This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

DTL-RnB: Algorithms and Tools for Summarizing the Space of DTL Reconciliations

W. Ma* D. Smirnov† J. Forman* A. Schweickart* C. Slocum‡ S. Srinivasan* R. Libeskind-Hadas*
*Harvey Mudd College, Claremont, California, USA †Pomona College, Claremont, California, USA
‡California Polytechnic University, Pomona, California, USA

***Abstract*—Phylogenetic tree reconciliation is an important technique for reconstructing the evolutionary histories of species and genes and other dependent entities. Reconciliation is typically performed in a maximum parsimony framework and the number of optimal reconciliations can grow exponentially with the size of the trees, making it difficult to understand the solution space. This paper demonstrates how a small number of reconciliations can be found that collectively contain the most highly supported events in the solution space. While we show that the formal problem is NP-complete, we give a $1 - \frac{1}{e}$ approximation algorithm, experimental results that indicate its effectiveness, and the new DTL-RnB software tool that uses our algorithms to summarize the space of optimal reconciliations (`www.cs.hmc.edu/dtlrnb`).**

I. Introduction

Phylogenetic tree reconciliation is a fundamental technique for studying the evolution of pairs of entities such as gene families and species, parasites and their hosts, and species and their geographical habitats. The reconciliation problem takes as input two trees and the associations between their leaves and seeks to find a mapping between the trees that accounts for their incongruence. In the widely-used DTL model, four types of events are considered: *speciation*, *duplication*, *transfer*, and *loss* [1, 2, 4, 5, 6, 7, 19]. Henceforth, we denote the two trees as the *species tree* (S) and the *gene tree* (G), although these trees could be host and species trees or area cladograms and species trees in the contexts of cophylogenetic and biogeographical studies, respectively.

Reconciliation in the DTL model is typically performed using a maximum parsimony formulation where each event type has an assigned cost and the objective is to find a reconciliation of minimum total cost. Figure 1(a) shows a small example of a species and gene tree and their leaf associations. Figures 1(b) and (c) show two different reconciliations of these trees with labels on the events. Speciation is generally considered a "null event" and given cost 0 while the other event types are given positive costs. For example if duplication, transfer, and loss each have cost 1, then the reconciliation in Figure 1(b) is optimal and incurs one speciation and one transfer, with total cost of 1. However, if duplication and loss have cost 1 and transfer has cost greater than 4, then the reconciliation in Figure 1(c) is optimal, incurring one speciation, one duplication, and three losses, with total cost of 4. Henceforth, we use the terms *optimal* and *maximum parsimony* interchangeably.

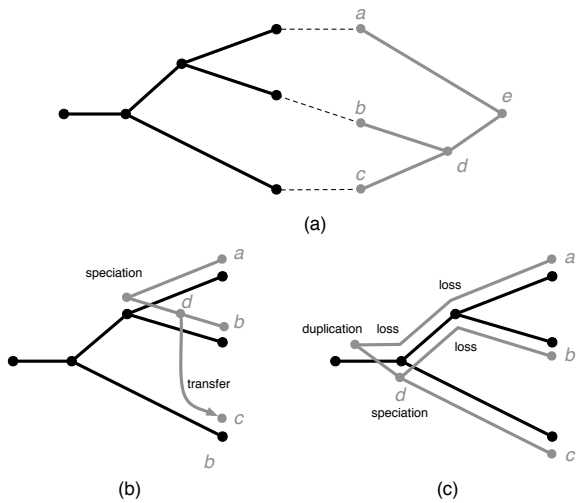


Fig. 1: (a) A species tree in black and a gene tree in gray with the leaf mapping shown with dotted lines. Two different reconciliations that are optimal for different event costs are shown in (b) and (c).

A species tree is said to be *dated* if the relative times of its internal nodes are known. For dated species trees, maximum parsimony reconciliations can be found in polynomial time [7, 10, 20]. However, accurately dating species trees is generally difficult [15], and estimated dates may be unreliable. Thus, much of the literature on DTL reconciliation assumes that the species tree is *undated*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

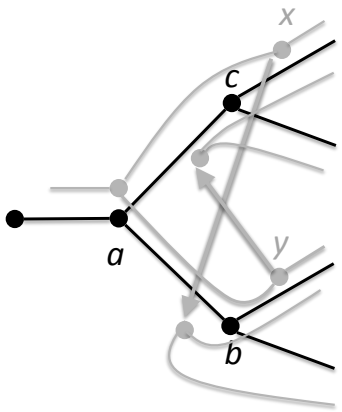


Fig. 2: A fragment of a temporally infeasible reconciliation for a species tree (black) and gene tree (gray). Node x transfers one child to the species edge from a to b, implying that x must occur before b and thus that c occurs before b. Node y transfers one child to the species edge from a to c, meaning that y must occur before c and thus that b occurs before c, contradicting the constraint that c occurs before b.

For the undated case, maximum parsimony reconciliations can be found in polynomial time [1, 18, 20] but these reconciliations may, in some cases, be *temporally infeasible* in the sense that there exists no ordering of the internal nodes that is consistent with the reconciliation. An example of a temporally infeasible reconciliation is illustrated in Figure 2. Temporal infeasibility can be detected in polynomial time [18] but the problem of finding temporally feasible maximum parsimony reconciliations is NP-complete [13, 19].

In general, the number of maximum parsimony reconciliations can grow exponentially with the size of the species and gene trees [17]. For example, Bansal et al. [2] examined a species tree with 100 primarily prokaryotic leaves and a corpus of over 4700 gene trees with median and average leaf-set sizes of 18 and 35.1, respectively.[1] Even for these relatively small trees, more than half of the cases had over 100 optimal reconciliations, 15% had over $10,000$ optimal reconciliations, and some had over 10^{14} optimal reconciliations.

Consequently, a number of efforts have been made to identify highly supported events and summarize the space of maximum parsimony reconciliations. For dated trees, Scornavacca et al. [16] defined a compact representation of the space of all maximum parsimony reconciliations called *reconciliation graphs*. Nguyen et al. [12] showed how reconciliation graphs can be used to efficiently compute a single "median reconcilation" that represents the entire space of maximum parsimony reconciliations. For undated trees, Bansal et al. [2] showed how the space of optimal reconciliations can be efficiently sampled uniformly to select representative reconciliations and identify frequently recurring events. Libeskind-Hadas et al. [11] used the notion of Pareto-optimal reconciliations to identify individual events that are highly supported across a range of different event costs in undated trees.

While these recent efforts have contributed to a better understanding of the very large space of maximum parsimony reconciliations, they provide only a partial view of this space. Reconciliation graphs have important theoretical properties but do not provide an intuitive understanding of the space of reconciliations. A median reconciliation is a useful summary as a single reconciliation, but any single reconciliation may exclude many highly-supported events. Moreover, these results have been restricted to dated trees. For undated trees, the sampling methods may, inherently, miss some reconciliations containing highly-supported events while the Pareto-optimal approach only identifies highly-supported individual events and not whole reconciliations.

In contrast, we address the problem of finding a relatively small number of maximum parsimony reconciliations that best represent the most highly-supported events found in the space of all maximum parsimony reconciliations in undated trees. Specifically, given a function that computes a "support" score for each event found in the space of maximum parsimony reconciliations and a user-specified parameter k, we wish to find k maximum parsimony reconciliations that maximize the sum of these scores.

Our contributions in this paper are the following:

1) We formally define the *k-Reconciliations Cover Problem (kRCP)* of finding a set of k reconciliations that best cover the maximum parsimony reconciliations for undated trees.
2) While we show that kRCP is NP-complete, we give a practical polynomial time approximation algorithm with approximation ratio $1 - \frac{1}{e}$.
3) We provide experimental results on a diverse dataset of 4848 genes from the Tree of Life [6]. These results show that surprisingly few reconciliations are needed to cover the events even in large solution

[1] Duplication, transfer, and loss costs were set to 2, 3, and 1, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

spaces. In this dataset, $k \leq 15$ reconciliations sufficed to cover *all* of the events in over 98% of the instances even though more than 15% of the instances had between 10^4 and 10^{39} maximum parsimony reconciliations.

4) We have implemented this algorithm in the publicly available *DTL-RnB (DTL Reconciliation Browser)* web-based tool that allows researchers to upload their own datasets and browse the reconciliations that approximate the best coverage of the large space of optimal solutions.

