ames, USA. mukul@csail.mit.edu, oeulenst@cs.iastate.edu

problem — also commonly referred to as the Minimum Deep Coalescence (MDC) problem [26] — provides a framework to deal systematically with incomplete lineage sorting. Thus, in all, there are four types of reconciliation costs that are commonly used in practice: The (i) duplication cost, (ii) loss cost, (iii) duplication-loss cost, and (iv) deep coalescence cost. It has been shown that, for a given gene tree and species tree, any of these four reconciliation costs can be computed in linear time [16], [31], [36], [37] by constructing an LCA mapping (this is defined formally in Definition 3.2 later) from the nodes of the gene tree to the nodes of the species tree.

The GTP problem is known to be NP-hard for each of these four formulations [10], [22], [32] (but see Section II for a more detailed discussion of the known complexities for different variants). Several methods have been developed to solve the different variants of the GTP problem exactly [19], [20], [26], [38]; while these are useful for computing exact solutions for small datasets, they all have exponential time complexity and therefore can not be applied to datasets with more than a few taxa. Exact polynomial-time solutions also exist, but only for restricted instances [12], [39]. In practice, these problems are typically approached using local search heuristics (e.g., [40], [41]). These heuristics start with some initial candidate species tree and find a minimum reconciliation cost tree in its neighborhood. This constitutes one local search step. The best tree thus found then becomes the starting point for the next local search step, and so on, until a local minima is reached. Thus, at each local search step, the heuristic solves an instance of a "local search problem". The time complexity of this local search problem depends on the tree edit operation used to define the neighborhood, as well as on the type of reconciliation cost used. The two standard tree edit operations used most commonly in practice are the (rooted) subtree prune and regraft (SPR) [42] operation and the (rooted) tree bisection and reconnection (TBR) [43] operation. SPR and TBR operations induce neighborhoods of $\Theta(n^2)$ and $\Theta(n^3)$ trees, respectively, where n is the size of the species tree [44]. Since local search heuristics typically solve the chosen local search problem hundreds or thousands of times during each run, their runtime depends critically on the time complexity of the local search problem. Previous solutions to the SPR and TBR local search problems, for each of the four reconciliation costs, estimate the reconciliation cost from scratch for each tree topology that is evaluated and can therefore only be applied to small datasets.

Our contributions. We present efficient, novel algorithms for SPR and TBR based local searches for the GTP problem under the duplication, loss, duplication-loss, and deep coalescence reconciliation costs. Let us assume, for convenience, that the size of the k given gene trees differs by a constant factor from the size of the resulting species tree. The previously best known (naïve) solutions for the SPR and TBR local search problems, for each of the four reconciliation costs, require $\Theta(kn^3)$ and $\Theta(kn^4)$ time respectively, where n is the size of the resulting species tree. Our algorithms solve these SPR and TBR local search problems in $O(kn^2)$ and $O(kn^3)$ time respectively. Consequently, our algorithms provide a speedup of a factor of n over the best known SPR and TBR local search algorithms (like the ones implemented in [40], [41]) for all four reconciliation costs.

Our efficient algorithms make it possible to perform GTP analyses with hundreds of taxa and thousands of genes, using either the duplication, loss, duplication-loss, or deep-coalescence reconciliation cost. Preliminary versions of the algorithms described in this work first appeared as conference proceedings in [45] and [46], and were incorporated into the software packages DupTree [47] and iGTP [48]. Since then, our algorithms and software have already enabled several compelling phylogenetic studies [49]–[51]. The improvement in runtime and scalability enabled by our algorithms has already been demonstrated earlier [47], [48]. In this work, we further demonstrate that our SPR local search heuristic is highly accurate. Specifically, we apply it to several biological datasets for which optimum GTP solutions had been previously computed using exact methods and observe that the heuristic easily computes the optimal solution on each dataset.

We also explicitly define and distinguish between "trimmed" and "untrimmed" versions of GTP (defined in the next section) based on exactly how the reconciliation cost is computed when the gene tree under consideration has fewer taxa than the species tree. Even though the distinction between these two versions is important, it has been largely ignored by the existing literature on GTP. We elucidate the differences between these two versions and summarize the known complexity results for each.

The current manuscript expands on the preliminary conference versions [45], [46] of this paper in several important ways. Specifically, the current manuscript includes (i) proofs for all lemmas and theorems (previously omitted), (ii) a new section on performing GTP under the deep-coalescence cost (which was considered only very briefly in the preliminary versions), (iii) simplified versions of the algorithms and lemmas/theorems from [45], (iv) results from both papers presented in a unified framework, (v) new experimental results in place of those that were already performed in the preliminary versions, and (vi) a new section discussing trimmed and untrimmed variants of GTP and clarifying their known time-complexities.

This paper is organized as follows: Our discussion on the trimmed and untrimmed versions of the GTP problem occurs in the next section. Section III introduces the basic definitions and notation. In Sections IV, V, and VI we present our algorithms for the GTP problem for the duplication, loss/duplication-loss, and deep coalescence costs respectively. In Section VII we discuss other applications of our algorithms. Experimental results are presented in Section VIII, and concluding remarks appear in Section IX.

II. On Trimmed and Untrimmed Versions of GTP

When reconciling a set of gene trees with a species tree, one often encounters gene trees that only contain genes from a strict subset of the species represented in the species tree. In such cases there are two possible alternatives: (i) We could reconcile the gene tree with a modified species tree obtained by restricting the species tree to just the species represented

in the gene tree, or (ii) we could reconcile the gene tree with the original species tree. We refer to these two alternatives as, respectively, the *trimmed* and *untrimmed* formulations of reconciliation. For instance, under the duplication-loss cost, it may make sense to perform untrimmed reconciliation if whole-genome sequence data is available for each species under consideration (i.e., any inferred losses are likely to be true losses), and trimmed reconciliation otherwise. Much of the existing literature on gene tree parsimony implicitly assumes either one or the other of the two alternatives; for example, [5], [10], [22], [32] assume trimmed reconciliation while [19], [36] assume untrimmed reconciliation.

While the difference between the two alternatives may appear to be a minor one, it is crucial to make this distinction when scoring optimal reconciliations or performing gene tree parsimony. This is because the score of the optimal reconciliation under the loss, duplication-loss, and deep coalescence costs may be different depending on whether trimmed or untrimmed reconciliation is used, which translates into potentially different optimal species trees under gene tree parsimony. Moreover, many of the complexity centric results for GTP have been proved only for the trimmed versions of the problems and, consequently, do not apply to the untrimmed versions. Here, we explicitly consider the seven different possible variants of GTP, depending on the reconciliation cost used and whether the trimmed or untrimmed formulation is used, and clarify the complexity status of each variant by consolidating known complexity results (Table I).

The table shows that, while the complexity of the trimmed variants of GTP is well studied, the complexity of the untrimmed GTP variants remains poorly understood. These untrimmed variants are conjectured to be NP-hard. The differences between trimmed and untrimmed variants of GTP can sometimes be overlooked, and we hope that our explicit consideration of trimmed and untrimmed variants and consolidation of the known complexity results in Table I will help clarify these differences. It is also worth noting that, for the deep coalescence cost, an alternative formulation based on optimally completing incomplete gene trees has been proposed in [33], [53]. This idea of completing incomplete gene trees is similar in spirit to the (+)-method discussed in supertree literature, e.g., [54].

The local search algorithms presented in this paper are applicable to both trimmed and untrimmed reconciliations and to all four cost measures (duplication, loss, duplication-loss, and deep coalescence), i.e., the algorithms in this manuscript can be used to solve any of the seven GTP variants discussed above. For the sake of brevity and clarity, in all subsequent sections, our definitions and algorithms will address only the trimmed versions of GTP. Adapting them to the untrimmed versions is trivial. Furthermore, we will not explicitly define or address the GTP problem on the loss reconciliation cost, since our algorithm for the duplication-loss cost applies directly to the loss reconciliation cost as well.

III. Basic Notation and Preliminaries

Given a rooted tree T, we denote its node set, edge set, and leaf set by $V(T)$, $E(T)$, and $Le(T)$ respectively. The root node of T is denoted by $rt(T)$. Given a node $v \in V(T)$, we denote its parent by $pa_T(v)$, its set of children by $Ch_T(v)$, and the subtree of T rooted at v by T_v. If two nodes in T have the same parent, they are called *siblings*. The set of *internal nodes* of T, denoted $I(T)$, is defined to be $V(T) \setminus Le(T)$. We define $\leq_T$ to be the partial order on $V(T)$ where $x \leq_T y$ if y is a node on the path between $rt(T)$ and x. The *least common ancestor* of a non-empty subset $L \subseteq V(T)$ in tree T, denoted as $lca_T(L)$, is the unique smallest upper bound of L under $\leq_T$. Given $x, y \in V(T)$, $x \rightarrow_T y$ denotes the unique path from x to y in T. We denote by $d_T(x, y)$ the number of edges on the path $x \rightarrow_T y$. T is fully *binary* if every node has either zero or two children. Throughout this paper, the term tree refers to a rooted fully binary tree.

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

A. The (Gene) Duplication and Duplication-Loss Problems

A *species tree* is a tree that depicts the evolutionary relationships of a set of species. Given a gene family for a set of species, a *gene tree* is a tree that depicts the evolutionary relationships of the genes in the gene family. Thus, the nodes in a gene tree represent genes. Species tree and gene trees are both leaf-labelled trees; specifically, each leaf in a species tree is uniquely labeled with the name (or label) of the corresponding species, and each leaf of a gene tree is labelled by the name (or label) of the species from which it was sampled. In order to reconcile a gene tree with a species tree, the species tree must be such that it contains all the species represented in the gene tree. More formally:

Definition 3.1 (Comparability): Given a gene tree G and species tree S, we say that G is *comparable* to S if, for any $g \in Le(G)$, S contains a leaf node with the same label as g. A set of gene trees $\mathcal{G}$ is *comparable* to S if each gene tree in $\mathcal{G}$ is comparable to S.

Given comparable G and S, their leaf labels define a leaf-mapping from the leaf nodes of G to the leaf nodes of S. The minimum number of duplications and/or losses required to reconcile G and S can be computed by generalizing this leaf-mapping to all nodes of the gene tree such that each node $g \in V(G)$ maps to the most recent species in S that could have contained g.

Definition 3.2 (Mapping): Leaf-mapping $\mathcal{L}_{G,S}\colon Le(G) \rightarrow Le(S)$ maps a leaf node $g \in Le(G)$ to that unique leaf node $s \in Le(S)$ which has the same label as g. The extension $\mathcal{M}_{G,S}\colon V(G) \rightarrow V(S)$ of $\mathcal{L}_{G,S}$ is the *mapping* defined by $\mathcal{M}_{G,S}(g) = lca_S(\bigcup_{x \in Le(G_g)} \mathcal{L}_{G,S}(x))$.

Throughout this paper we assume that $\mathcal{G}$ is a set of gene trees that is comparable to a species tree S, and that $Le(S) = \bigcup_{G \in \mathcal{G}} \bigcup_{g \in Le(G)} \mathcal{M}_{G,S}(g)$. The gene tree G will be assumed to be some tree from $\mathcal{G}$.

For any node $s \in V(S)$, we use $\mathcal{M}^{-1}_{G,S}(s)$ to denote the set of nodes in G that map to node $s \in V(S)$ under the mapping $\mathcal{M}_{G,S}$. In addition, for any $X \subseteq V(G)$, we use

Type of reconciliation cost	NP-hardness	Fixed parameter tractability	Approximability
Duplication	NP-hard: shown in [10], [22]	W[2]-hard: shown in [22]	Inapproximable to within log factor: shown in [22]; APX-hard even for five uniquely leaf-labeled gene trees: shown in [52]
Trimmed loss	NP-hard: follows from [22]	W[2]-hard: follows from [22]	Inapproximable to within log factor: follows from [22]
Untrimmed loss	*Unknown*	*Unknown*	*Unknown*
Trimmed duplication-loss	NP-hard: shown in [10], [22]	W[2]-hard: shown in [22]	Inapproximable to within log factor: shown in [22]
Untrimmed duplication-loss	*Unknown*	*Unknown*	*Unknown*
Trimmed deep coalescence	NP-hard: shown in [32], follows from [22]	W[2]-hard: follows from [22]	Inapproximable to within log factor: follows from [22]
Untrimmed deep coalescence	*Unknown*	*Unknown*	*Unknown*

TABLE I

Complexities of different GTP variants. THIS TABLE SHOWS KNOWN RESULTS ON THE NP-HARDNESS, FIXED PARAMETER TRACTABILITY, AND APPROXIMABILITY OF SEVEN GTP VARIANTS. FOR FIXED PARAMETER TRACTABILITY, THE PARAMETER OF INTEREST IS THE TOTAL MINIMUM RECONCILIATION COST. FOR THE INAPPROXIMABILITY RESULTS, n DENOTES THE NUMBER OF TAXA IN THE ANALYSIS. NOTE THAT, FOR THE DUPLICATION COST, WE DO NOT DISTINGUISH BETWEEN TRIMMED AND UNTRIMMED RECONCILIATIONS SINCE THEY ARE IDENTICAL IN TERMS OF THEIR OPTIMAL RECONCILIATION COSTS.