II. Preliminaries

For reconciliations, we follow the definitions and notation from Bansal [1]. Given a rooted tree T, we denote its node, edge, and leaf sets by $V(T)$, $E(T)$, and $Le(T)$, respectively. The set of internal nodes is denoted $I(T)$, the root node of T is denoted by $rt(T)$, the parent of a node $v \in V(T)$ by $pa_T(v)$, its set of children by $Ch_T(v)$, and the (maximal) subtree of T rooted at v by $T(v)$. 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 partial order $\geq_T$ is defined analogously, i.e., $x \geq_T y$ if x is a node on the path between $rt(T)$ and y. We say that x and y are *incomparable* if neither $x \leq_T y$ nor $y \leq_T x$. Given two nodes $x, y \in V(T)$, we denote by $lca_T(x, y)$ the least common ancestor (LCA) of x and y in tree T; that is, $lca_T(x, y)$ is the unique smallest upper bound of x and y under $\leq_T$. Given $x, y \in V(G)$, $x \rightarrow_T y$ denotes the unique path from x to y in G. We denote by $d_T(x, y)$ the number of edges on the path $x \rightarrow_T y$.

We assume that the two input trees are denoted by G (e.g., gene tree) and S (e.g., species tree), where the goal is to map tree G to tree S. Each leaf of tree G is labeled with the leaf-label from S with which it is associated. This labeling defines a *leaf-mapping* $\mathcal{L}\colon Le(G) \rightarrow Le(S)$ that maps a leaf node $g \in Le(G)$ to that unique leaf node $s \in Le(S)$ which has the same label as g. Note that tree G may have more than one leaf associated with the same leaf of S. Throughout this work we assume that $\mathcal{L}(g)$ is well-defined for each $g \in Le(G)$.

A. Reconciliation and DTL-scenarios

Next, we define a *Duplication-Transfer-Loss scenario (DTL-scenario)* [1, 19] for G and S that formally characterizes a mapping of G into S that corresponds to a biologically valid reconciliation. A DTL-scenario extends the leaf-mapping to map every node of G to a unique node in S in a way that respects the immediate temporal constraints implied by the topology of S, and designates each internal node of G as representing either a speciation, duplication, or transfer event.

The following three definitions in this subsection are from [1].

Definition II.1 (DTL-scenario)**.** *A* DTL-scenario *for G and S is a seven-tuple $(\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$, where $\mathcal{L}\colon Le(G) \rightarrow Le(S)$ represents the leaf-mapping from G to S, $\mathcal{M}\colon V(G) \rightarrow V(S)$ maps each node of G to a node of S, the sets Σ, Δ, and Θ partition $I(G)$ into speciation, duplication, and transfer nodes respectively, Ξ is a subset of edges of G that represent transfer edges, and $\tau\colon \Theta \rightarrow V(S)$ specifies the recipient (or "landing site") for each transfer event, subject to the following constraints:*

1) *If $g \in Le(G)$, then $\mathcal{M}(g) = \mathcal{L}(g)$.*
2) *If $g \in I(G)$ and g' and g'' denote the children of g, then,*
   a) *$\mathcal{M}(g) \not\leq_S \mathcal{M}(g')$ and $\mathcal{M}(g) \not\leq_S \mathcal{M}(g'')$,*
   b) *At least one of $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ is a descendant of $\mathcal{M}(g)$.*
3) *Given any edge $(g, g') \in E(G)$, $(g, g') \in \Xi$ if and only if $\mathcal{M}(g)$ and $\mathcal{M}(g')$ are incomparable.*
4) *If $g \in I(G)$ and g' and g'' denote the children of g, then,*
   a) *$g \in \Sigma$ only if $\mathcal{M}(g) = lca_S(\mathcal{M}(g'), \mathcal{M}(g''))$ and $\mathcal{M}(g')$ and $\mathcal{M}(g'')$ are incomparable,*
   b) *$g \in \Delta$ only if $\mathcal{M}(g) \geq_S lca_S(\mathcal{M}(g'), \mathcal{M}(g''))$,*
   c) *$g \in \Theta$ if and only if either $(g, g') \in \Xi$ or $(g, g'') \in \Xi$.*
   d) *If $g \in \Theta$ and $(g, g') \in \Xi$, then $\mathcal{M}(g)$ and $\tau(g)$ must be incomparable, and $\mathcal{M}(g')$ must be a descendant of $\tau(g)$, i.e., $\mathcal{M}(g') \leq_S \tau(g)$.*

Constraint 1 ensures that the mapping $\mathcal{M}$ is consistent with the leaf-mapping $\mathcal{L}$. Constraint 2a imposes on $\mathcal{M}$ the temporal constraints implied by S. Constraint 2b implies that any internal node in G may represent at most one transfer event. Constraint 3 determines the edges of G that are transfer edges. Constraints 4a, 4b, and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

4c state the conditions under which an internal node of G may represent a speciation, duplication, and transfer, respectively. Constraint 4d specifies which species may be designated as the recipient species for any given transfer event.

DTL-scenarios correspond naturally to reconciliations, and it is straightforward to infer the reconciliation of G and S implied by any DTL-scenario. However, as noted earlier, the resulting reconciliation is not guaranteed to be temporally feasible.

Given a DTL-scenario, one can directly count the minimum number of gene losses [1] in the corresponding reconciliation as follows.

Definition II.2 (Losses)**.** *Given a DTL-scenario* $\alpha = (\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$ *for* G *and* S*, let* $g \in V(G)$ *and* $\{g', g''\} = Ch_G(g)$*. The number of* losses $Loss_\alpha(g)$ *at node* g*, is defined to be:*

- $(d_S(\mathcal{M}(g), \mathcal{M}(g')) - 1) + (d_S(\mathcal{M}(g), \mathcal{M}(g'')) - 1)$*, if* $g \in \Sigma$*,*
- $d_S(\mathcal{M}(g), \mathcal{M}(g')) + d_S(\mathcal{M}(g), \mathcal{M}(g''))$*, if* $g \in \Delta$*, and*
- $d_S(\mathcal{M}(g), \mathcal{M}(g'')) + d_S(\tau(g), \mathcal{M}(g'))$ *if* $(g, g') \in \Xi$*.*

The total number of losses in the reconciliation corresponding to the DTL-scenario α *is defined to be* $Loss_\alpha = \sum_{g \in I(G)} Loss_\alpha(g)$*.*

We assume that speciations have zero cost and let C_Δ, C_Θ, and C_Λ denote the assigned positive costs for duplication, transfer, and loss events, respectively. The cost of reconciling G and S according to a DTL-scenario α is defined as follows:

Definition II.3 (Reconciliation cost of a DTL-scenario)**.** *Given a DTL-scenario* $\alpha = (\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$ *for* G *and* S*, the* reconciliation cost *associated with* α *is given by* $C_\Delta \cdot |\Delta| + C_\Theta \cdot |\Theta| + C_\Lambda \cdot Loss_\alpha$*.*

B. Maximum Parsimony Reconciliations

An instance of the maximum parsimony reconciliation problem comprises a gene tree G, a species tree S, a leaf mapping $\mathcal{L}$: $Le(G) \rightarrow Le(S)$, and positive costs C_Δ, C_Θ, and C_Λ for duplication, transfer, and loss events, respectively. A maximum parsimony reconciliation, henceforth denoted *MPR*, is a DTL-reconciliation of minimum total cost with respect to the given set of event costs.

A number of closely-related dynamic programming algorithms have been given for finding MPRs in undated trees [1, 19, 20]. Here, we use the U-MPR Algorithm from Bansal *et. al* [1] which runs in time $O(|G||S|)$. This algorithm computes a table $c(g, s)$ that gives the cost of an optimal reconciliation for the subtree of G rooted at g with the subtree of S rooted at s, including reconciliations that involve duplications on and transfers from the edge entering s. Thus, the cost of a MPR is $\min_{s \in V(S)} c(rt(G), s)$.