$\mathcal{M}_{G,S}(X)$ to denote the set $\cup_{g\in X}\mathcal{M}_{G,S}(g)$. Based on the above mapping of gene tree nodes to species tree nodes, the minimum number of duplications and losses required for the reconciliation can be computed as follows. Figure S1 in the supplement illustrates how the mapping is constructed and how the number of duplications and losses is computed.

Definition 3.3 (Duplication): A node $g \in I(G)$ is a *(gene) duplication* if $\mathcal{M}_{G,S}(g) \in \bigcup_{g'\in Ch(g)}\mathcal{M}_{G,S}(g')$ and we define *Dup*(G,S) to be the cardinality of the set $\{g \in I(G)\colon\ g \text{ is a duplication}\}$.

Following [12], the number of losses is defined as follows.

Definition 3.4 (Losses): The number of *losses* *Loss*(G,S,g) at a node $g \in I(G)$, is defined to be:

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

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

The reconciliation cost of G with S is defined differently under the duplication and duplication-loss cost models. In particular, under the duplication cost model the reconciliation cost is simply the number of duplications necessary for the reconciliation, while under the duplication-loss model the reconciliation cost is the number of duplications and losses.

Definition 3.5 (Reconciliation cost):

1) Under the duplication cost model, the *reconciliation cost* of G with S, denoted $\Delta^{gd}(G,S)$, is defined to be *Dup*(G,S). Correspondingly, the *reconciliation cost* from $\mathcal{G}$ to S, denoted by $\Delta^{gd}(\mathcal{G},S)$ is defined to be $\sum_{G\in\mathcal{G}}\Delta^{gd}(G,S)$.
2) Under the duplication-loss cost model, the *reconciliation cost* of G with S, denoted $\Delta^{dl}(G,S)$, is defined to be *Dup*(G,S) + *Loss*(G,S). Correspondingly, the *reconciliation cost* from $\mathcal{G}$ to S, denoted by $\Delta^{dl}(\mathcal{G},S)$ is defined to be $\sum_{G\in\mathcal{G}}\Delta^{dl}(G,S)$.

We refer to the GTP problem under the duplication and duplication-loss reconciliation costs as the *duplication* and *duplication-loss* problems, respectively. More formally:

Problem 1 (Duplication): Given a set $\mathcal{G}$ of gene trees, the *(Gene) Duplication* problem is to find a species tree S^* comparable with $\mathcal{G}$, such that $\Delta^{gd}(\mathcal{G},S^*)$ is minimized.

Problem 2 (Duplication-Loss): Given a set $\mathcal{G}$ of gene trees, the *Duplication-Loss* problem is to find a species tree S^* comparable with $\mathcal{G}$, such that $\Delta^{dl}(\mathcal{G},S^*)$ is minimized.

B. Local Search Problems

Here we first provide the definition of an SPR edit operation [42] and then formulate the related local search problems. Informally, an SPR operation on a tree T consists of first pruning out a subtree of T by deleting some edge from $E(T)$ and then regrafting the pruned subtree back into the tree at any location.

Definition 3.6 (Subtree Prune and Regraft (SPR) operation): (See Figure 1) Given a tree T, an edge $\{w,v\} \in E(T)$, where $w = pa(v)$, and a node $y \in V(T) \setminus \{V(T_v) \cup \{w\}\}$, the tree $SPR_T(v,y)$ is obtained from T by cutting the edge $\{w,v\}$, thereby pruning the subtree T_v, and then regrafting the subtree above node y by the same cut edge as follows:

1) *Suppressing the node w.* If w is not the root of T then suppress the degree-two node w. If w is the root of T, delete w and the edge incident with w, making the other end-node of this edge the new root. Denote the resulting tree (without T_v) as $\overline{T_v}$.

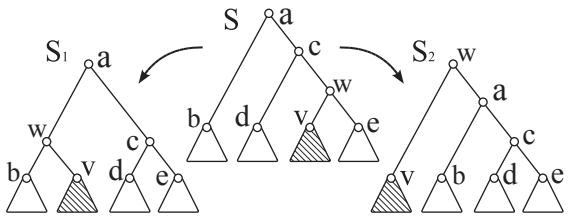

Fig. 1. **SPR operation.** The trees S_1 and S_2 are obtained from S by pruning the subtree rooted at v and regrafting it back into S at two different locations. Specifically, $S_1 = SPR_S(v, b)$ and $S_2 = SPR_S(v, a)$.

2) *Regrafting T_v into $\overline{T_v}$.* This is done in one of the following two ways:
   a) *If $y \in V(\overline{T_v}) \setminus \{rt(\overline{T_v})\}$*: Create a new node w' which subdivides the edge $\{pa_{\overline{T_v}}(y), y\}$ and regraft T_v by the cut edge at node w'.
   b) *If $y = rt(\overline{T_v})$*: Create a new root node w' and a new edge between w' and the original root. Then regraft the subtree by the cut edge at node w'.
3) Rename the node w' in the resulting tree to w. This last step ensures that the set of internal node labels remains the same before and after an SPR operation.

Notation. We define the following:

1. $SPR_T(v) = \bigcup_{y \in \overline{T_v}} \{SPR_T(v, y)\}$
2. $SPR_T = \bigcup_{(w,v) \in E(T)} SPR_T(v)$

Throughout the remainder of this manuscript, v denotes a non-root node in $V(S)$. We now define the relevant local search problems based on the SPR operation.

Problem 3 (Local Search (LS)):
Given $\mathcal{G}$ and S, find a tree $T^* \in SPR_S$ with minimum reconciliation cost.

Our goal is to solve the LS problem efficiently. To that end, we first define a restricted version of the LS problem, called the *Restricted Local Search* problem.

Problem 4 (Restricted Local Search (RLS)):
Given $\mathcal{G}$, S, and v, find a tree $T^* \in SPR_S(v)$ with minimum reconciliation cost.

Under the duplication cost model, the LS and RLS problems will be referred to as the *Duplication-LS (D-LS)* and *Duplication-RLS (D-RLS)* problems respectively. Similarly, under the duplication-loss cost model, the LS and RLS problems will be referred to as the *Duplication-Loss-LS (DL-LS)* and *Duplication-Loss-RLS (DL-RLS)* problems.

Let $n = |Le(S)|$. In the next two sections (i.e., Sections IV and V), we first show how to solve the D-RLS problem in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time and then show how to solve the DL-RLS problem in $O(\sum_{G \in \mathcal{G}} (|V(G)| + n))$ time. Since $SPR_S = \bigcup_{v \in V(S) \setminus \{rt(S)\}} SPR_S(v)$, it is easy to see that the LS problem can be solved by solving the RLS problem $O(n)$ times. This yields $O(\sum_{G \in \mathcal{G}} |V(G)| \cdot n)$– and $O(\sum_{G \in \mathcal{G}} (|V(G)| + n) \cdot n)$–time algorithms for the D-LS and DL-LS problems respectively. Later, in Section VI, we define the deep-coalescence problem and show how to solve the corresponding local search problem in $O(\sum_{G \in \mathcal{G}} (|V(G)| + n) \cdot n)$ time by slightly modifying the algorithms from Section V.

IV. Solving the D-RLS problem

Throughout this section, we shall limit our attention to just one gene tree $G \in \mathcal{G}$; in particular, we show how to solve the D-RLS problem for the instance $\langle \{G\}, S, v \rangle$ in $O(|V(G)| + n)$ time. Our algorithm extends trivially to solve the D-RLS problem on the instance $\langle \mathcal{G}, T, v \rangle$ in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time. Next, we introduce some basic structural properties that are helpful in the current setting. These will also be useful later in Section V.

A. Basic Structural Properties

Let N denote the tree $SPR_S(v, rt(S))$. Observe that, since $SPR_N(v) = SPR_S(v)$, solving the D-RLS problem on instance $\langle \{G\}, S, v \rangle$ is equivalent to solving it on the instance $\langle \{G\}, N, v \rangle$. Thus, in the remainder of this section, we will work with tree N instead of tree S; the motivation for doing so will become apparent as we proceed further. Also, in the remainder of this section, we abbreviate $\mathcal{M}_{G,T}$ to $\mathcal{M}_T$, for any species tree T.

Throughout the remainder of this work, let u denote the sibling of v in N. We color the nodes of N as follows: (i) All nodes in the subtree N_v are colored red, (ii) the root node of N is colored blue, and (iii) all the remaining nodes, i.e. all nodes in N_u, are colored green. See Figure 2 for an example. Correspondingly, we color the nodes of G by assigning to each $g \in V(G)$ the color of the node $\mathcal{M}_N(g)$ (Figure 3).

We define a special tree Γ which is derived from G based on its coloring.

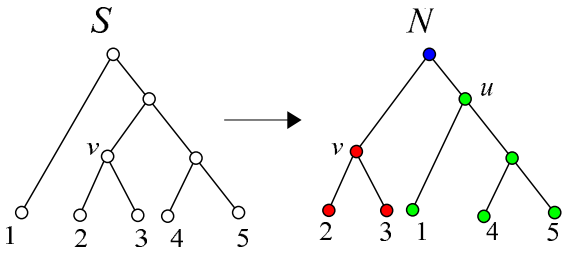

Fig. 2. **Construction and coloring of N.** The figure shows how to construct the tree N from S, as well as the subsequent coloring of the nodes in N.

Definition 4.1 (Γ): We define Γ to be the tree obtained from G by removing all red nodes (along with any edges incident on these red nodes). Observe that while Γ must be binary, it might not be *fully* binary. Observe the trees G and Γ in Figure 3 for an example.

The significance of the tree Γ is that if we build the mapping $\mathcal{M}_{\Gamma,N}$, then, for any non-red node g from $V(G)$, the node $\mathcal{M}_{\Gamma,N}(g)$ represents the most recent common ancestor of exactly all the green descendants of g. This fact will be used in Lemma 4.2.

To understand the behavior of duplications and losses as SPR operations are performed, we must first characterize how the mapping of any node from $I(G)$ changes as N is modified into any $S' \in SPR_S(v)$. This characterization is developed in the next two lemmas. Specifically, Lemma 4.1 states simply that the red and green nodes of G do not change their mappings no matter where the pruned subtree is regrafted,

while Lemma 4.2 characterizes the mappings from the blue nodes of G.

Lemma 4.1: Given G and N, if $g \in V(G)$ is either red or green, then $\mathcal{M}_{S'}(g) = \mathcal{M}_N(g)$ for all $S' \in SPR_N(v)$.

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

Similarly, if g is green, then so are all its descendants. Thus, the mappings from g and its descendants depend only on the green nodes in any $S' \in SPR_N(v)$ and are therefore independent of the placement of the red nodes. Since the subtree N_u is identical in all trees in $SPR_N(v)$, except for the addition of the red nodes, g and its descendants continue to map to the same green nodes under the mappings $\mathcal{M}_N$ and $\mathcal{M}_{S'}$ for any $S' \in SPR_N(v)$. ■

Lemma 4.2: Given G and N, if $g \in V(G)$ is a blue node, then $\mathcal{M}_{S'}(g) = lca_{S'}(v, \mathcal{M}_{\Gamma,N}(g))$ for any $S' \in SPR_N(v)$.

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

From the definition of the tree Γ, it follows that $lca_{S'}(\mathcal{M}_{S'}(G(g))) = \mathcal{M}_{\Gamma,S'}(g)$, and consequently, by Lemma 4.1, $lca_{S'}(\mathcal{M}_{S'}(G(g))) = \mathcal{M}_{\Gamma,N}(g)$. Thus, $\mathcal{M}_{S'}(g) = lca_{S'}(\mathcal{M}_{S'}(R(g)), \mathcal{M}_{\Gamma,N}(g))$.

Now observe that the subtree S'_v contains precisely all the red nodes in S'. Thus, $lca_{S'}(\mathcal{M}_{S'}(R(g)), \mathcal{M}_{\Gamma,N}(g))$ must be identical to $lca_{S'}(v, \mathcal{M}_{\Gamma,N}(g))$, yielding the lemma. ■

We illustrate Lemma 4.2 through an example: Consider the trees G and N depicted in Figure 3. If the subtree N_v is regrafted at the edge $\{u, 1\}$, then the nodes g_1, g_5, and g_6 would map to the node u, and the nodes g_2 and g_3 would map to the newly created node which is the parent of node labeled "1" in the new tree (i.e., to the regraft point, labeled a by our convention).

B. Characterizing Duplications

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

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

The next lemma follows immediately from Lemma 4.1.

Lemma 4.3: Given G and N, if $g \in V(G)$ is either red or green, then $\delta_{S'}(g) = \delta_N(g)$ for all $S' \in SPR_N(v)$.

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

Lemma 4.2 characterizes the behavior of the mapping from any given blue node in G when the tree N is modified into some other tree $S' \in SPR_N(v)$. Lemmas 4.4 through 4.7 are based on this characterization.

Observe that any blue node $g \in V(G)$ must belong to one of the following four categories: (i) g has two blue children, (ii) g has one blue child and one red child, (iii) g has one blue child and one green child, or (iv) g has one red child and one green child. The next four lemmas consider each of these cases separately. Figure 3 illustrates these four lemmas through an example.