Standard dynamic programming "bookkeeping" methods can be used to associate an annotation with each entry $c(g, s)$ comprising the set of events that are found in all optimal solutions for the subproblem of mapping g onto s. Recording these annotations in a list does not effect the $O(|G||S|)$ running time of the U-MPR algorithm since the algorithm considers these events as it computes the matrix c. Let $\{g', g''\} = Ch_G(g)$ and let $\{s', s''\} = Ch_S(s)$, denoting the children of g and s, respectively. Then, let $c(g, s).events$ denote the set of events associated with $c(g, s)$ where each event is a tuple of the following type:

- $(\mathbb{S}_{(g,s)}, \{(g', s'), (g'', s'')\})$ if the event is a speciation in which g' is associated with s' and g'' is associated with s'';
- $(\mathbb{D}_{(g,s)}, \{(g', s), (g'', s)\})$ if the event is a duplication;
- $(\mathbb{T}_{(g,s)}, \{(g', s), (g'', \hat{s})\})$ if the event is a transfer in which g' remains on s and g'' transfers to $\hat{s}$ (s and $\hat{s}$ are called the *take-off* and *landing* sites of the transfer, respectively);
- $(\mathbb{L}_{(g,s)}, \{(g, s')\})$ if the event is a loss in which g continues onto s';
- $(\mathbb{C}_{(g,s)}, \varnothing)$ if the event is a leaf association (i.e., a contemporary association) of (g, s), and $\mathcal{L}(g) = s$.

For an event e, represented as a tuple as indicated above, let $e.type$ denote its first element, namely the event type, and let $e.associations$ denote its second element, namely the set of associations. Note that if e is a speciation, duplication, or transfer event, then $e.associations$ is a set containing two ordered pairs, each representing an association between a gene tree node and a species tree node. If e is a loss event, then $e.associations$ is a set containing one such ordered pair. If e is a leaf association, then $e.associations$ is the empty set. We refer to the matrix c along with these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

event annotations as an *annotated MPR matrix*.

C. Undated Reconciliation Graphs

Scornavacca *et. al* [16] described a technique for compactly representing the space of all MPRs for dated species trees. We adapt this technique for undated trees. The motivating principle is to encode the events and associations stored in the annotated MPR matrix using a directed graph. The graph contains a *mapping node* for each (g, s) pair associating a gene $g \in V(G)$ to a species $s \in V(S)$ that appears in some MPR *and* an *event node* for each event in $c(g, s).events$. The representation is compact by merit of the fact that while the mapping (g, s) and its events may arise in many different MPRs, they can be shared in this graph representation and thus only need to appear there once. This structure is analogous to the one first proposed in [16], but is constructed using the annotated MPR matrix from the dynamic program for undated trees rather than from the dated ones used in the original formulation.

Definition II.4 (Undated Reconciliation Graph)**.** *Given an instance of the maximum parsimony reconciliation problem $(G, S, \mathcal{L}, C_\Delta, C_\Theta, C_\Lambda)$ and an annotated maximum parsimony matrix c, we construct a directed* undated reconciliation graph $\mathcal{G} = (V_m \dot\cup V_e, E)$ *(where $\dot\cup$ represents disjoint union) with the following properties: For each $g \in V(G)$ and $s \in V(S)$ such that g is associated with s in some MPR:*

1) *There is a* mapping node $(g, s) \in V_m$,
2) *For each event $e \in c(g, s).events$, there is an* event node $e \in V_e$ *and directed edge $((g, s), e)$.*
3) *For each event node $e \in V_e$, there is an edge to each mapping node corresponding to an association in $e.associations$.*

A formal description of the algorithm for constructing an undated reconciliation graph and a derivation of its $O(|G||S|^2)$ running time are given in section A1 of the appendix.

For a directed edge (u, v) in a directed acyclic graph (DAG), we say that u is the *parent* of v and v is the *child* of u. We say that a node with indegree 0 is a *root* of the graph (there can be multiple roots) and that a node with outdegree 0 is a *leaf*. For a DAG $\mathcal{G}$, let *rt*$(\mathcal{G})$ and *Le*$(\mathcal{G})$ denote the set of roots and leaves of $\mathcal{G}$, respectively.

Given an undated reconciliation graph, we may associate a non-negative real valued *score* with each event node denoted $\sigma : V_e \to \mathbb{R}_{\geq 0}$. For example, relevant scoring functions include uniform scoring (a score of 1 for each event) and the frequency of the event over all MPRs. We discuss these and other scoring functions in more detail in Sections 4 and 5.

Next, we define undated reconciliation trees which correspond to MPRs. (Undated reconciliation trees should not be confused with the gene and species trees. Instead, they are subgraphs of the undated reconciliation graph that are topological trees.)

Definition II.5 (Undated Reconciliation Tree)**.** *Given an undated reconciliation graph $\mathcal{G} = (V_m \dot\cup V_e, E)$ for gene tree G and species tree S, an* undated reconciliation tree *is a subgraph of $\mathcal{G}$ constructed as follows:*

1) *The root of the tree is a root of $\mathcal{G}$ and thus, necessarily, one of the form $(rt(G), \cdot)$ corresponding to the association of the root of G with some node in S.*
2) *Each non-leaf mapping node added to the tree has exactly one of its event node children added to the tree.*
3) *Each event node added to the tree has all of its mapping node children added to the tree.*

There is a straightforward bijection between the set of undated reconciliation trees and the set of MPRs. This bijection is shown constructively via conversion algorithms in Appendix A2.

In addition, we note that undated reconciliation trees are acyclic, a result that will be needed in our algorithms in the next section.

Lemma II.1. *An undated reconciliation graph is acyclic.*

Proof. Every path in an undated reconciliation graph alternates between mapping nodes and event nodes. By construction, two successive mapping nodes (g, s) and (g', s') on such a path require that either g' is a child of g in G (in the case of speciation, duplication, and transfer events) or that $g = g'$ and s' is a child of s in S (in the case of loss events). This relationship induces a partial ordering on the mapping nodes and thus there cannot exist a cycle in the graph. □

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

III. The k-Reconciliations Cover Problem

Since the number of MPRs can grow exponentially with the sizes of the trees [17], our objective is to provide a summary of the most representative reconciliations. In this spirit, Nguyen et al. [12] gave an algorithm for finding a single "median reconciliation" in a dated tree. However, any single MPR may, inherently, exclude many highly-supported events that are found in other MPRs. Thus, we seek to provide a user-specified number of MPRs that best cover the events in the potentially large MPR solution space.

To that end, given a positive integer k of desired MPRs and a scoring function associating a score (e.g., event frequency) with each event, we seek to find k MPRs that maximize the sum of the scores of the events that they contain. We call this the *k-Reconciliations Cover Problem (kRCP)* and, for computational reasons, use undated reconciliation trees as proxies for their corresponding MPRs.

Definition III.1 (k-Reconciliations Cover Problem (kRCP))**.** *Given an undated reconciliation graph $\mathcal{G} = (V_m \dot{\cup} V_e, E)$ with scoring function $\sigma : V_e \to \mathbb{R}_{\geq 0}$ and a positive integer k, find a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of k undated reconciliation trees in $\mathcal{G}$ that maximizes the quantity $\sum_{v \in V_e(\mathcal{T})} \sigma(v)$ where $V_e(\mathcal{T}) = \bigcup_{1 \leq i \leq k} V_e(T_i)$.*

A. NP-completeness Result

Next, we show that an abstraction of kRCP is NP-complete. Specifically, we define graphs called *R-graphs* and *R-trees* that capture the structure of undated reconciliation graphs and reconciliation trees, respectively. We then show that the analog of kRCP using R-graphs and R-trees, in lieu of undated reconciliation graphs and reconciliation trees, is NP-complete.

Recall that the roots of undated reconciliation graphs are mapping nodes, the leaves are event nodes representing leaf associations, each event node has indegree 1 and outdegree either 0 (for leaf associations), 1 (for losses), or 2 (for speciations, duplications, and transfer), and every edge has one endpoint that is an event node and the other that is a mapping node.

Definition III.2 (R-graph)**.** *An* R-graph *is a finite simple directed acyclic graph $\mathcal{G} = (V_m \dot{\cup} V_e, E)$ such that the roots are in V_m, the leaves are in V_e, every node in V_e has indegree 1 and outdegree 0, 1, or 2, and every edge has one endpoint in V_m and the other in V_e.*

Next, we define an R-tree corresponding to an undated reconciliation tree.

Definition III.3 (R-tree)**.** *An* R-tree *T with respect to an R-graph $\mathcal{G} = (V_m \dot{\cup} V_e, E)$ is a directed rooted subtree of $\mathcal{G}$ satisfying the following properties:*

- *$rt(\mathcal{G}) \cap V(T) = \{rt(T)\}$ and $Le(T) = Le(\mathcal{G})$;*
- *If $v \in V_m \cap V(T)$ then exactly one child of v is in T;*
- *If $v \in V_e \cap V(T)$ then every child of v in $\mathcal{G}$ is in T.*

Definition III.4 (k-R-graph Cover Decision Problem (kRGCDP))**.** *Given a R-graph $\mathcal{G} = (V_m \dot{\cup} V_e, E)$ with scoring function $\sigma : V_e \to \mathbb{R}_{\geq 0}$, a positive integer k, and a real number ω, does there exist a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of k R-trees in $\mathcal{G}$ such that $\sum_{v \in V_e(\mathcal{T})} \sigma(v) \geq \omega$ where $V_e(\mathcal{T}) = \bigcup_{1 \leq i \leq k} V_e(T_i)$.*

Theorem III.1. *kRGCDP is NP-complete.*

The proof is given in Appendix A3.

B. An Approximation Algorithm for kRCP

The kRCP is a special case of the much more general *Weighted Maximum Coverage Problem* defined as follows [9]:

Problem III.1. Weighted Maximum Coverage Problem (WMCP) *Given a universe set $U = \{u_1, \ldots, u_n\}$ with a weight function $w : U \to \mathbb{R}_{\geq 0}$, a collection of sets $S = \{S_1, \ldots, S_m\}$ over the given universe, and a positive integer $k \leq m$, find a subset $S' \subseteq S$ such that $|S'| = k$ and $\sum_{u \in \bigcup_{S_i \in S'} S_i} w(u)$ is maximized.*

In the case of kRCP, the universe is the set of events, the weight function is the scoring function, and the sets are the reconciliation trees. While WMCP is NP-complete, a simple polynomial-time approximation algorithm computes solutions for WMCP that are, at worst, within a factor of $1 - \frac{1}{e}$ of optimal [9]. The algorithm chooses the set with the largest total weight, removes that set from the collection and removes the elements in that set from all remaining sets in the collection, and repeats until k sets have been selected.

However, in the case of kRCP, we cannot enumerate all of the sets since there can be exponentially many MPRs. Instead, we adapt the simple dynamic programming

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

algorithm from [16] to efficiently find the reconciliation with the greatest total score. Specifically, we begin by traversing the undated reconciliation graph in postorder – that is, first the leaves, then their parents, and so forth towards the roots. This is possible because, by Lemma II.1, the undated reconciliation graph is acyclic. We compute a *total score* τ for each node v in the reconciliation graph as follows:

- If v is a leaf then its total score is simply its score, $\sigma(v)$;
- If v is a non-leaf event node, then $\tau(v)$ is the sum of the τ values of its children;
- If v is a mapping node, then $\tau(v)$ is the maximum of the τ values of its children.

Once the total scores have been computed, the reconciliation tree with maximum total score can be found by starting at the root node with maximum total score in the reconciliation graph and traversing the graph using the events that constitute that maximum total score. Finally, the scores of those events are set to zero so that those events do not contribute to the total scores of reconciliations chosen in subsequent iterations. Note that while each event contributes its score to only one reconciliation, an event may (and often will) occur in several of the reconciliations found by this algorithm because that event is necessary in the construction of other high scoring reconciliations.

In summary, given an instance of the maximum parsimony reconciliation problem comprising gene tree G, species tree S, leaf mapping $\mathcal{L} : Le(G) \rightarrow Le(S)$, DTL costs C_Δ, C_Θ, and C_Λ, scoring function σ, and a desired number of reconciliations k, our kRCP approximation algorithm works as follows:

Algorithm 1: kRCP

Input: Gene tree G, Species tree S, leaf mapping $\mathcal{L} : Le(G) \rightarrow Le(S)$, DTL costs C_Δ, C_Θ, and C_Λ, scoring function σ, and a desired number of reconciliations k

Output: A set of k reconciliation trees that gives a $1 - \frac{1}{e}$ approximation to the k-Reconciliations Cover Problem

```
1 ReconciliationCover = ∅
2 Compute the annotated MPR matrix c using the U-MPR Algorithm [1]
3 Compute the undated reconciliation graph as described in Appendix A1
4 for k times do
5     Compute total score τ for the reconciliation graph
6     Find the reconciliation tree T of maximum total score and set the event scores in T to 0
7     Add T to ReconciliationCover
8 return ReconciliationCover
```

Theorem III.2. *The worst-case running time of the kRCP Algorithm is* $O(k|G||S|^2)$.

Proof. Line 2 takes time $O(|G||S|)$, line 3 takes time $O(|G||S|^2)$ and creates a reconciliation graph with $O(|G||S|^2)$ nodes and $O(|G||S|^2)$ edges. Each iteration of the loop at line 4 takes time linear in the size of the reconciliation graph. Thus, the total running time is $O(k|G||S|^2)$. □

We note that this running time does not include the cost of computing the scores for the events because this depends entirely on the choice of scoring function. Scoring functions are addressed in the next section.

C. Temporal Feasibility

Finally, since we assume that the species trees are not dated, the reconciliations that are returned by the kRCP algorithm may be temporally infeasible. We test each reconciliation for temporal feasibility using an algorithm similar to one described by Tofigh [18].[2]

Specifically, given gene tree G, species tree S, and a DTL-scenario found by the kRCP algorithm we construct a directed graph $F = (V, E)$, called a *temporal feasibility graph*, as follows:

1) $V = I(S) \cup I(G) \cup \{\ell\}$ where $I(S)$ and $I(G)$ represent the internal nodes of S and G, respectively, and ℓ is a single node representing all of the leaves of S and G.
2) For each pair of nodes $u, v \in V$ such that u is the parent of v in either S or G, there is a directed edge $(u, v) \in E$.
3) For each $v \in V$ such that its corresponding node in S or G is the parent of a leaf, there is an edge $(v, \ell) \in E$.
4) For each gene node g associated with species node s in the DTL-scenario:
   a) If the association is via a speciation event, g and s are identified (i.e., g is removed from the graph and all edges entering g are redirected to enter s and all nodes leaving g now leave s).

[2]One difference between our test and the one in [18] is that we test that the reconciliation is temporally consistent given the takeoff and landing sites for each transfer event. The test in [18] does not specify the landing sites of transfer events and thus may determine that the scenario is feasible by moving landing sites to locations that are not consistent with any MPR.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

b) If the association is via a duplication event, we add the directed edge $(pa_S(s), g)$ (unless $s = rt(S)$) and edge (g, s).
c) If the association is via a transfer event with landing site s', we add the directed edges $(pa_S(s), g), (g, s), (pa_S(s'), g), (g, s')$.

A directed edge (u, v) represents the constraint that node u must have a date that comes before the date of v. Thus, the edges in 1–3 above enforce that ancestor nodes must have dates that come before their descendants. The edges in 4 enforce the relative dates of genes and the species with which they are associated. In particular, 4(c) ensures that the dates of takeoff and landing sites for transfers are contemporaneous. It is easily verified that there exists a dating of the tree in which all events are temporally consistent if and only if the temporal feasibility graph is acyclic. Moreover, if the graph is acyclic, a topological ordering of that graph gives a feasible dating for the species tree in the given MPR.

IV. Software Tool and Experimental Results

The kRCP Algorithm has been implemented in the DTL-RnB (Reconciliation Browser) software tool which is implemented in Python and available for download as well as a web-based application (www.cs.hmc.edu/dtlrnb).

A. Scoring Functions

DTL-RnB currently supports three different event scoring functions described below. We note that the DTL-RnB source code is publicly available and other scoring functions can be easily added there.

Uniform: Each event in the reconciliation graph has a score of 1. Thus, the kRCP problem seeks to find k reconciliations that include as many events as possible in the space of MPRs.

Frequency: Each event in the reconciliation graph is scored by its frequency, namely the number of MPRs that contain that event divided by the total number of MPRs. The total number of MPRs can be computed by algorithm U-MPR without impacting its asymptotic running time of $O(|G||S|)$[1]. The frequencies of individual events can then be computed by a single preorder traversal of the reconciliation graph, and thus in time $O(|G||S|^2)$.