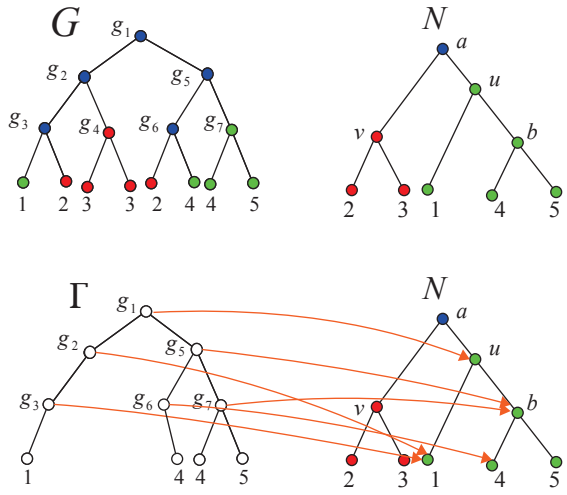

Fig. 3. **Characterizing duplications**. Consider the depicted trees G and N (colored using our scheme), the corresponding tree Γ, and the mapping from Γ to N. This figure illustrates how Lemmas 4.3 through 4.7 characterizes the duplication status of each internal node of G for any species tree in $SPR_N(v)$. Lemma 4.3 implies that for nodes g_4 and g_7, we must have $\delta_{S'}(g_4) = \delta_N(g_4)$ and $\delta_{S'}(g_7) = \delta_N(g_7)$, for all $S' \in SPR_N(v)$. Lemmas 4.4 and 4.5 imply that $\delta_{S'}(g_1) = 1$ and $\delta_{S'}(g_2) = 1$, respectively, for all $S' \in SPR_N(v)$. By Lemma we can infer that $\delta_{S'}(g_5) = 0$ if and only if the pruned subtree is regrafted at the edge $\{u, 1\}$, i.e., if $S' = SPR_N(v, 1)$. Lemma 4.7 implies that, in this example, neither g_3 nor g_6 can be duplications since $V(N_s) \setminus \{s\} = \emptyset$ for both $s = \mathcal{M}_{\Gamma,N}(g_3)$ and $s = \mathcal{M}_{\Gamma,N}(g_6)$.

Lemma 4.4: If $g \in V(G)$ is blue, and g has two blue children, then $\delta_{S'}(g) = 1$ for all $S' \in SPR_N(v)$.

Proof: Let s denote the node $\mathcal{M}_{\Gamma,N}(g)$, $\{s', s''\} = Ch_N(s)$, and $\{g', g''\} = Ch_G(g)$. If either $\mathcal{M}_{\Gamma,N}(g')$ or $\mathcal{M}_{\Gamma,N}(g'')$ is the same as s, then it follows from Lemma 4.2 that $\delta_{S'}(g) = 1$ for all $S' \in SPR_N(v)$. Therefore, let us assume, without any loss of generality, that $\mathcal{M}_{\Gamma,N}(g') \in N_{s'}$ and $\mathcal{M}_{\Gamma,N}(g'') \in N_{s''}$. Let $S' = SPR_N(v, y)$ for some $y \in V(N_u)$. There are now three possible cases:

1) $y \in N_{s'}$: In this case, by Lemma 4.2, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = s$. Therefore, $\delta_{S'}(g) = 1$.
2) $y \in N_{s''}$: In this case, by Lemma 4.2, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = s$. Therefore, $\delta_{S'}(g) = 1$.
3) All other y: Here, we must have $lca_{S'}(v, \mathcal{M}_{\Gamma,N}(g)) = lca_{S'}(v, \mathcal{M}_{\Gamma,N}(g')) = lca_{S'}(v, \mathcal{M}_{\Gamma,N}(g''))$. Consequently, by Lemma 4.2, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S'}(g'')$, and therefore, $\delta_{S'}(g) = 1$.

Thus, since $SPR_N(v) = \bigcup_{y \in N_u} SPR_N(v, y)$, we have $\delta_{S'}(g) = 1$ for all $S' \in SPR_N(v)$. ■

Lemma 4.5: If $g \in V(G)$ is blue, and g has one blue and one red child, then $\delta_{S'}(g) = 1$ for all $S' \in SPR_N(v)$.

Proof: Let g' and g'' denote the red and blue child of g, respectively. By definition, since all nodes in the subtree $G_{g'}$ must be red, they can not appear in the tree Γ. Thus,

in Γ, the node g has only one child, g''. This implies that $\mathcal{M}_{\Gamma,N}(g) = \mathcal{M}_{\Gamma,N}(g'')$. Consequently, by Lemma 4.2, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'')$, i.e. $\delta_{S'}(g) = 1$, for all $S' \in SPR_N(v)$. ■

Lemma 4.6: Let $g \in V(G)$ be a blue node and let s denote the node $\mathcal{M}_{\Gamma,N}(g)$. Let $\{s', s''\} = Ch_N(s)$, and S' be a tree in $SPR_N(v)$. If g has one blue and one green child, denoted g' and g'' respectively, then $\delta_{S'}(g) = 0$ if and only if $\mathcal{M}_{\Gamma,N}(g') \in V(N_{s'})$, $\mathcal{M}_{\Gamma,N}(g'') \in V(N_{s''})$, and $S' = SPR_N(v, y)$ for $y \in V(N_{s'})$.

Proof: Suppose $\mathcal{M}_{\Gamma,N}(g') = s$. Then, by Lemma 4.2, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$, and, consequently, $\delta_{S'}(g) = 1$, for any $S' \in SPR_N(v)$.

Similarly, suppose $\mathcal{M}_{\Gamma,N}(g'') = s$. We have two possible cases: (i) $y \in V(N_s) \setminus \{s\}$, or (ii) $y \notin V(N_s \setminus \{s\})$. In case (i), Lemma 4.2 implies that $\mathcal{M}_{S'}(g) = s$, i.e. $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$. Consequently, $\delta_{S'}(g) = 1$ in this case. In case (ii), Lemma 4.2 implies that $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$ and hence, $\delta_{S'}(g) = 1$.

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

1) $y \in N_{s'}$: In this case, by Lemma 4.2, we have $\mathcal{M}_{S'}(g) = s$. Now since $\mathcal{M}_{\Gamma,N}(g'') \neq s$, by Lemma 4.1, we know that $\mathcal{M}_{\Gamma,S'}(g'') \neq s$. And, since $\mathcal{M}_{\Gamma,N}(g') \in N_{s'}$, we know, by Lemma 4.2, that $\mathcal{M}_{\Gamma,S'}(g') \neq s$ in this case. Thus, $\delta_{S'}(g) = 0$.
2) $y \in N_{s''}$: In this case, by Lemma 4.2, we have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = s$. Thus, $\delta_{S'}(g) = 1$.
3) All other $y \in V(N_u)$: In this case, Lemma 4.2 implies that we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$. Thus, $\delta_{S'}(g) = 1$.

The lemma follows. ■

Lemma 4.7: Let $g \in V(G)$ be a blue node and let s denote the node $\mathcal{M}_{\Gamma,N}(g)$. Let S' be a tree in $SPR_N(v)$. If g has one red and one green child, then $\delta_{S'}(g) = 1$ if and only if $S' = SPR_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$.

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

Observe that $\mathcal{M}_N(g'') = \mathcal{M}_{\Gamma,N}(g'')$, and consequently, by Lemma 4.1, we must have $\mathcal{M}_{S'}(g'') = s$ for all $S' \in SPR_N(v)$. Now, Lemma 4.2 implies that $\mathcal{M}_{S'}(g) = s$ if and only if $S' = SPR_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$. Thus, it follows immediately that $\delta_{S'}(g) = 1$ if and only if $S' = SPR_N(v, y)$ for $y \in V(N_s) \setminus \{s\}$. ■

C. The Algorithm

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

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

The idea behind the algorithm is simple: If $g \in I(G)$ is either red or green, or if it satisfies the preconditions of Lemmas 4.4 or 4.5, then we know that $\delta_{S'}(g) = \delta_N(g)$ for all $S' \in SPR_N(v)$ and, consequently, do nothing. For all other g, we rely on the fact that there exist simple characterizations (Lemmas 4.6 and 4.7) of the regraft locations where those nodes change their duplication status. Specifically,

1) If g satisfies the precondition of Lemma 4.6, then we increment the value of $\alpha(y)$ at each node $y \in V(N_{s'})$ (where s' is as in the statement of Lemma 4.6). To do this efficiently we can simply increment a counter at node s' such that, after all $g \in V(G)$ have been considered, a single post-order traversal of N_u can be used to compute the correct values of $\alpha(y)$ at each $y \in V(N_u)$.
2) If g satisfies the precondition of Lemma 4.7, then we increment the value of $\beta(y)$ at each node $y \in V(N_s) \setminus \{s\}$ (where s is as in the statement of Lemma 4.7). Again, to do this efficiently, we can simply increment a counter at node s such that, after all $g \in V(G)$ have been considered, a single post-order traversal of N_u can be used to compute the correct values of $\beta(y)$ at each $y \in V(N_u)$.

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

Theorem 4.1: The D-RLS problem on instance $\langle \mathcal{G}, S, v \rangle$ can be solved in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time.

Proof: We will show that Procedure-D-RLS solves the D-RLS problem on instance $\langle \mathcal{G}, S, v \rangle$ in $O(\sum_{G \in \mathcal{G}} |V(G)|)$ time. *Correctness:* Since $SPR_N(v) = \bigcup_{y \in N_u} SPR_N(v, y)$, it is sufficient to show that the values $A(t)$ and $B(t)$ are computed correctly at each node $t \in V(N_u)$, where $A(t) = |\{g \in \bigcup_{G \in \mathcal{G}} V(G) \colon \delta_{S'}(g) = 0, \text{but } \delta_N(g) = 1\}|$ and $B(t) = |\{g \in \bigcup_{G \in \mathcal{G}} V(G) \colon \delta_{S'}(g) = 1, \text{but } \delta_N(g) = 0\}|$. Since the values $A(t)$ and $B(t)$, for each $t \in V(N_u)$, are computed according to Lemmas 4.3 through 4.7, the correctness of Procedure-D-RLS follows.

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

The 'for' loop of Step 4 can be executed in $O(\sum_{G \in \mathcal{G}} |V(G)| + n)$ time as follows: During a preprocessing step, with-in $O(n)$ time, we can process the tree N so that *lca* queries on any two nodes in $V(N)$ can be answered in $O(1)$ time; see [37] for details on how to do this. Subsequently, the task of constructing the mapping $\mathcal{M}_{G,N}$ only takes $O(|V(G)|)$ time. Coloring the nodes of G and constructing the mapping $\mathcal{M}_{\Gamma,N}$ also take $O(|V(G)|)$ time. Thus, the total time complexity of this 'for' loop is $O(\sum_{G \in \mathcal{G}} |V(G)| + n)$, which is, since $\sum_{G \in \mathcal{G}} |V(G)| \geq n$, simply $O(\sum_{G \in \mathcal{G}} |V(G)|)$.

Algorithm 1 PROCEDURE-D-RLS

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

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

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

The total time complexity of Procedure-D-RLS is thus $O(\sum_{G\in\mathcal{G}} |V(G)|)$. ■

Theorem 4.2: The D-LS problem on instance $\langle \mathcal{G}, S\rangle$ can be solved in $O(\sum_{G\in\mathcal{G}} |V(G)| \cdot n)$ time.

Proof: Observe that $SPR_S = \bigcup_{v\in V(S)\setminus\{rt(S)\}} SPR_S(v)$. The theorem therefore follows immediately from Theorem 4.1 ■

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

V. SOLVING THE DL-RLS PROBLEM

As before, we limit our attention to one gene tree G; in particular, we show how to solve the DL-RLS problem for G in $O(|V(G)| + n)$ time. Our algorithm extends trivially to solve the DL-RLS problem on the set of gene trees $\mathcal{G}$ in $O(\sum_{G\in\mathcal{G}}(|V(G)| + n))$ time. In keeping with the definition of the trimmed version of GTP, we will assume that $Le(G) = Le(S)$. In practice, if $Le(G) \neq Le(S)$ then we simply set the species tree to be $S[Le(G)]$; this takes $O(|Le(G)| + n)$ time and, consequently, does not affect the time complexity of our algorithm.

To solve the DL-RLS problem for G, it is sufficient to compute the values $Dup(G, S')$ and $Loss(G, S')$ for each $S' \in SPR_S(v)$. In the previous section we showed how to compute the value $Dup(G, S')$ for each $S' \in SPR_S(v)$, in $O(m)$ time. We now show how to compute the value $Loss(G, S')$ for each $S' \in SPR_S(v)$ in $O(m)$ time as well. Altogether, this implies that the DL-RLS problem for G can be solved in $O(m)$ time.

As in the previous section we will work with the tree $N = SPR_S(v, rt(S))$. Also recall the coloring scheme of the nodes of N (Figure2), the definition of the tree Γ, and Lemmas 4.1 and 4.2 from Section IV-A.