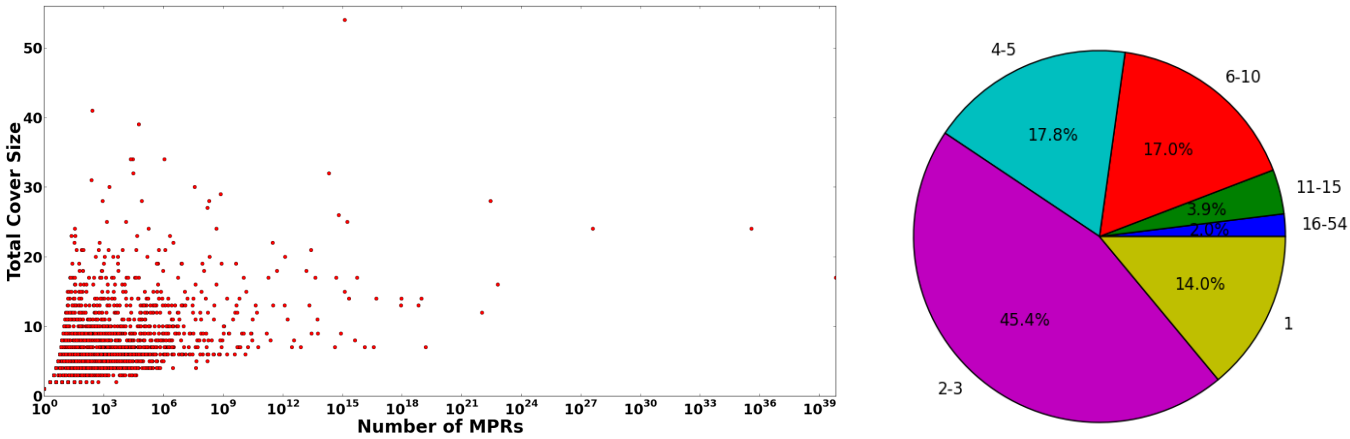
Region-Based: This function scores each event with its robustness with respect to perturbations in event costs. That is, an event that occurs in many MPRs in a specified range of "nearby" event costs should have a high score and one that occurs in fewer MPRs in that range should have a lower score.
To that end, Libeskind-Hadas et al. [11] used the concept of Pareto-optimal reconciliations to partition the space of DTL event costs into a finite number of equivalence classes, or "regions", $R = \{R_1, \ldots, R_n\}$, each comprising the set of positive event costs (for duplication, transfer, and loss; speciation is assumed to be a null event of cost zero) that give rise to the same MPRs. In other words, for any region $R_i \in R$ and any two event costs in R_i, the set of MPRs will be the same for those costs. Computing these regions takes time $O(|G|^5|S| \log |G|)$ [11]. Since event costs are unit-less, we assume that the cost of duplication is normalized to 1 and the cost of transfer and loss are relative to this normalized duplication cost. Given an original set of event costs $(1, C_\Theta, C_\Lambda)$, we first compute the corresponding reconciliation graph $\mathcal{G}$. Next, given two parameters t and ℓ, we use the aforementioned algorithm to compute the regions for transfer costs ranging from $C_\Theta - t$ to $C_\Theta + t$ and for loss costs ranging from $C_\Lambda - \ell$ to $C_\Lambda + \ell$. For each such region, we compute the set of events that arise in the MPRs in that region. Then, for each event in $\mathcal{G}$, its score is the total number of regions that contain that event. Finally, we normalize scores to be between 0 and 1. Ultimately, a score of 1 indicates that this event occurs in at least one MPR in every region in the given range whereas a score close to 0 indicates that the event occurs in relatively few other regions. Note that this scoring function assigns a strictly positive cost to every event in $\mathcal{G}$ since every event occurs in some reconciliation in the region corresponding to the original event costs $(1, C_\Theta, C_\Lambda)$.

B. Experimental Results

To demonstrate the utility of our approach, we tested the kRCP Algorithm on a real dataset comprising 4848 gene trees for a species tree comprising 100 (predominantly prokaryotic) species from the Tree of Life [6]. We used DTL values of 2, 3, and 1, respectively.

We define a *total cover* to be a set of reconciliations that cover all of the events in the space of MPRs, that is, the least value of k in the k-Reconciliations Cover Problem such that the score of the k selected MPRs is equal to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics



(a) (b)

Fig. 3: (a) Total cover size versus number of MPRs and (b) range of total cover sizes for the Tree of Life dataset.

the total score of the events in the space of all MPRs. We define the *total cover size* to be the size of that total cover.

Figure 3(a) shows the distribution of total cover sizes for the 4848 gene trees and (b) shows the range of total cover sizes using event frequency as the scoring function. Almost 50% of the cases required at most three reconciliations to cover the space of all events found in MPRs, and fewer than 6% required more than ten reconciliations. The largest number of reconciliations needed was 54, and this was for a case in which there were over 10^{15} MPRs. These results clearly demonstrate that a small number of reconciliations can be used to summarize the set of events in potentially large reconciliation spaces.

The results for uniform and region-based scoring were very similar with respect to the total cover size. However, the MPRs comprising the total covers under the three scoring schemes were not identical because the kRCP algorithm uses the event scores in choosing the reconciliations. For the Tree of Life data, we found that the frequency and uniform scoring differed in 7.9% of events, uniform and region-based by 9.1%, and frequency and region-based by 10.0%[3].

[3]We chose 100 random gene trees from the Tree of Life dataset. For each one, we computed the total cover using uniform, frequency, and region-based scoring. For each pair of scoring functions, we examined corresponding MPRs in the respective kRCP solutions (e.g., the first MPRs in each solution, the second, etc.). We then computed the percentage difference in those two reconciliations (i.e., the size of the symmetric set difference in their respective sets of events divided by the total number of events in the two reconciliations). We then averaged these percentage differences to compute the overall differences between the two scoring functions.

We also used synthetic simulated datasets from [3] comprising 100 species trees with 50 taxa and three sets of 100 gene trees, constructed with different duplication and transfer rates. For each of the 300 total instances, we used uniform, frequency, and region-based scoring. For these data, the number of MPRs ranged from 1 to 100 and required at most 7 MPRs in a .

For the Tree of Life data, 17.4% of all reconciliations found in the total covers were temporally infeasible. For the synthetic data, only 0.4% of the reconciliations in the s were temporally infeasible. The discrepancy between these rates of temporal infeasibility is not well understood and, while it is not in the scope of this study, it is an interesting issue for future exploration.

C. DTL-RnB Software Tool

The DTL-RnB (**R**econciliatio**n** **B**rowser) is a web-based tool that allows users to upload gene trees, species trees, and leaf associations in newick format, input values for the DTL costs, choose from the aforementioned scoring functions, and browse the MPRs found by the kRCP Algorithm. The reconciliations are rendered using the vistrans tool developed by Matthew D. Rasmussen [14]. The DTL-RnB source code, written in Python 2.7, is also available for download. The web tool, software, and instructions can be found at `www.cs.hmc.edu/dtlrnb`.

For each reconciliation found by the kRCP algorithm, DTL-RnB uses the test described in Section III-C to determine if it is temporally feasible. If so, it uses a topological ordering of the temporal feasibility graph to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

date the species tree and renders the reconciliation with respect to that dating. If the reconciliation is determined to be temporally infeasible, we wish to remove the least number of edges in order to make the temporal reconciliation graph acyclic, and then use that graph to date the species tree. However, this is the NP-complete FEEDBACK ARC SET problem [8]. Thus, DTL-RnB instead uses a greedy heuristic that identifies a cycle in the temporal feasibility graph, removes the edges in that graph corresponding to an arbitrarily selected transfer event in that cycle, and repeats until the graph becomes acyclic. It then uses a topological ordering of this graph to date the species tree. All of the events, including the transfer events that were implicated in cycles, are rendered in the reconciliation. However, those cycle-inducing transfer events are highlighted in the DTL-RnB viewer. This feature is intended to help the user identify the "degree" of temporal infeasibility in the reconciliation. A screenshot of the DTL-RnB software is shown in Figure 4.