A. Characterizing Losses

To solve the DL-RLS problem efficiently we rely on the following six lemmas, which make it possible to efficiently infer the value of $Loss(G, S', g)$ for any $S' \in SPR_N(v)$ and $g \in I(G)$.

Consider any $g \in I(G)$, and let g' and g'' be its two children. Let $a = \mathcal{M}_N(g)$, $b = \mathcal{M}_N(g')$ and $c = \mathcal{M}_N(g'')$. Without loss of generality, node g must correspond to one of the following six categories: 1) g is red, 2) g is green, 3) g, g', and g'' are all blue, 4) g and g' are blue, and g'' is green, 5) g and g' are blue, and g'' is red, or, 6) g is blue, g' is red, and g'' is green.

Lemmas 5.1 through 5.6 characterize the behavior of the loss cost $Loss(G, S', g)$, for each $S' \in SPR_N(v)$, for each of these six cases. Losses behave in a more complex manner than duplications and, therefore, Lemmas 5.1 through 5.6 are more technical than the corresponding lemmas for duplications presented in the previous section. For instance, some gene tree nodes and their children may change their mappings as the pruned subtree is regrafted along different edges, affecting their loss counts. For some other nodes, regrafting the pruned subtree may not affect their mappings but may simply increase the length of the path from a parent to one of its children, resulting in an increase in the loss cost for the parent. In some cases, certain nodes may either gain or lose duplication status depending on the regraft location, affecting the way in which losses must be computed at those nodes. All these different possibilities complicate the characterization of losses. The underlying idea, however, remains the same and these new

lemmas are based directly on the characterization of mappings developed in Lemmas 4.1 and 4.2.

At this point, it would help to observe that $SPR_N(v) = \{SPR_N(v,s)\colon s \in V(N_u)\}$. The next two lemmas follow easily from Lemma 4.1.

Lemma 5.1: If g is red then $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,N,g)$ for all $S' \in SPR_N(v)$.

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

Lemma 5.2: If g is green then $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,N,g)+1$ if $S' = SPR_N(v,x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, and $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,N,g)$ otherwise.

Proof: Since g is green, so are g' and g'', and therefore, by Lemma 4.1 we must have $\mathcal{M}_{S'}(y) = \mathcal{M}_N(y)$ for any $S' \in SPR_N(v)$ and $y \in \{g,g',g''\}$. Thus, if $S' = SPR_N(v,x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, then either $d_{S'}(a,b) = d_N(a,b)+1$ or $d_{S'}(a,c) = d_N(a,c)+1$; and $d_{S'}(a,b) = d_N(a,b)$, $d_{S'}(a,c) = d_N(a,c)$ otherwise. Hence, following Def. 3.4, $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,N,g)+1$ if $S' = SPR_N(v,x)$ where $b \leq_N x <_N a$ or $c \leq_N x <_N a$, and $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,N,g)$ otherwise. ■

The next four lemmas are more technical. While the previous two lemmas express the new loss cost for a node, g, in terms of the change from its loss cost against tree N, the following lemmas will often express the new loss cost in terms of how it changes as the pruned subtree is regrafted, step-by-step, one edge lower in the tree (i.e., onto a child edge).

Lemma 5.3: Let g, g' and g'' all be blue nodes, $x \in V(N_u)$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$.

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

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

- Part 1: Since $x \not\leq_N a'$, we must have $lca_N(v,a') = lca_N(v,b') = lca_N(v,c')$. Therefore, by Lemma 4.2, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S'}(g'')$ where $S' = SPR_N(v,x)$ and $x \not\leq_N a'$. Thus, for each S' in this case, we must have $\mathit{Loss}(G,S',g) = 0 = \mathit{Loss}(G,N,g)$.
- Part 2(a): This case is relevant only if at least one of b' or c' is not the same as a'. Therefore, without any loss of generality we may assume that $b' \neq a'$. Suppose $S' = SPR_N(v,x)$ where $b' \leq_N x <_N a'$; then, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $d_{S'}(a',\mathcal{M}_{S'}(g')) = d_N(a',x)$. Also, if b'' denotes the child of a in tree N along the path $a' \rightarrow_N b'$, then, by Def. 3.4, we must have $\mathit{Loss}(G,SPR_N(v,b''),g) = 1$, which is indeed one greater than $\mathit{Loss}(G,SPR_N(v,a'),g)$. Thus, $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,S'',g)+1$ if $b' \leq_N x <_N a'$. The argument for the case when $c' \leq_N x <_N a'$ is completely analogous.
- Part 2(b): Let b'' denote the child of a along the path $a' \rightarrow_N b'$, and c'' denote the child of a along the path $a' \rightarrow_N c'$, in tree N. When $x <_N a'$ but neither $b' \leq_N x <_N a'$ nor $c' \leq_N x <_N a'$, we must have either (i) $x <_N b''$ but not such that $c' \leq_N x <_N a'$, or (ii) $x <_N c''$ but not such that $b' \leq_N x <_N a'$. In case (i), we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = \mathcal{M}_{S'}(g'') = \mathcal{M}_{S'}(g'') = a'$, and both $\mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g')$ must be nodes along the path $b'' \rightarrow_{S''} b'$ such that $d_{S'}(a',\mathcal{M}_{S'}(g')) = d_{S''}(a',\mathcal{M}_{S''}(g'))$ (note that this is true even if $b' \leq_N pa(x) <_N a'$). Thus, for case (i), $\mathit{Loss}(G,S',g) = \mathit{Loss}(G,S'',g)$. An analogous argument holds for case (ii).

■

Lemma 5.4: Let g and g' be blue nodes and g'' be a green node, $x \in V(N_u) \setminus \{u\}$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$.

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

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

- Part 1(a): For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_{S'}(x)$, and $\mathcal{M}_{S'}(g'') = c'$.

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

- Part 1(b): In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_N(x)$, and $\mathcal{M}_{S'}(g'') = c'$, and, $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g') = pa_{S''}(x)$, and $\mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c') - 1$. Hence, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) - 1$.
- Part 1(c): In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = lca_{S''}(x, a')$ and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c')$. Thus, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g)$.
- Part 2(a)(i): Let T and T' denote the trees $SPR_N(v, a')$ and $SPR_N(v, b'')$, respectively. Then, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$ and $\mathcal{M}_T(g'') = c'$, and, $\mathcal{M}_{T'}(g) = a'$, $\mathcal{M}_{T'}(g') = pa_{T'}(b'')$ and $\mathcal{M}_{T'}(g'') = c'$. For the case when $a' \neq c'$ we must therefore have $d_T(\mathcal{M}_T(g), c') = d_{T'}(\mathcal{M}_T(g), c') + 1$, and $d_T(\mathcal{M}_{T'}(g), \mathcal{M}_{T'}(g')) = 1$. Note that while g is a duplication under mapping $\mathcal{M}_T$, it is not one under mapping $\mathcal{M}_{T'}$. Thus, by Definition 3.4, we must have $\mathit{Loss}(G, T', g) = \mathit{Loss}(G, T, g) - 2$. For the case when, $a' = c'$, we must have $\mathcal{M}_T(g'') = a'$, and therefore, $d_T(\mathcal{M}_T(g), \mathcal{M}_T(g'')) = 1$, $d_T(\mathcal{M}_T(g), \mathcal{M}_T(g')) = 0$, and, $d_{T'}(\mathcal{M}_T(g), \mathcal{M}_T(g'')) = 0$, $d_{T'}(\mathcal{M}_T(g), \mathcal{M}_T(g')) = 1$. Thus, $\mathit{Loss}(G, T', g) = \mathit{Loss}(G, T, g)$.
- Part 2(a)(ii): This case is relevant only if $b' \neq b''$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, $\mathcal{M}_{S'}(g') = pa_{S'}(x)$ and $\mathcal{M}_{S''}(g') = pa_{S''}(pa(x))$. Thus, $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g')) + 1$, and consequently $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$.
- Part 2(a)(iii): In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, and both $\mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g')$ must be nodes along the path $b'' \rightarrow_{S''} b'$ such that $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g'))$ (note that this is true even if $b' \leq_N pa(x) <_N a'$). The result follows.
- Part 2(a)(iv): This case exists only if $a' \neq c'$. Let T denote the tree $SPR_N(v, a')$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$. Therefore, $d_{S'}(a', c') = d_T(\mathcal{M}_T(g), c') = d_N(a', c') + 1$. Thus, if $c' \leq_N x <_N a'$, then $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, SPR_N(v, a'), g)$.
- Part 2(a)(v): Let c'' denote the sibling of b'' in tree N. Then, in this case, we must have $x \in N_{c''}$. Moreover, x is not such that $c' \leq_N x <_N a'$. Thus, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_{S'}(g'') = c'$. Also, for the tree $T = SPR_N(v, a')$, we have $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$ and $d_T(\mathcal{M}_{S'}(g), c') = d_{S'}(a', c') + 1$. Hence, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, T, g) - 1$.
- Part 2(b)(i): The proof for this part is identical to the proof of part *2.(a).iv.*
- Part 2(b)(ii): Let T denote the tree $SPR_N(v, a')$. There are two possible cases, either $a' = c'$ or $a' \neq c'$. For $a' = c'$, we must have $\mathit{Loss}(G, T, g) = 1$ and $\mathit{Loss}(G, S', g) = 0$. For $a' \neq c$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_T(g'') = c'$; and hence $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, T, g) - 1$. Thus, part *2.(b).ii.* of the lemma holds for both cases.

■

Lemma 5.5: Let g and g' be blue nodes and g'' be a red node, $x \in V(N_u)$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$.

1) If $S' = SPR_N(v, x)$ where $x <_N a'$, and $S'' = SPR_N(v, pa(x))$, then $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$.
2) If $S' = SPR_N(v, x)$ where $x \not<_N a'$, then,
   a) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, N, g)$ if $a' \leq_N x \leq_N u$, and,
   b) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$ for $S'' = SPR_N(v, pa(x))$ otherwise.

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

- Part 1: If $x <_N a'$, then $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$ and $\mathcal{M}_{S'}(g'') = c$. Since g'' is red, c must be a node in the pruned subtree N_v, therefore, assuming $S'' \neq SPR_N(v, a')$, we must have $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$ and, consequently, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$. If $S'' = SPR_N(v, a')$, then we have $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g') = pa_{S''}(a')$ and $\mathcal{M}_{S''}(g'') = c$. And therefore, again, we must have $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$, implying $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$.
- Part 2(a): In the tree N we have $\mathcal{M}_N(g) = \mathcal{M}_N(g') = \mathit{rt}(N)$ and $d_N(\mathit{rt}(N), c) = d_{N_v}(v, c) + 1$. Similarly, if $a' \leq_N x \leq_N u$, then $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_{S'}(x)$ and, consequently, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{N_v}(v, c) + 1$. Thus, in this case $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, N, g)$.
- Part 2(b): We have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g')$. Now, if $a' \leq_N pa(x) \leq_N u$, then we must have $\mathcal{M}_{S''}(g) = pa_{S''}(pa_N(x))$ and $\mathcal{M}_{S'}(g) = pa_N(x)$, and therefore, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$; otherwise, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = lca_N(x, a')$ and therefore, again, $d_{S'}(\mathcal{M}_{S'}(g), c) = d_{S''}(\mathcal{M}_{S''}(g), c) + 1$. Part 2. (b) of the lemma now follows directly.

■

Lemma 5.6: Let g be blue, g' be red, and g'' be green. Let $x \in V(N_u) \setminus \{u\}$ and $a' = \mathcal{M}_{\Gamma,N}(g)$.

1) If $S' = SPR_N(v, x)$ where $x \not<_N a'$, and $S'' = SPR_N(v, pa(x))$, then,
   a) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) - 1$ if $a' \leq_N x <_N u$,

b) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g)$ if $a' \leq_N \mathit{pa}(x) <_N u$ but x is not such that $a' \leq_N x <_N u$, and,

c) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$ otherwise.

2) If $S' = SPR_N(v, x)$ where $x <_N a'$, and $S'' = SPR_N(v, \mathit{pa}(x))$, then,

a) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, SPR_N(v, a'), g) + 2$ if $x \in \mathit{Ch}_N(a')$, and,

b) $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$ otherwise.

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

- Part 1(a): For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = \mathit{pa}_{S'}(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$. Also observe that the same holds for the case when $x = u$. Thus, for each x such that $a' \leq_N x <_N u$, we have $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'))$ and $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g'')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'')) - 1$. Part 1.(a) of the lemma follows immediately.
- Part 1(b): In this case, we must have $\mathcal{M}_{S'}(g) = \mathit{pa}_N(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$, and, $\mathcal{M}_{S''}(g) = \mathit{pa}_{S''}(x)$, $\mathcal{M}_{S''}(g') = b$ and $\mathcal{M}_{S''}(g'') = a'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and $d_{S'}(\mathcal{M}_{S'}(g), a') = d_{S''}(\mathcal{M}_{S''}(g), a') - 1$. Hence, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g)$.
- Part 1(c): In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = lca_{S''}(b, a')$, $\mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = b$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = a'$. Now since b is a node in the pruned subtree N_v, we must have $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$.
- Part 2(a): If $x \in \mathit{Ch}_N(a')$ then we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Thus, $\mathit{Loss}(G, S', g) = |d_{S'}(a', b) - 1| + 1$. Now, let T denote the tree $SPR_N(v, a')$, then we must have $\mathcal{M}_T(g) = \mathit{pa}_T(a')$, $\mathcal{M}_T(g'') = a'$ and $\mathcal{M}_T(g') = b$. Thus, $\mathit{Loss}(G, T, g) = |d_T(\mathit{pa}_T(a'), b) - 1|$. Finally, observe that $d_{S'}(a', b) = d_T(\mathit{pa}_T(a'), b) + 1$, and hence, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, T, g) + 2$.
- Part 2(b): For any $x <_N a'$, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Since b is a node in the pruned subtree N_v and in this case $x <_N y$ for $y \in \mathit{Ch}(a')$, we must have $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $\mathit{Loss}(G, S', g) = \mathit{Loss}(G, S'', g) + 1$.