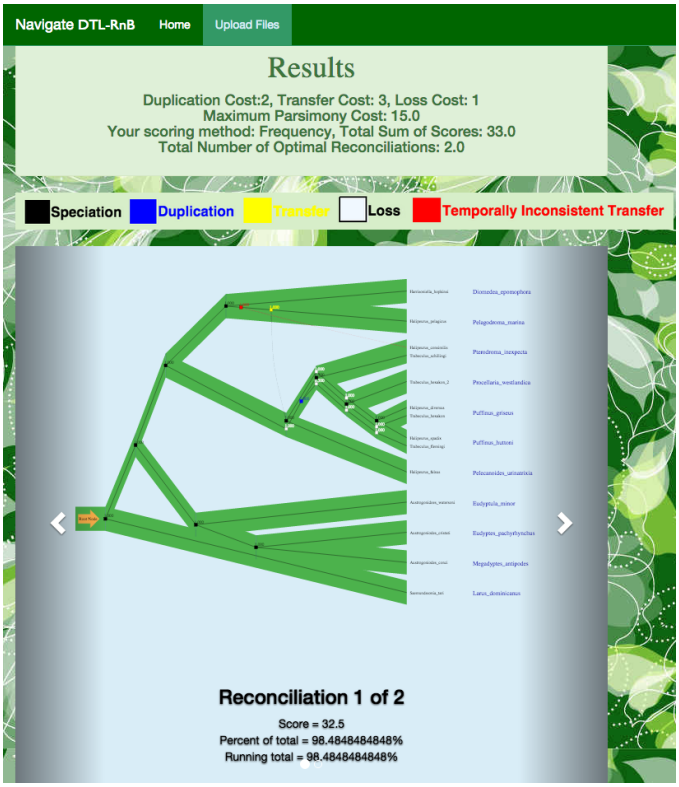


Fig. 4: A screenshot of the DTL-RnB web-based software tool.

V. CONCLUSIONS

In this work, we have described a new approach for summarizing the potentially very large space of maximum parsimony reconciliations. In particular, we formulated the k-Reconciliations Cover Problem (kRCP), proved that it is NP-complete, and showed how the greedy approximation algorithm for the Weighted Maximum Coverage Problem can be applied to find solutions that are within a factor of $1 - \frac{1}{e}$ of optimal. Experimental results indicate that this approach can summarize very large solution spaces using very few maximum parsimony reconciliations. Finally, we implemented our algorithms in the DTL-RnB web-based tool.

There are a number of interesting directions for future research. First, the $1 - \frac{1}{e}$ approximation algorithm used here might be improved by exploiting properties of undated reconciliation graphs. Second, our approach seeks to maximize the total score of the included events for a given number, k, of reconciliations. An alternative is to use an approach like k-means clustering to partition the space of reconciliations into k clusters and report the k reconciliations that best represent each cluster (with respect to an appropriate distance function). Finally, additional experimental studies could help determine the merits of different event scoring functions.

ACKNOWLEDGEMENTS

The authors thank Yi-Chieh Wu for many helpful discussions, advice, and assistance with the data and software used in this paper. The last author also thanks Mukul Bansal for valuable conversations regarding this work. This material is based on work supported by the National Science Foundation under Grant Number IIS-1419739. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

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

[2] Mukul S. Bansal, Eric J. Alm, and Manolis Kellis. Reconciliation revisited: handling multiple optima when reconciling with duplication, transfer, and

loss. In *Research in Computational Molecular Biology*, pages 1–13. Springer, 2013.

[3] Mukul S. Bansal, Yi-Chieh Wu, Eric J. Alm, and Manolis Kellis. Improved gene tree error correction in the presence of horizontal gene transfer. *Bioinformatics*, 31(8):1211–1218, 2015.

[4] Michael A. Charleston and Susan L. Perkins. Traversing the tangle: algorithms and applications for cophylogenetic studies. *Journal of biomedical informatics*, 39(1):62–71, 2006.

[5] Chis Conow, Daniel Fielder, Yaniv Ovadia, and Ran Libeskind-Hadas. Jane: A new tool for cophylogeny reconstruction problem. *Algorithms for Molecular Biology*, 5(16), 2010.

[6] Lawrence A. David and Eric J. Alm. Rapid evolutionary innovation during an archaean genetic expansion. *Nature*, 469:93–96, 2011.

[7] Jean-Philippe Doyon, Celine Scornavacca, K Yu Gorbunov, Gergely J Szöllősi, Vincent Ranwez, and Vincent Berry. An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers. In *Comparative genomics*, volume 6398, pages 93–108. Springer, 2011.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[9] Dorit Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

[10] Ran Libeskind-Hadas and Michael Charleston. On the computational complexity of the reticulate cophylogeny reconstruction problem. *Journal of Computational Biology*, 16(1):105–117, 2009.

[11] Ran Libeskind-Hadas, Yi-Chieh Wu, Mukul S Bansal, and Manolis Kellis. Pareto-optimal phylogenetic tree reconciliation. *Bioinformatics*, 30(12): i87–i95, 2014.

[12] Thi-Hau Nguyen, Vincent Ranwez, Vincent Berry, and Celine Scornavacca. Support measures to estimate the reliability of evolutionary events predicted by reconciliation methods. *PloS One*, 8(10): e73667, 2013.

[13] Yaniv Ovadia, Daniel Fielder, Chris Conow, and Ran. The cophylogeny reconstruction problem is NP-complete. *Journal of Computational Biology*, 18(1):59–65, 2011.

[14] Matthew D. Rasmussen. compbio respository. https://github.com/mdrasmus/compbio/tree/master/bin.

[15] Frank Rutschmann. Molecular dating of phylogenetic trees: A brief review of current methods that estimate divergence times. *Divers. Distrib.*, 12(1): 35–48, 2006.

[16] Celine Scornavacca, Wojciech Paprotny, Vincent Berry, and Vincent Ranwez. Representing a set of reconciliations in a compact way. *Journal of Bioinformatics and Computational Biology*, 11(02): 1250025, 2013.

[17] Cuong Than, Derek Ruths, Hideki Innan, and Luay Nakhleh. Confounding factors in hgt detection: statistical error, coalescent effects, and multiple solutions. *Journal of Computational Biology*, 14 (4):517–535, 2007.

[18] Ali Tofigh. *Using Trees to Capture Reticulate Evolution: Lateral Gene Transfers and Cancer Progression*. PhD thesis, KTH Royal Institute of Technology, 2009.

[19] Ali Tofigh, Michael T. Hallett, and Jens Lagergren. Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(2):517–535, 2011.

[20] Anak Yodpinyanee, Benjamin Cousins, John Peebles, Tselil Schramm, and Ran Libeskind-Hadas. Faster dynamic programming algorithms for the cophylogeny reconstruction problem. Technical Report CS-2011-1, Harvey Mudd College, Department of Computer Science, August 2011.

APPENDIX

A.1 Algorithm for Constructing Undated Reconciliation Graphs

Here we give a formal description of the algorithm for constructing undated reconciliation graphs and derive its worst-case running time.

Lemma. *The worst-case running time of Algorithm 2 is* $O(|G||S|^2)$.

Proof. First, each gene node g can be associated with each species node s as a speciation in two different ways depending on how the two children of s are associated with the two children of g, duplication in just one way, transfer in up to $O(|S|)$ ways depending on which child of g is transferred and the landing site of that transferred child, or loss in two ways. Thus, there are a total of $O(|S|)$ possible associations for a pair (g, s).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

Algorithm 2: MakeReconciliationGraph

Input: Gene tree G, species tree S, and annotated maximum parsimony matrix c.
Output: Undated reconciliation graph $\mathcal{G} = (V_m \dot\cup V_e, E)$ for G and S

1 Initialize Q as an empty queue and V_m, V_e, and E as $\varnothing$
2 $c_{min} = \min_{s \in V(S)} c(rt(G), s)$
3 **foreach** $s \in V(S)$ **do**
4 **if** $c(rt(G), s) = c_{min}$ **then**
5 $V_m = V_m + (rt(G), s)$
6 Enqueue $(rt(G), s)$ to Q
7 **while** Q *is not empty* **do**
8 Dequeue (g, s) from Q
9 **foreach** $event \in c(g, s).events$ **do**
10 $V_e = V_e + event$
11 $E = E + ((g, s), event)$
12 **foreach** $(g', s') \in event.associations$ **do**
13 **if** $(g', s') \notin V_m$ **then**
14 Enqueue (g', s') to Q
15 $V_m = V_m + (g', s')$
16 $E = E + (event, (g', s'))$
17 **return** $\mathcal{G} = (V_m \dot\cup V_e, E)$

Line 2 takes $O(|S|)$ time, the loop in lines 3–6 takes $O(|S|)$ time, and the loop in lines 7–16 takes $O(|G||S|^2)$ time because each pair (g, s) can be dequeued in line 8 and enqueued at line 14 at most once and has $O(|S|)$ associations that can be considered at line 12. □

A.2 Algorithms for Converting Between MPRs and Undated Reconciliation Trees

Algorithms 3 and 4 below transform an MPR to an undated reconciliation tree and an undated reconciliation tree to a MPR, respectively. These transformations establish the bijection between MPRs and undated reconciliation trees.