■

B. The Algorithm

Observe that $SPR_N(v) = \{SPR_N(v, s) \colon s \in V(N_u)\}$. Therefore, the goal of our algorithm is to compute at each node $s \in V(N_u)$ the value $\mathit{Loss}(G, S')$, where $S' = SPR_N(v, s)$. To do this efficiently, we rely on the characterization of losses given in Lemmas 5.1 through 5.6.

The first step of the algorithm is to compute the value $\mathit{Loss}(G, N)$. This "loss value" is assigned to the node u. The basic idea behind the algorithm is the same as that for the algorithm for computing duplications, developed in the previous section. Recall that the algorithm from the previous section made use of two counters ($\alpha(x)$ and $\beta(x)$) to capture the change in duplication status as characterized by the lemmas. Due to the increased complexity of the characterization of losses (Lemmas 5.1 through 5.6), however, the current algorithm requires many more types of counters to efficiently capture the different patterns of the change in the loss cost. Specifically, to compute the loss value for the rest of the nodes (besides u) our algorithm makes use of six different types of *counters* at each node in N_u; we refer to these counters as counter-i, for $i \in \{1, \ldots, 6\}$. These counters make it possible to efficiently compute the difference between the values $\mathit{Loss}(G, N)$ and $\mathit{Loss}(G, S')$, where $S' = SPR_N(v, s)$, for each $s \in V(N_u)$. Next, we describe each of these six counters; throughout our description, s represents some node in N_u.

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

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

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

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

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

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

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

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

Let $a = \mathcal{M}_N(g)$, $b = \mathcal{M}_N(g')$ and $c = \mathcal{M}_N(g'')$. Also, whenever properly defined, let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$. Based on Lemmas 5.1 through 5.6, we now study how the six counters can be updated so as to capture the behavior of losses in each of these cases.

- Case 1: By Lemma 5.1, we do nothing in this case.
- Case 2: Based on Lemma 5.2, the contribution of any node g that satisfies the condition of case 2 can be

captured by incrementing the value of counter-1 by one at each node on paths $a \rightarrow_N b$ and $a \rightarrow_N b$, except at node a.

- Case 3: From Lemma 5.3 it follows that in this case the contribution of g to the loss value changes in a way that is captured by incrementing counter-2 by 1, at each node, except a', on the paths $a' \rightarrow_N b'$ and $a' \rightarrow_N c'$.
- Case 4: According to Lemma 5.4, if N_v is regrafted on an edge of N_u that is not in $N_{a'}$, then the contribution of g to the loss cost is captured by incrementing counter-3 by 1 at each node except u along the path $u \rightarrow_N a'$, and at their siblings. If N_v is regrafted on an edge of N_u that is in $N_{a'}$ then there are two possible cases:
  - $a' \neq b'$: Recall that b'' represents the child of a' along the path $a' \rightarrow_N b'$. Here, the contribution of g to the loss cost is captured by (i) incrementing counter-3 by two at node b'', (ii) incrementing counter-2 by one at each node except b'' along the path $b'' \rightarrow_N b'$, (iii) incrementing counter-3 by one at the sibling of b'', and (iv) incrementing counter-1 by one at each node except a' on the path $a' \rightarrow_N c'$.
  - $a' = b'$: In this case, the contribution of g to the loss cost is captured by (i) incrementing counter-3 by one at both children of a', and (ii) incrementing counter-1 by one at each node except a' on the path $a' \rightarrow_N c'$.
- Case 5: By Lemma 5.5, for this case, the change in the loss contribution of g is captured by incrementing counter-5 by 1 at node a', and by incrementing counter-4 by 1 at both children of a' in N.
- Case 6: By Lemma 5.6, for this case, the change in the loss contribution of g is captured by incrementing counter-6 by 1 at node a', and by incrementing counter-4 and counter-2 by 1 each at both children of a' in N.

2) Computing the final loss values: Our algorithm considers each internal node of gene tree G, one at a time, and updates the relevant counters at the relevant nodes in N_u, as shown in the previous subsection. Then, based on the counters, it computes, at each node $s \in V(N_u)$ the value $\alpha(s) = \textit{Loss}(G, S') - \textit{Loss}(G, N)$, where $S' = SPR_N(v, s)$; this can be achieved by performing a constant number of pre- and post-order traversals of N_u. A final traversal of the tree now allows us to compute the value $\textit{Loss}(G, S') = \alpha(s) + \textit{Loss}(G, N)$ at each $s \in V(N_u)$. A complete description of our algorithm to solve the DL-RLS problem on instance $\langle \{G\}, S, v \rangle$ appears in Procedure-DL-RLS (see Algorithm 2).

Lemma 5.7: Procedure-DL-RLS solves the DL-RLS problem on the instance $\langle \{G\}, S, v \rangle$.

Proof: Procedure-DL-RLS computes the duplication cost $\textit{Dup}(G, SPR_N(v, s))$, at each $s \in V(N_u)$, as shown in Section IV. It then computes the values of the six counters, i.e. counter-i, for $i \in \{1, \ldots, 6\}$, in accordance with Lemmas 5.1 through 5.6. Then, in Steps 11 through 13, the algorithm encodes the changes in loss cost implied by counter-4, counter-5, and counter-6, at each node in N_u, in terms of the values of counter-1, counter-2, and counter-3. Procedure-Loss then correctly computes the value of $\alpha(s)$ at each $s \in V(N_u)$ based on these three counters. Thus, for any $SPR_S(v, s)$, where $s \in V(N_u)$, the values $\textit{Dup}(G, SPR_N(v, s))$ and $\alpha(s)$ are computed correctly. Note that the reconciliation cost of G and $SPR_S(v, s)$, where $s \in V(N_u)$, is given by $\textit{Dup}(G, SPR_N(v, s) + \alpha(s) - \textit{Loss}(G, N)$. The lemma follows. ■

Algorithm 2 Procedure-DL-RLS

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

Algorithm 3 Procedure-Loss

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

Lemma 5.8: Procedure-DL-RLS can be implemented to run in $O(|V(G)| + n)$ time.

Proof: We analyze the complexity of Procedure-DL-RLS step-by-step. The total time complexity of Step 2 is $O(|V(G)|+n)$ and of Steps 3 and 4 is $O(n)$. Next, Step 5 can be implemented in $O(|V(G)|+n)$ time as follows: During an $O(n)$-time preprocessing step we can process the tree N so that *lca* queries on any two nodes in $V(N)$ can be answered

in $O(1)$ time; see [37] for details. Subsequently, the task of constructing the mapping $\mathcal{M}_{G,N}$ only takes $O(|V(G)|)$ time. Coloring the nodes of G and N and constructing the mapping $\mathcal{M}_{\Gamma,N}$ also take $O(|V(G)|)$ time. Thus, the total time complexity of this step is $O(|V(G)|+n)$. In Step 6, the value $\mathit{Loss}(G,N)$ can be computed in $O(|V(G)|+n)$ time by first traversing through N to compute the depth of each node, and then traversing through G and computing $\mathit{Loss}(G,N,g)$ for each $g \in I(G)$ according to Definition 3.4. By the result of the previous section, the 'for' loop of Step 7 requires $O(|V(G)|+n)$ time as well.

Let us now consider the 'for' loop of Step 9 in detail. For any g that satisfies the criteria for case 1, there is nothing to be done. For any g satisfying the criterion for cases 5, or 6, we are required to update the counters at a constant number of nodes in N_u. Therefore, all such g can be handled within $O(|V(G)|)$ time. However, for cases 2, 3 and 4, handling each g might, in the worst case, require updating the counters at $\Theta(n)$ nodes, yielding a total time complexity of $O(n \cdot |V(G)|)$ for these cases. This happens because these cases require us to update specific counters along the entire length of certain paths in N_u. A simple way to deal with this issue is to only mark the start node and end node of the path for the specified counter. Once this is done for each g satisfying the criterion for cases 2, 3 or 4, we can perform a post-order traversal and set all the relevant counters to their correct value based on the marked start and end nodes. In this way, cases 2, 3, and 4 can be handled in a total of $O(|V(G)|+n)$ time as well. This gives us a total time complexity of $O(|V(G)|+n)$ for computing all the counters.

In Steps 11 through 13, the algorithm encodes the changes in loss cost implied by counter-4, counter-5, and counter-6, at each node in N_u, in terms of the other counters. It is easy to verify that each of these steps can be implemented to run in $O(n)$ time by doing either a post-order or pre-order traversal of N_u, as appropriate. Step 14 calls Procedure-Loss on parameters $\langle u, 0\rangle$. Procedure-Loss simply performs a pre-order traversal of the tree N_u, spending $O(1)$ at each node. The time complexity of Step 14 is thus $O(n)$. Finally, in Step 15, computing the reconciliation cost for each $\mathit{SPR}_S(v,s)$, where $s \in V(N_u)$, takes $O(n)$ time. ∎

Thus, we have the following two theorems.

Theorem 5.1: The DL-RLS problem can be solved in $O(\sum_{G\in\mathcal{G}}(|V(G)|+n))$ time.

Proof: By Lemmas 5.7 and 5.8 we know that the DL-RLS problem on the restricted instance $\langle\{G\}, S, v\rangle$ can be solved in $O(m)$ time. It is straightforward to extend Procedure-DL-RLS to solve the DL-RLS problem by considering each gene tree in $\mathcal{G}$ separately and combining the computed reconciliation costs. ∎

Theorem 5.2: The DL-LS problem can be solved in $O(\sum_{G\in\mathcal{G}}(|V(G)|+n)\cdot n)$ time.

Proof: Observe that $\mathit{SPR}_S = \bigcup_{v\in V(S)\setminus\{rt(S)\}} \mathit{SPR}_S(v)$. The theorem therefore follows immediately from Theorem 5.1. ∎

The time complexity of the best known (naïve) solution for the DL-LS problem is $\Theta(\sum_{G\in\mathcal{G}}(|V(G)|+n)\cdot n^2)$. Our algorithm improves on this by a factor of n.

VI. The Deep Coalescence Cost Model

Our algorithms for efficient SPR local search for the duplication-loss model also apply, with a few changes, to the corresponding SPR local search problem for the deep coalescence model. In the following, we first introduce the deep coalescence model and then show how to modify the algorithm from the previous section. As before, we assume that $\mathit{Le}(S) = \bigcup_{G\in\mathcal{G}} \mathit{Le}(G)$.

It has been shown [31] that the same mapping ($\mathcal{M}$) that minimizes the duplication and/or loss costs also minimizes the deep coalescence cost when reconciling the gene tree and species tree. Deep coalescence (or incomplete lineage sorting) arises when ancestral gene copies fail to coalesce into a common ancestral copy at their earliest opportunity on the species tree. This is illustrated in Supplementary Figure S2. Under the deep coalescence model [1], the reconciliation cost of gene tree G and species tree S is defined in terms of the number of "extra lineages" required to fit G into S. The idea is to count the number of gene lineages present in each edge of the species tree, and if a species tree edge has more than one gene lineage passing through it, then this implies that the gene lineages have not coalesced at their earliest opportunity on the species tree (Supplementary Figure S2). More formally:

Definition 6.1 (Extra Lineages): The number of *extra lineages* $\mathit{EL}(G,S,v)$ at an edge $\{\mathit{pa}(v),v\} \in E(S')$, is defined to be $|\{x \in I(G)\colon \mathcal{M}_{G,S'}(x) \geq_{S'} \mathit{pa}(v)$ and $\exists x' \in \mathit{Ch}(x)$ such that $\mathcal{M}_{G,S'}(x') \leq_{S'} v\}| - 1$, where $S' = S[\mathit{Le}(G)]$. We define $\mathit{EL}(G,S) = \sum_{v\in V(S)\setminus\{rt(S)\}} \mathit{EL}(G,S,v)$ to be the number of extra lineages in S with respect to G.

Definition 6.2 (Reconciliation cost): Under the deep coalescence model, the *reconciliation cost* of G with S, denoted $\Delta^{dc}(G,S)$, is defined to be $\mathit{EL}(G,S)$. Correspondingly, the *reconciliation cost* from $\mathcal{G}$ to S, denoted by $\Delta^{dc}(\mathcal{G},S)$ is defined to be $\sum_{G\in\mathcal{G}} \Delta^{dc}(G,S)$.

Problem 5 (Deep-Coalescence): Given a set $\mathcal{G}$ of gene trees, the *Deep-Coalescence* problem is to find a species tree S^* comparable with $\mathcal{G}$, such that $\Delta^{dc}(\mathcal{G},S^*)$ is minimized.

We refer to the LS and RLS problems under the deep coalescence model as the *Deep-Coalescence-LS (DC-LS)* and *Deep-Coalescence-RLS (DC-RLS)* problems respectively.

As before, we will show that the DC-RLS problem can be solved in $O(\sum_{G\in\mathcal{G}}(|V(G)|+n))$ time, yielding a $O(\sum_{G\in\mathcal{G}}(|V(G)|+n)\cdot n)$ time algorithm for the DC-LS problem.

A. Solving the DC-RLS Problem

To reuse the algorithm from Section V, we explore the link between the number of extra lineages and the definition of Losses. As in previous sections, we limit our attention to one gene tree G; in particular, we show how to solve the DC-RLS problem for G in $O(|V(G)|+n)$ time. The algorithm extends trivially to solve the DC-RLS problem on the set of gene trees $\mathcal{G}$ in $O(\sum_{G\in\mathcal{G}}(|V(G)|+n))$ time. We will also assume that $\mathit{Le}(G) = \mathit{Le}(S)$. If $\mathit{Le}(G) \neq \mathit{Le}(S)$ then we can set the species tree to be $S[\mathit{Le}(G)]$; this takes $O(|\mathit{Le}(G)|+n)$

time and, consequently, does not affect the time complexity of the algorithm.

Definition 6.3 (Stretch): The *stretch* of a node $g \in I(G)$ with respect to S, denoted *Stretch*(G, S, g), is defined to be $\sum_{g' \in Ch(g)} d_S(\mathcal{M}_{G,S}(g), \mathcal{M}_{G,S}(g'))$. The stretch of G with respect to S, denoted *Stretch*(G, S) is defined to be $\sum_{g \in I(G)}$ *Stretch*(G, S, g).

Lemma 6.1: *EL*$(G, S) =$ *Stretch*$(G, S) - 2(n-1)$.

Proof: Following the definition of *EL*(G, S), we must show that:

$$\sum_{v \in V(S) \setminus \{rt(S)\}} \Big(|\{x \in I(G) \colon \mathcal{M}_{G,S}(x) \geq_S pa(v) \text{ and } \exists x' \in Ch(x) \text{ such that } \mathcal{M}_{G,S}(x') \leq_S v\}| - 1 \Big) = Stretch(G, S) - 2(n-1).$$

Observe that $|V(S) \setminus \{rt(S)\}| = 2(n-1)$, thus we must show that:

$$\sum_{v \in V(S) \setminus \{rt(S)\}} |\{x \in I(G) \colon \mathcal{M}_{G,S}(x) \geq_S pa(v) \text{ and } \exists x' \in Ch(x) \text{ such that } \mathcal{M}_{G,S}(x') \leq_S v\}| = Stretch(G, S).$$

Following the usual convention, we refer to the left hand side of this previous equation by LHS and the right hand side by RHS.

Consider the LHS. Any particular $x \in I(G)$ is counted once each at exactly those $v \in V(S) \setminus \{rt(S)\}$ for which $(pa(v), v)$ is an edge along the path between $\mathcal{M}_{G,S}(x)$ and $\mathcal{M}_{G,S}(x')$ or the path between $\mathcal{M}_{G,S}(x)$ and $\mathcal{M}_{G,S}(x'')$, where $x', x'' \in Ch(x)$. In other words, each $x \in I(G)$ is counted exactly *Stretch*(G, S, x) times. The LHS can thus be written as $\sum_{x \in I(G)}$ *Stretch*(G, S, x) which is, by definition, equal to the RHS. The lemma follows. ■

Observe the similarity between the definition of *Loss*(G, S, g) and *Stretch*(G, S, g). This is what makes it possible for us to reuse the Lemmas and Algorithm from Section V. We follow the setup and notation from Section V-A. Of the six lemmas (Lemmas 5.1 through 5.1) that characterize the behavior of the loss cost, four also correctly describe the behavior of the deep coalescence cost. These are Lemmas 5.1, 5.2, 5.3, and 5.5. The only change in these four lemmas is to replace all instances of *Loss*$(\cdots)$ with *Stretch*$(\cdots)$. The proofs of these lemmas also remain unchanged (except for a few minor attendant changes that are easily discerned). The remaining two lemmas also remain largely identical to their counterparts Lemmas 5.4 and 5.6). For completeness, these two modified lemmas appear as Lemmas S1 and S2 in the supplement.

Due to the similarity of the characterization, we can use Algorithm 2 to efficiently compute the stretch of all the trees in $SPR_S(v)$. The only change is in the way the counters are set for cases 4 and 6 (which correspond to the two modified Lemmas S1 and S2) in Section V-B.1. The updated descriptions for these two cases is as follows.

Updated Case 4: According to Lemma S1, if N_v is regrafted on an edge of N_u that is not in $N_{a'}$, then the contribution of g to the stretch is captured by incrementing counter-3 by 1 at each node except u along the path $u \rightarrow_N a'$, and at their siblings. If N_v is regrafted on an edge of N_u that is in $N_{a'}$ then there are two possible cases:

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

Updated Case 6: By Lemma S2, for this case, the change in the stretch of g is captured by incrementing counter-6 by 1 at node a', and by incrementing counter-4 by 1 each at all the grand-children (i.e, children of children) of a' in N.

The following theorem now follows immediately from Lemma 6.1 and Theorems 5.1 and 5.2.

Theorem 6.1: The DC-RLS and DC-LS problems can be solved in $O(\sum_{G \in \mathcal{G}}(|V(G)| + n))$ and $O(\sum_{G \in \mathcal{G}}(|V(G)| + n) \cdot n)$ time respectively.

VII. Related Speed-ups for Other Search Heuristics

A. Step-wise Taxon Addition Heuristic

To improve the performance of local search heuristics in phylogeny construction, the starting tree for the first local search step is typically constructed using a greedy random taxon addition procedure. This greedy procedure builds a starting species tree step-by-step by adding one taxon at a time at its locally optimal position. Our efficient algorithms for the LS problem under the duplication, duplication-loss, and deep coalescence cost models also yield a $\Theta(n)$ speed-up over naïve algorithms for this greedy procedure.

B. TBR Local Search

TBR, like SPR, is a tree edit operation. Heuristics based on the TBR local search problem significantly extend the search space explored at each local search step; however, due to inefficient running times, they have rarely been applied in practice. Our solution to the RLS problem allows us to improve the time complexity of the TBR local search problem by a factor of n.

Intuitively, a (rooted) TBR operation may be viewed as being like an SPR operation except that the TBR operation allows the pruned subtree to be arbitrarily rerooted before being regrafted. A formal definition of the TBR operation appears in the Supplement (Definition S2). Let TBR-LS denote the local search problem analogous to LS defined on the TBR neighborhood of a tree. Our goal is to solve the TBR-LS

problem. Observe that there are $\Theta(n)$ different ways to select a subtree of S to be pruned. Furthermore, there are $O(n)$ different ways to reroot the pruned subtree. In particular,

Observation 7.1: The TBR-LS problem can be solved by solving $O(n^2)$ instances of the RLS problem.

Let D-TBR-LS, DL-TBR-LS and DC-TBR-LS denote the TBR-LS problem under the duplication, duplication-loss and deep coalescence cost models respectively.

Based on Observation 7.1 and Theorems 4.1, 5.1, and 6.1, the following theorem follows immediately.

Theorem 7.1: The D-TBR-LS, DL-TBR-LS, and DC-TBR-LS problems can be solved in $O(\sum_{G\in\mathcal{G}}|V(G)|\cdot n^2)$, $O(\sum_{G\in\mathcal{G}}(|V(G)|+n)\cdot n^2)$, and $O(\sum_{G\in\mathcal{G}}(|V(G)|+n)\cdot n^2)$ times respectively.

This improves on the time complexity of the naïve solutions for these three problems by a factor of n. Note that the complexity of the D-TBR-LS problem has been further reduced to $O(\sum_{G\in\mathcal{G}}|V(G)|\cdot n\log|V(G)|)$ in [55].

VIII. Experimental Evaluation

The algorithms presented in this paper have been implemented into the software packages DupTree [47] and iGTP [48]. DupTree implements the algorithms for the duplication problem, while iGTP incorporates DupTree as well as the algorithms for the duplication-loss and deep-coalescence problems into a graphical user interface. These implementations enable GTP analyses under these cost measures for thousands of gene trees and hundreds of species. Both DupTree and iGTP also implement a taxon-addition heuristic to construct effective starting trees for the local search step, and both have the ability to handle unrooted input gene trees. The utility of gene tree parsimony in performing phylogenetic analyses has already been extensively demonstrated in the literature; see, for example, [11], [13], [14], [17], [26], [34], [35], [49]–[51]. The run time performance as well as scalability of our algorithms has also been rigorously studied and demonstrated using DupTree and iGTP [47], [48]. For instance, on a dataset with 200 taxa and 20 gene trees, iGTP required at most 5 minutes using any of the cost measures, while the fastest previously existing software, GeneTree [40], required at least 85 hours to execute the same local search heuristic. We have tested our implementations on datasets with up to 2000 taxa.

Thus, here we focus on testing the ability of our SPR local search heuristic in inferring optimal or near-optimal solutions. Several algorithms have been developed for optimally solving the GTP problem under the duplication, duplication-loss, and deep coalescence costs [19], [20], [26]. Even though such exact algorithms have exponential time complexity and are only applicable to small datasets with only a few taxa, they allow us to evaluate the accuracy of our heuristic in inferring optimal solutions. To that end, we identified five datasets that have been previously analyzed using these optimal algorithms. These include: (i) A dataset of six angiosperm species and 577 gene trees, analyzed under the duplication cost [17], (ii) a dataset of 29 eukaryotic species and 1111 gene trees, analyzed under the duplication cost [19], (iii) a dataset of 8 yeast species and 106 gene trees from [56], analyzed under the deep coalescence cost [26], (iv) a dataset of 8 *Apicomplexan* species and 268 gene trees from [57], analyzed under the deep coalescence cost [26], and (v) a dataset of 12 plant species and 6084 gene trees [20] analyzed under the duplication cost. We applied our algorithms (using iGTP) to these datasets and observed that, in each case, we obtain the optimal solution in the very first run of the heuristic, and with each run taking no more than a handful of seconds. These results demonstrate the effectiveness of the SPR local search heuristic for gene tree parsimony. Our algorithms thus make it possible to perform large-scale gene tree parsimony analyses efficiently and accurately.

IX. Conclusion

In this work, we have developed efficient local search algorithms for performing GTP analyses under the duplication, loss, duplication-loss, and deep coalescence cost models. We also discuss and clarify the existing alternative problem formulations for GTP under these costs, and elucidate the known complexity results for each GTP variant. Our algorithms make it possible to perform large-scale GTP analyses efficiently and accurately. These algorithms have been implemented into the software packages DupTree [47] and iGTP [48], and have already enabled several phylogenetic studies [49]–[51].

GTP is a parsimony approach that minimizes the number of evolutionary events that are counted and, therefore, may not perform well when the gene families under consideration exhibit high rates of duplication and loss or deep coalescence. In such cases, inferring the species tree based on a probabilistic model of gene evolution may produce more accurate results. Under the coalescent model, many methods have been developed for inferring species trees from collections of conflicting genes in a probabilistic framework. These include likelihood-based approaches like STEM [58], MT [59], STELLS [60], Bayesian approaches [61]–[64] including BEST [65], *BEAST [66], BUCKY [67] and ST-ABC [68], and also other statistics-based methods like STAR [69], STEAC [69], and iGLASS [70]. However, these method are highly computationally demanding and have only been applied to datasets with a few taxa. Similarly, probabilistic models of gene tree-species tree reconciliation that account for duplication and loss have also been developed [71]–[74]. However, due to their complexity, they have not been incorporated into any tree search methods. In contrast, the algorithms presented in this paper, make it possible to incorporate genome-scale information from thousands of genes for reconstructing species phylogenies with hundreds of taxa.

Acknowledgements

This work was supported in part by NSF grants 0334832 and 0830012. The authors thank J. Gordon Burleigh for many helpful discussions.

References

[1] W. P. Maddison, "Gene trees in species trees," *Syst. Biol.*, vol. 46, pp. 523–536, 1997.

[2] 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.

[3] O. Eulenstein, S. Huzurbazar, and D. A. Liberles, *Evolution after Gene Duplication*. John Wiley and Sons, Inc., 2010, ch. Reconciling Phylogenetic Trees, pp. 185–206.

[4] R. D. M. Page, "Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas," *Syst. Biol.*, vol. 43, no. 1, pp. 58–77, 1994.

[5] 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.

[6] B. Mirkin, I. Muchnik, and T. F. Smith, "A biologically consistent model for comparing molecular phylogenies," *J. Comput. Biol.*, vol. 2, no. 4, pp. 493–507, 1995.

[7] O. Eulenstein and M. Vingron, "On the equivalence of two tree mapping measures," *Discrete Applied Mathematics*, vol. 88, pp. 101–126, 1998.