Algorithm 3: MPRToReconciliationTree

Input: Gene tree G, species tree S, and MPR $\alpha = (\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$ for G and S
Output: Undated reconciliation tree T corresponding to α

1 $V_m(T) = \{(rt(G), \mathcal{M}(rt(G)))\}$
2 Initialize $V_e(T)$ and $E(T)$ as $\varnothing$
3 **foreach** $g \in I(G)$ *in pre-order* **do**
4 **if** $g \in \Sigma$ **then**
5 $event.type = \mathbb{S}_{(g, \mathcal{M}(g))}$
6 **else if** $g \in \Theta$ **then**
7 $event.type = \mathbb{T}_{(g, \mathcal{M}(g))}$
8 **else**
9 $event.type = \mathbb{D}_{(g, \mathcal{M}(g))}$
10 $event.associations = \varnothing$
11 **foreach** $g' \in Ch_G(g)$ **do**
12 $s' = \mathcal{M}(g')$
13 $V_m(T) = V_m(T) + (g', \mathcal{M}(g'))$
14 **if** $g \in \Sigma$ **then**
15 $k = d_S(\mathcal{M}(g), \mathcal{M}(g'))$
16 **else if** $g \in \Theta$ **and** $(g, g') \in \Xi$ **then**
17 $k = d_S(\tau(g), \mathcal{M}(g')) + 1$
18 **else**
19 $k = d_S(\mathcal{M}(g), \mathcal{M}(g')) + 1$
20 **for** j **from** $k - 1$ **to** 1 **do**
21 $event' = (\mathbb{L}_{(g', pa_S(s'))}, \{(g', s')\})$
22 $V_m(T) = V_m(T) + (g', pa_S(s'))$
23 $V_e(T) = V_e(T) + event'$
24 $E(T) = E(T) \cup$ $\{((g', pa_S(s')), event'), (event', (g', s'))\}$
25 $s' = pa_S(s')$
26 $event.associations = event.associations + (g', s')$
27 $V_e(T) = V_e(T) + event$
28 $E(T) = E(T) + ((g, \mathcal{M}(g)), event)$
29 **foreach** $(g', s') \in event.associations$ **do**
30 $E(T) = E(T) + (event, (g', s'))$
31 **foreach** $g \in Le(G)$ **do**
32 $event = (\mathbb{C}_{(g, \mathcal{L}(g))}, \varnothing)$
33 $V_e(T) = V_e(T) + event$
34 $E(T) = E(T) + ((g, \mathcal{L}(g)), event)$
35 **return** T

A.3 NP-completeness

In this section we prove Theorem III.1.

Proof. First, given a solution $\mathcal{T}$ to a kRGCDP instance with $\mathcal{G} = (V_m \dot\cup V_e, E)$, k and ω, we can trivially verify in polynomial time (with respect to $|V_m \dot\cup V_e|$ and $|E|$ whether $|\mathcal{T}| = k$ and $\sum_{v \in V_e(\mathcal{T})} \sigma(v) \geq \omega$. Therefore kRGCDP $\in$ NP.

Next, we show that kRGCDP is NP-hard using a polynomial-time reduction from 3SAT. Suppose we are given a 3SAT instance with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Consider an R-graph $\mathcal{G}$ constructed as follows.

1) For each variable x_i, introduce nodes X_i and X_i'' in $V_m(\mathcal{G})$ and node X_i' (which we call a variable node) in $V_e(\mathcal{G})$. Add an edge from X_i to X_i' and another from X_i' to X_i''.

2) For each variable x_i, introduce node x_i (which we call a literal node) in $V_e(\mathcal{G})$ and an edge from X_i'' to x_i if the literal x_i appears in the given 3SAT instance. Similarly, introduce literal node $\overline{x_i}$ in $V_e(\mathcal{G})$ and an edge from X_i'' to $\overline{x_i}$ if the literal $\overline{x_i}$ appears in the given 3SAT instance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

Algorithm 4: ReconciliationTreeToMPR

Input: Gene tree G, species tree S, and undated reconciliation tree T
Output: MPR $\alpha = (\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$ corresponding to T

Initialize Σ, Δ, Θ and Ξ as $\varnothing$
foreach $(g, s) \in V_m(T)$ **do**
  $\{event\} = Ch_T((g, s))$
  if $event.type \neq \mathbb{L}_{(g,s)}$ **then**
    $\mathcal{M}(g) = s$
  if $event.type = \mathbb{S}_{(g,s)}$ **then**
    $\Sigma = \Sigma + g$
  else if $event.type = \mathbb{D}_{(g,s)}$ **then**
    $\Delta = \Delta + g$
  else if $event.type = \mathbb{C}_{(g,s)}$ **then**
    $\mathcal{L}(g) = s$
  else if $event.type = \mathbb{T}_{(g,s)}$ **then**
    $\Theta = \Theta + g$
foreach $g \in \Theta$ **do**
  $\{(g', s'), (g'', s'')\} = Ch_T(\mathbb{T}_{(g,\mathcal{M}(g))})$
  if $s' = \mathcal{M}(g)$ **then**
    $\Xi = \Xi + (g, g'')$
    $\tau(g) = s''$
  else
    $\Xi = \Xi + (g, g')$
    $\tau(g) = s'$
return $\alpha = (\mathcal{L}, \mathcal{M}, \Sigma, \Delta, \Theta, \Xi, \tau)$

3) For each clause C_j, introduce nodes C_j and C_j'' in $V_m(\mathcal{G})$ and node C_j' (which we call a clause node) in $V_e(\mathcal{G})$. Add an edge from C_j to C_j' and another from C_j' to C_j''.

4) For each literal x_i (or $\overline{x_i}$) that appears in the 3SAT instance, add nodes and edges to the graph as follows. Initialize $A_i = \{\alpha_1^i, \alpha_2^i, \ldots\} = \{\text{node } C_j : \text{clause } C_j \text{ contains literal } x_i\}$. Next, create a binary tree that contains these nodes as leaves (although these leaves will later become internal nodes) as follows: Create $\lceil \frac{|A_i|}{2} \rceil$ new nodes, $\alpha_\ell^{i'} \in V_e(\mathcal{G})$, and add an edge from $\alpha_{\lceil \ell/2 \rceil}^{i'}$ to α_ℓ^i for $1 \leq \ell \leq |A_i|$. Then create $\lceil \frac{|A_i|}{2} \rceil$ new nodes, $\alpha_\ell^{i''} \in V_m(\mathcal{G})$, and add an edge from $\alpha_\ell^{i''}$ to $\alpha_\ell^{i'}$ for each ℓ. Recurse by performing the process described above with A_i set to be the nodes $A_i = \{\alpha_1^{i''}, \ldots, \alpha_\ell^{i''}, \ldots\}$ until A_i contains only one node. At this point, terminate the recursion and denote the single node by $x_i' \in V_m(\mathcal{G})$. Add an edge from x_i to x_i'.

5) Introduce m leaf nodes $L_1, \ldots, L_m \in V_e(\mathcal{G})$. For each leaf node L_i, add a node L_i' in $V_m(\mathcal{G})$ and an edge from L_i' to L_i.

6) For each clause C_j introduce $\binom{m}{2}$ nodes $R_{1,1}^j, R_{1,2}^j, \ldots, R_{1,m}^j$, $R_{2,2}^j, \ldots$, $R_{2,m}^j, \ldots, R_{m,m}^j$ (which we call range nodes) in $V_e(\mathcal{G})$. Add an edge from C_j'' to each of these $\binom{m}{2}$ nodes.

7) For each range node $R_{s,e}^j$ recursively add nodes and edges to the graph as follows: Initialize $B_{s,e}^j = \{\beta_1^j, \beta_2^j, \ldots\} = \{L_s', L_{s+1}', \ldots, L_e'\}$. Create $\lceil \frac{|B_{s,e}^j|}{2} \rceil$ new nodes, $\beta_\ell^{j'} \in V_e(\mathcal{G})$, and add an edge from $\beta_{\lceil \ell/2 \rceil}^{j'}$ to β_ℓ^j for $1 \leq \ell \leq |B_{s,e}^j|$. Then create $\lceil \frac{|B_{s,e}^j|}{2} \rceil$ new nodes, $\beta_\ell^{j''} \in V_m(\mathcal{G})$, and add an edge from $\beta_\ell^{j''}$ to $\beta_\ell^{j'}$ for each ℓ. Recurse by performing the process described above with $B_{s,e}^j$ set to be the nodes $B_{s,e}^j = \{\beta_1^{j''}, \ldots, \beta_\ell^{j''}, \ldots\}$ until $B_{s,e}^j$ contains only one node. At this point, terminate the recursion and denote the single node by $R_{s,e}^{j'} \in V_m(\mathcal{G})$. Add an edge from $R_{s,e}^j$ to $R_{s,e}^{j'}$.

Figure 5 shows an example of an R-graph constructed from a 3SAT instance.

Note that for each of the $O(n)$ literal nodes, we add $O(m)$ nodes in step 4. Then for each of the m clause nodes we add $\binom{m}{2} \in O(m^2)$ range nodes, and for each range node, we add $O(m)$ nodes in step 7. Therefore, $|V(\mathcal{G})| \in O(nm + m^4)$, so this construction takes polynomial time.

We first prove the following claims. For each literal x_i (or $\overline{x_i}$):

1) There exists a nonempty set $\mathcal{T}_{x_i}$ of R-trees in $\mathcal{G}$, each of which contains literal node x_i.

2) Every $T \in \mathcal{T}_{x_i}$ does not contain any other literal nodes.

3) For every $T \in \mathcal{T}_{x_i}$, the only variable node that T contains is X_i'.

4) Every $T \in \mathcal{T}_{x_i}$ contains all clause nodes C_j' corresponding to clauses that contain literal x_i.

We prove Claim 1 by constructing an R-tree containing literal node x_i as follows. We start by choosing x_i to be part of the tree and moving up from x_i. By construction of $\mathcal{G}$, there is only one unique path to a root, specifically X_i. We add all the nodes and edges on that path. We then move down from x_i, adding nodes and edges consistent with the definition of an R-tree. This results in a unique partial traversal up until we reach nodes $C = \{C_j'' : \text{clause } C_j \text{ contains literal } x_i\}$. Next, we add node $R_{j,j}^j$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2016.2537319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

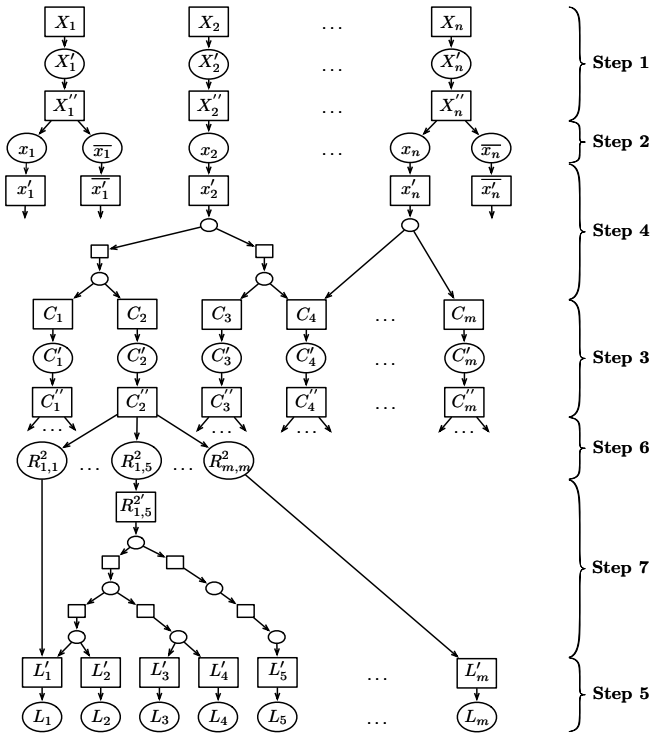


Fig. 5: A sample R-graph constructed from a 3SAT instance. Nodes and edges are labeled on the right with the step number (as described above) in which they were introduced. Nodes in V_m are represented by squares and nodes in V_e are represented by circles. Note that for this graph, the original 3SAT instance included literal x_2 in clauses C_1 through C_4 and literal x_n in clauses C_4 and C_m. None of the clauses contained literal $\overline{x_2}$. Not all edges and nodes are displayed in the figure.

and an edge from the jth element of C to $R^j_{j,j}$ for $1 \leq j < |C|$. We also add node $R^{|C|}_{|C|,m}$ and an edge from the last element of C to $R^{|C|}_{|C|,m}$. Again, we continue adding nodes and edges consistent with the definition of an R-tree, eventually reaching all of the leaf nodes.

Note that upon choosing literal node x_i to be part of the tree, all of the nodes of types mentioned in Claims 2-4 are fixed by construction of $\mathcal{G}$ and by definition of an R-tree. This completes the proof.

Next, we define the scoring function $\sigma : V_e(\mathcal{G}) \to \mathbb{R}_{\geq 0}$ by

$$\sigma(v) = \begin{cases} 1, & \text{if } v = X'_i \text{ for some } 1 \leq i \leq n \\ & \text{or } v = C'_j \text{ for some } 1 \leq j \leq m, \\ 0, & \text{otherwise.} \end{cases}$$

We now show that the given 3SAT instance has a solution if and only if the kRGCDP instance with $\mathcal{G}, \sigma, k = n, \omega = n + m$ has a solution:

($\Rightarrow$) Suppose the given 3SAT instance has a satisfying assignment Φ. Construct $\mathcal{T}$ as follows:

1) Initially, let $\mathcal{T} = \varnothing$.

2) For each $1 \leq i \leq n$: If $\Phi(x_i) = True$, then add an R-tree in G that contains literal node x_i to $\mathcal{T}$; otherwise, add an R-tree that contains literal node $\overline{x_i}$ to $\mathcal{T}$.

Let $\mathcal{X} = \{X'_i \in V_e(\mathcal{T})\}$ and $\mathcal{C} = \{C'_j \in V_e(\mathcal{T})\}$. By our construction of $\mathcal{T}$, $\mathcal{X} = \{X'_i : 1 \leq i \leq n\}$. Moreover, since Φ is a satisfying assignment, $\mathcal{C} = \{C'_j : 1 \leq j \leq m\}$.

Therefore, we have

$$\begin{aligned} \sum_{v \in V_e(\mathcal{T})} \sigma(v) &\geq \sum_{X'_i \in V_e(\mathcal{T})} \sigma(X'_i) + \sum_{C'_j \in V_e(\mathcal{T})} \sigma(C'_j) \\ &= \sum_{i=1}^{n} \sigma(X'_i) + \sum_{j=1}^{m} \sigma(C'_j) \\ &= n + m. \end{aligned}$$

Also, since $|\mathcal{T}| = n$, $\mathcal{T}$ is a valid solution to the kRGCDP instance with $\mathcal{G}, \sigma, k = n, \omega = n + m$.

($\Leftarrow$) Suppose that the kRGCDP instance with $\mathcal{G}$, σ, $k = n$, $\omega = n + m$ has a solution $\mathcal{T}$ such that $|\mathcal{T}| = n$ and $\sum_{v \in V_e(\mathcal{T})} \sigma(v) \geq n+m$. Since only the n variable nodes and m clause nodes have nonzero scores, this implies that all of them are covered by $\mathcal{T}$. Therefore, there is exactly one tree in $\mathcal{T}$ through each of the n variable nodes X'_i. We construct the assignment Φ for the original 3SAT instance as follows. For each $T \in \mathcal{T}$, if T contains x_i, let $\Phi(x_i) = True$; if T contains $\overline{x_i}$, let $\Phi(x_i) = False$. Because every clause node C'_j is covered by $\mathcal{T}'$, by our construction of $\mathcal{G}$, every corresponding clause in the original 3SAT instance is satisfied. Since each variable node is covered by exactly one tree, there are no conflicts in the variable assignments. Therefore, Φ a valid solution to the original 3SAT instance. □