[8] L. Zhang, "On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies," *J. Comput. Biol.*, vol. 4, no. 2, pp. 177–187, 1997.

[9] J. Slowinski and R. D. M. Page, "How should species phylogenies be inferred from sequence data?" *Syst. Biol.*, vol. 105, pp. 147–158, 1999.

[10] B. Ma, M. Li, and L. Zhang, "From gene trees to species trees," *SIAM J. Comput.*, vol. 30, no. 3, pp. 729–752, 2000.

[11] R. D. M. Page, "Extracting species trees from complex gene trees: reconciled trees and vertebrate phylogeny," *Molecular Phylogenetics and Evolution*, vol. 14, pp. 89–106, 2000.

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

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

[14] J. A. Cotton and R. D. M. Page, "Tangled tales from multiple markers: reconciling conflict between phylogenies to build molecular supertrees," in *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, O. R. P. Bininda-Emonds, Ed. Springer-Verlag, 2004, pp. 107–125.

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

[16] P. Górecki and J. Tiuryn, "Dls-trees: A model of evolutionary scenarios," *Theor. Comput. Sci.*, vol. 359, no. 1-3, pp. 378–399, 2006.

[17] M. J. Sanderson and M. M. McMahon, "Inferring angiosperm phylogeny from EST data with widespread gene duplication," *BMC Evolutionary Biology*, vol. 7, no. suppl 1:S3, 2007.

[18] M. S. Bansal, O. Eulenstein, and A. Wehe, "The gene-duplication problem: Near-linear time algorithms for NNI-based local searches," *IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 6, no. 2, pp. 221–231, 2009.

[19] J.-P. Doyon and C. Chauve, "Branch-and-bound approach for parsimonious inference of a species tree from a set of gene family trees," in *Software Tools and Algorithms for Biological Systems*, ser. Advances in Experimental Medicine and Biology, H. R. R. Arabnia and Q.-N. Tran, Eds. Springer New York, 2011, vol. 696, pp. 287–295.

[20] W.-C. Chang, G. Burleigh, D. Fernandez-Baca, and O. Eulenstein, "An ILP solution for the gene duplication problem," *BMC Bioinformatics*, vol. 12, no. Suppl 1, p. S14, 2011.

[21] R. Chaudhary, J. G. Burleigh, and O. Eulenstein, "Algorithms for rapid error correction for the gene duplication problem," in *ISBRA*, ser. Lecture Notes in Computer Science, J. Chen, J. Wang, and A. Zelikovsky, Eds., vol. 6674. Springer, 2011, pp. 227–239.

[22] M. S. Bansal and R. Shamir, "A note on the fixed parameter tractability of the gene-duplication problem," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 8, no. 3, pp. 848–850, 2011.

[23] P. Górecki, J. G. Burleigh, and O. Eulenstein, "Gtp supertrees from unrooted gene trees: Linear time algorithms for nni based local searches," in *ISBRA*, ser. Lecture Notes in Computer Science, L. G. Bleris, I. I. Mandoiu, R. Schwartz, and J. Wang, Eds., vol. 7292. Springer, 2012, pp. 102–114.

[24] W. P. Maddison and L. L. Knowles, "Inferring phylogeny despite incomplete lineage sorting," *Syst.Biol.*, vol. 55, no. 1, pp. 21–30, 2006.

[25] C. Than, R. Sugino, H. Innan, and L. Nakhleh, "Efficient inference of bacterial strain trees from genome-scale multilocus data," *Bioinformatics*, vol. 24, no. 13, pp. i123–i131, 2008.

[26] C. Than and L. Nakhleh, "Species tree inference by minimizing deep coalescences," *PLoS Computational Biology*, vol. 5, no. 9, p. e1000501, 2009.

[27] C. V. Than and L. Nakhleh, *Estimating Species Trees: Practical and Theoretical Aspects*. Wiley-VCH, 2010, ch. Inference of parsimonious species phylogenies from multi-locus data by minimizing deep coalescences, pp. 79–98.

[28] C. V. Than and N. A. Rosenberg, "Consistency properties of species tree inference by minimizing deep coalescences," *J. Comput. Biol.*, vol. 18, no. 1, pp. 1–15, 2011.

[29] H. T. Lin, J. G. Burleigh, and O. Eulenstein, "The deep coalescence consensus tree problem is pareto on clusters," in *ISBRA*, ser. Lecture Notes in Computer Science, J. Chen, J. Wang, and A. Zelikovsky, Eds., vol. 6674. Springer, 2011, pp. 172–183.

[30] Y. Yu, T. Warnow, and L. Nakhleh, "Algorithms for MDC-based multi-locus phylogeny inference: Beyond rooted binary gene trees on single alleles," *J. Comput. Biol.*, vol. 18, no. 11, pp. 1543–1559, 2011.

[31] T. Wu and L. Zhang, "Structural properties of the reconciliation space and their applications in enumerating nearly-optimal reconciliations between a gene tree and a species tree," *BMC Bioinformatics*, vol. 12, no. Suppl 9, p. S7, 2011.

[32] L. Zhang, "From gene trees to species trees ii: Species tree inference by minimizing deep coalescence events," *IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 8, no. 6, pp. 1685–1691, 2011.

[33] M. S. Bayzid and T. Warnow, "Estimating optimal species trees from incomplete gene trees under deep coalescence," *J. Comput. Biol.*, vol. 19, no. 6, pp. 591–605, 2012.

[34] 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, pp. 349–362, 1997.

[35] C. Than, R. Sugino, H. Innan, and L. Nakhleh, "Efficient inference of bacterial strain trees from genome-scale multilocus data," *Bioinformatics*, vol. 24, no. 13, pp. i123–131, 2008.

[36] C. Chauve and N. El-Mabrouk, "New perspectives on gene family evolution: Losses in reconciliation and a link with supertrees," in *RECOMB*, 2009, pp. 46–58.

[37] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *J. Algorithms*, vol. 57, no. 2, pp. 75–94, 2005.

[38] W.-C. Chang, A. Wehe, P. Gorecki, and O. Eulenstein, "Exact solutions for classic gene tree parsimony problems," in *Proceedings of the 5th International Conference on Bioinformatics and Computational Biology (BICOB)*, F. Saeed and B. DasGupta, Eds., 2013, pp. 225–230.

[39] A. Wehe, J. G. Burleigh, and O. Eulenstein, "Efficient algorithms for knowledge-enhanced supertree and supermatrix phylogenetic problems," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 99, no. PrePrints, 2012.

[40] R. D. M. Page, "GeneTree: comparing gene and species phylogenies using reconciled trees," *Bioinformatics*, vol. 14, no. 9, pp. 819–820, 1998.

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

[42] 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.

[43] 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.

[44] Y. S. Song, "On the combinatorics of rooted binary phylogenetic trees," *Annals of Combinatorics*, vol. 7, no. 3, pp. 365–379, 2003.

[45] 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 *RECOMB*, 2007, pp. 238–252.

[46] 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*, vol. 11, no. Suppl 1, p. S42, 2010.

[47] A. Wehe, M. S. Bansal, J. G. Burleigh, and O. Eulenstein, "Duptree: a program for large-scale phylogenetic analyses using gene tree parsimony," *Bioinformatics*, vol. 24, no. 13, 2008.

[48] R. Chaudhary, M. S. Bansal, A. Wehe, D. Fernandez-Baca, and O. Eulenstein, "iGTP: A software package for large-scale gene tree parsimony analysis," *BMC Bioinformatics*, vol. 11, no. 1, p. 574, 2010.

[49] J. G. Burleigh, M. S. Bansal, O. Eulenstein, S. Hartmann, A. Wehe, and T. J. Vision, "Genome-scale phylogenetics: Inferring the plant tree of life from 18,896 gene trees," *Syst. Biol.*, vol. 60, no. 2, pp. 117–125, 2011.

[50] R. W. Ness, S. W. Graham, and S. C. H. Barrett, "Reconciling gene and genome duplication events: Using multiple nuclear gene families to infer the phylogeny of the aquatic plant family pontederiaceae," *Mol. Biol. Evol.*, vol. 28, no. 11, pp. 3009–3018, 2011.
[51] L. A. Katz, J. R. Grant, L. W. Parfrey, and J. G. Burleigh, "Turning the crown upside down: Gene tree parsimony roots the eukaryotic tree of life," *Syst. Biol.*, 2012.
[52] G. Blin, P. Bonizzoni, R. Dondi, R. Rizzi, and F. Sikora, "Complexity insights of the minimum duplication problem," in *SOFSEM*, ser. Lecture Notes in Computer Science, M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, Eds., vol. 7147. Springer, 2012, pp. 153–164.
[53] C. Than, D. Ruths, and L. Nakhleh, "Phylonet: a software package for analyzing and reconstructing reticulate evolutionary relationships," *BMC Bioinformatics*, vol. 9, no. 1, p. 322, 2008.
[54] J. A. Cotton and M. Wilkinson, "Majority-rule supertrees," *Syst. Biol.*, vol. 56, no. 3, pp. 445–452, 2007.
[55] 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. Comput. Biol. Bioinform.*, vol. 5, no. 4, pp. 514–524, 2008.
[56] A. Rokas, B. L. Williams, N. King, and S. B. Carroll, "Genome-scale approaches to resolving incongruence in molecular phylogenies," *Nature*, vol. 425, pp. 798–804, 2003.
[57] C.-H. Kuo, J. P. Wares, and J. C. Kissinger, "The Apicomplexan Whole-Genome Phylogeny: An Analysis of Incongruence among Gene Trees," *Mol. Biol. Evol.*, vol. 25, no. 12, pp. 2689–2698, 2008.
[58] L. S. Kubatko, B. C. Carstens, and L. L. Knowles, "STEM: species tree estimation using maximum likelihood for gene trees under coalescence," *Bioinformatics*, vol. 25, no. 7, pp. 971–973, 2009.
[59] L. Y. Liang Liu and D. K. Pearl, "Maximum tree: a consistent estimator of the species tree," *J. Math. Biol.*, vol. 60, no. 1, pp. 95–106, 2010.
[60] Y. Wu, "Coalescent-based species tree inference from gene tree topologies under incomplete lineage sorting by maximum likelihood," *Evolution*, vol. 66, no. 3, pp. 763–775, 2012.
[61] S. V. Edwards, L. Liu, and D. K. Pearl, "High-resolution species trees without concatenation," *P. Natl. Acad. Sci. USA*, vol. 104, no. 14, pp. 5936–5941, 2007.
[62] L. Liu and D. K. Pearl, "Species Trees from Gene Trees: Reconstructing Bayesian Posterior Distributions of a Species Phylogeny Using Estimated Gene Tree Distributions," *Syst. Biol.*, vol. 56, no. 3, pp. 504–514, 2007.
[63] C. Ané, B. Larget, D. A. Baum, S. D. Smith, and A. Rokas, "Bayesian Estimation of Concordance Among Gene Trees," *Mol. Biol. Evol.*, vol. 24, no. 7, p. 1575, 2007.
[64] L. Liu, D. K. Pearl, R. T. Brumfield, and S. V. Edwards, "Estimating species trees using multiple-allele dna sequence data," *Evolution*, vol. 62, no. 8, pp. 2080–2091, 2008.
[65] L. Liu, "Best: Bayesian estimation of species trees under the coalescent model," *Bioinformatics*, vol. 24, no. 21, pp. 2542–2543, 2008.
[66] J. Heled and A. J. Drummond, "Bayesian inference of species trees from multilocus data," *Mol. Biol. Evol.*, vol. 27, no. 3, pp. 570–580, 2010.
[67] B. R. Larget, S. K. Kotha, C. N. Dewey, and C. Ané, "Bucky: Gene tree/species tree reconciliation with bayesian concordance analysis," *Bioinformatics*, vol. 26, no. 22, pp. 2910–2911, 2010.
[68] H. H. Fan and L. S. Kubatko, "Estimating species trees using approximate bayesian computation," *Molecular Phylogenetics and Evolution*, vol. 59, no. 2, pp. 354 – 363, 2011.
[69] L. Liu, L. Yu, D. K. Pearl, and S. V. Edwards, "Estimating species phylogenies using coalescence times among sequences," *Syst. Biol.*, vol. 58, no. 5, pp. 468–477, 2009.
[70] E. M. Jewett and N. A. Rosenberg, "iglass: An improvement to the glass method for estimating species trees from gene trees," *J. Comput. Biol.*, vol. 19, no. 3, pp. 293–315, 2012.
[71] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad, "Bayesian gene/species tree reconciliation and orthology analysis using mcmc," in *ISMB (Supplement of Bioinformatics)*, 2003, pp. 7–15.
[72] O. Äkerborg, B. Sennblad, L. Arvestad, and J. Lagergren, "Simultaneous Bayesian gene tree reconstruction and reconciliation analysis," *P. Natl. Acad. Sci. USA*, vol. 106, no. 14, pp. 5714–5719, 2009.
[73] P. Gorecki, G. Burleigh, and O. Eulenstein, "Maximum likelihood models and algorithms for gene tree evolution with duplications and losses," *BMC Bioinformatics*, vol. 12, no. Suppl 1, p. S15, 2011.
[74] M. D. Rasmussen and M. Kellis, "Unified modeling of gene duplication, loss, and coalescence using a locus tree," *Genome Research*, 2012.

Mukul S. Bansal is currently a postdoctoral associate in the research group of Professor Manolis Kellis at the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT). Prior to joining MIT, he was an Edmond J. Safra postdoctoral fellow at the School of Computer Science at Tel Aviv University, Israel, and worked in the research group of Professor Ron Shamir. He received the PhD degree in computer science from Iowa State University, Ames, USA, in 2009. His research interests are in computational biology and bioinformatics, with a focus on computational molecular evolution.

Oliver Eulenstein received the PhD degree in computational biology from the University of Bonn with Thomas Lengauer in 1998, and was a postdoctoral fellow with Dan Gusfield at the University of California at Davis. He is an associate professor of computer science in the Department of Computer Science, Iowa State University, Ames, USA. His research focuses on computational biology, particularly on computational phylogenetics.

SUPPLEMENT FOR "ALGORITHMS FOR GENOME-SCALE PHYLOGENETICS USING GENE TREE PARSIMONY"

Lemmas S1 and S2

Lemma S1: Let g and g' be blue nodes and g'' be a green node, $x \in V(N_u) \setminus \{u\}$, and let $a' = \mathcal{M}_{\Gamma,N}(g)$, $b' = \mathcal{M}_{\Gamma,N}(g')$ and $c' = \mathcal{M}_{\Gamma,N}(g'')$.

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

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

- Part 1(a): For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_{S'}(x)$, and $\mathcal{M}_{S'}(g'') = c'$. Also observe that the same holds for the case when $x = u$. Thus, for each x such that $a' \leq_N x <_N u$, we have $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c') - 1$. Part 1.(a) of the lemma now follows immediately.
- Part 1(b): In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = pa_N(x)$, and $\mathcal{M}_{S'}(g'') = c'$, and, $\mathcal{M}_{S''}(g) = \mathcal{M}_{S''}(g') = pa_{S''}(x)$, and $\mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c') - 1$. Hence, $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g) - 1$.
- Part 1(c): In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = \mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = lca_{S''}(x,a')$ and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g')) = 0$ and $d_{S'}(\mathcal{M}_{S'}(g), c') = d_{S''}(\mathcal{M}_{S''}(g), c')$. Thus, $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g)$.
- Part 2(a)(i): Let T and T' denote the trees $SPR_N(v,a')$ and $SPR_N(v,b'')$, respectively. Then, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$ and $\mathcal{M}_T(g'') = c'$, and, $\mathcal{M}_{T'}(g) = a'$, $\mathcal{M}_{T'}(g') = pa_{T'}(b'')$ and $\mathcal{M}_{T'}(g'') = c'$. Therefore, we must have $d_T(\mathcal{M}_T(g), c') = d_{T'}(\mathcal{M}_T(g), c') + 1$, and $d_T(\mathcal{M}_{T'}(g), \mathcal{M}_{T'}(g')) = 1$. Consequently, $\mathit{Stretch}(G,T',g) = \mathit{Stretch}(G,T,g)$.
- Part 2(a)(ii): This case is relevant only if $b' \neq b''$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, $\mathcal{M}_{S'}(g') = pa_{S'}(x)$ and $\mathcal{M}_{S''}(g') = pa_{S''}(pa(x))$. Thus, $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g')) + 1$, and consequently $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g) + 1$.
- Part 2(a)(iii): In this case, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = a'$, $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = c'$, and both $\mathcal{M}_{S'}(g')$ and $\mathcal{M}_{S''}(g')$ must be nodes along the path $b'' \rightarrow_{S''} b'$ such that $d_{S'}(a', \mathcal{M}_{S'}(g')) = d_{S''}(a', \mathcal{M}_{S''}(g'))$ (note that this is true even if $b' \leq_N pa(x) <_N a'$). The result follows.
- Part 2(a)(iv): This case exists only if $a' \neq c'$. Let T denote the tree $SPR_N(v,a')$. We must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$. Therefore, $d_{S'}(a',c') = d_T(\mathcal{M}_T(g), c') = d_N(a',c') + 1$. Thus, if $c' \leq_N x <_N a'$, then $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G, SPR_N(v,a'), g)$.
- Part 2(a)(v): Let c'' denote the sibling of b'' in tree N. Then, in this case, we must have $x \in N_{c''}$. Moreover, x is not such that $c' \leq_N x <_N a'$. Thus, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, and $\mathcal{M}_{S'}(g'') = c'$. Also, for the tree $T = SPR_N(v,a')$, we have $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$ and $d_T(\mathcal{M}_{S'}(g), c') = d_{S'}(a',c') + 1$. Hence, $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,T,g) - 1$.
- Part 2(b)(i): The proof for this part is identical to the proof of part *2.(a).iv.*
- Part 2(b)(ii): Let T denote the tree $SPR_N(v,a')$. There are two possible cases, either $a' = c'$ or $a' \neq c'$. For $a' = c'$, we must have $\mathit{Stretch}(G,T,g) = 1$ and $\mathit{Stretch}(G,S',g) = 0$. For $a' \neq c$ we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g') = a'$, $\mathcal{M}_T(g) = \mathcal{M}_T(g') = pa_T(a')$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_T(g'') = c'$; and hence $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,T,g) - 1$. Thus, part *2.(b).ii.* of the lemma holds for both cases.

■

Lemma S2: Let g be blue, g' be red, and g'' be green. Let $x \in V(N_u) \setminus \{u\}$ and $a' = \mathcal{M}_{\Gamma,N}(g)$.

1) If $S' = SPR_N(v,x)$ where $x \not<_N a'$, and $S'' = SPR_N(v, pa(x))$, then,
   a) $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g) - 1$ if $a' \leq_N x <_N u$,
   b) $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g)$ if $a' \leq_N pa(x) <_N u$ but x is not such that $a' \leq_N x <_N u$, and,
   c) $\mathit{Stretch}(G,S',g) = \mathit{Stretch}(G,S'',g) + 1$ otherwise.

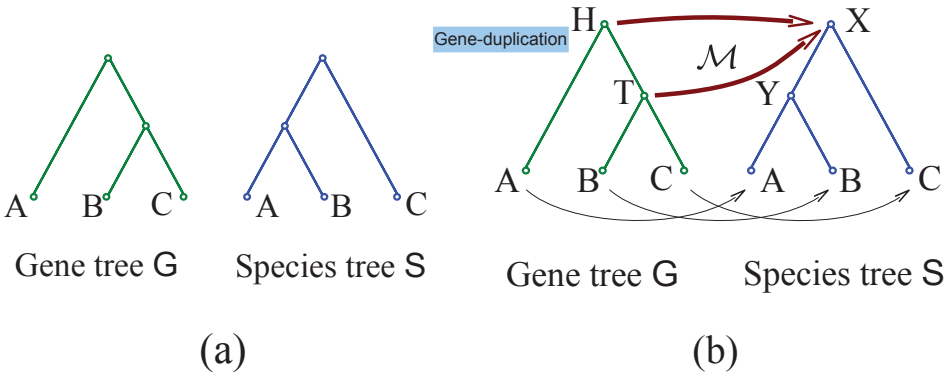

Fig. S1. **Counting duplications and losses.** (a) Input gene tree G and species tree S, each with three leaves labeled with species names A, B, and C. (b) The mapping from G to S is depicted by the arrows from nodes of G to nodes of S. The leaf-mapping is depicted by the thin black arrows, and the mapping from the internal nodes of G is depicted by the thicker brown arrows. Based on this mapping, the node H can inferred to be a duplication node (since $\mathcal{M}_{G,S}(H) = \mathcal{M}_{G,S}(T)$). Following the definition of losses, we can compute the number of losses at node T to be one and at node H to be two. The total number of losses, $Loss(G, S)$, is thus three.

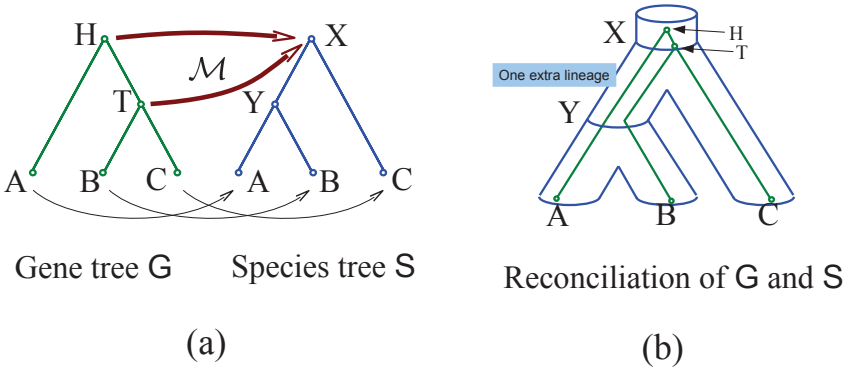

Fig. S2. **Deep coalescence and extra lineages.** (a) Input gene tree G and species tree S, along with the mapping from G to S. (b) The embedding of the gene tree G into the species tree S implied by the mapping. According to this embedding, the edge $\{X, Y\}$ of the species tree has two gene tree lineages passing through it. Thus, the number of extra lineages at edge $\{X, Y\}$, i.e., $EL(G, S, Y)$, is one. Since none of the other edges of the species tree has extra lineages, the total number of extra lineages in S, i.e., $EL(G, S)$, is one.

2) If $S' = SPR_N(v, x)$ where $x <_N a'$, and $S'' = SPR_N(v, pa(x))$, then,
   a) $Stretch(G, S', g) = Stretch(G, SPR_N(v, a'), g)$ if $x \in Ch_N(a')$, and,
   b) $Stretch(G, S', g) = Stretch(G, S'', g) + 1$ otherwise.

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

- Part 1(a): For any $a' \leq_N x <_N u$ we must have $\mathcal{M}_{S'}(g) = pa_{S'}(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$. Also observe that the same holds for the case when $x = u$. Thus, for each x such that $a' \leq_N x <_N u$, we have $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'))$ and $d_{S'}(\mathcal{M}_{S'}(g), \mathcal{M}_{S'}(g'')) = d_{S''}(\mathcal{M}_{S''}(g), \mathcal{M}_{S''}(g'')) - 1$. Part 1.(a) of the lemma follows immediately.
- Part 1(b): In this case, we must have $\mathcal{M}_{S'}(g) = pa_N(x)$, $\mathcal{M}_{S'}(g') = b$ and $\mathcal{M}_{S'}(g'') = a'$, and, $\mathcal{M}_{S''}(g) = pa_{S''}(x)$, $\mathcal{M}_{S''}(g') = b$ and $\mathcal{M}_{S''}(g'') = a'$. Therefore, $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and $d_{S'}(\mathcal{M}_{S'}(g), a') = d_{S''}(\mathcal{M}_{S''}(g), a') - 1$. Hence, $Stretch(G, S', g) = Stretch(G, S'', g)$.
- Part 1(c): In this case, $\mathcal{M}_{S'}(g) = \mathcal{M}_{S''}(g) = lca_{S''}(b, a')$, $\mathcal{M}_{S'}(g') = \mathcal{M}_{S''}(g') = b$, and $\mathcal{M}_{S'}(g'') = \mathcal{M}_{S''}(g'') = a'$. Now since b is a node in the pruned subtree N_v, we must have $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $Stretch(G, S', g) = Stretch(G, S'', g) + 1$.
- Part 2(a): If $x \in Ch_N(a')$ then we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Thus, $Stretch(G, S', g) = d_{S'}(a', b)$. Now, let T denote the tree $SPR_N(v, a')$, then we must have $\mathcal{M}_T(g) = pa_T(a')$, $\mathcal{M}_T(g'') = a'$ and $\mathcal{M}_T(g') = b$. Thus, $Stretch(G, T, g) = d_T(pa_T(a'), b) + 1$. Finally, observe that $d_{S'}(a', b) = d_T(pa_T(a'), b) + 1$, and hence, $Stretch(G, S', g) = Stretch(G, T, g)$.
- Part 2(b): For any $x <_N a'$, we must have $\mathcal{M}_{S'}(g) = \mathcal{M}_{S'}(g'') = a'$ and $\mathcal{M}_{S'}(g') = b$. Since b is a node in the pruned subtree N_v and in this case $x <_N y$ for $y \in Ch(a')$, we must have $d_{S'}(\mathcal{M}_{S'}(g), b) = d_{S''}(\mathcal{M}_{S''}(g), b) + 1$ and, consequently, $Stretch(G, S', g) = Stretch(G, S'', g) + 1$.

■

Definition of TBR operation

In order to define a TBR operation formally, we need the following definition.

Definition S1 (RR operation): Let T be a tree and $x \in V(T)$. $RR(T, x)$ is defined to be the tree T, if $x = rt(T)$. Otherwise, $RR(T, x)$ is the tree that is obtained from T by (i) suppressing $rt(T)$, and (ii) subdividing the edge $\{pa(x), x\}$ by a new root node.

Definition S2 (TBR operation): For technical reasons we first define for a tree T the *planted tree* $\Phi(T)$ to be the tree obtained by adding an additional edge, called *root edge*, $\{p, rt(T)\}$ to T.

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

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