Efficient algorithms for reconciling gene trees and species networks via duplication and loss events

Thu-Hien To, Celine Scornavacca

To and Scornavacca *BMC Genomics* 2015, **16**(Suppl 10):S6
http://www.biomedcentral.com/1471-2164/16/S10/S6

BMC Genomics

RESEARCH Open Access

Efficient algorithms for reconciling gene trees and species networks via duplication and loss events

Thu-Hien To[1,2*], Celine Scornavacca[1,2]

From 13th Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics
Frankfurt, Germany. 4-7 October 2015

Abstract

Reconciliation methods explain topology differences between a species tree and a gene tree by evolutionary events other than speciations. However, not all phylogenies are trees: hybridization can occur and create new species and this results into reticulate phylogenies. Here, we consider the problem of reconciling a gene tree with a species network via duplication and loss events. Two variants are proposed and solved with effcient algorithms: the first one finds the best tree in the network with which to reconcile the gene tree, and the second one finds the best reconciliation between the gene tree and the whole network.

Background

Reconciliations explain topology incompatibilities between a species tree and a gene tree by evolutionary events - other than speciation - affecting genes [[1], for a review]. However, not all phylogenies are trees: indeed, hybridization can occur and create new species [2] and this results into reticulate phylogenies, i.e. species (phylogenetic) networks [3]. In [4], the authors presented a first contribution toward solving a problem similar to the reconciliation problem, namely the cophylogeny problem [5-8], on networks. In their article, they first propose a polynomial algorithm to solve this problem on dated host trees taking into account codivergence, duplication, host switching, and loss events. This model is similar to the DTL model in gene tree reconciliation - that takes into account speciation, duplication, transfer, and loss events [[9], among others]. However, when extending the cophylogeny problem to species networks, their model may not be the more pertinent one for the DTL problem. Indeed, in [4] the parasite tree that is "reconciled" with the host network can take any path in the latter, modeling the fact that some hybridization species can receive the parasites of both parents. In the problem of gene tree reconciliation, their model is more adapted to novel hybridizations, where the genes still keep trace of the polyploidy due to the hybridization. But, for ancient hybridizations, the polyploidy of the extant species being reduced, a model where each gene of an hybridization species can be inherited from at most one of its two parents is more pertinent. In other words, for solving the latter problem, we are interested in finding a tree that is "displayed" by the species network such that its reconciliation with a given gene tree is optimum. We propose an efficient algorithm that takes into account duplication and loss events whose complexity does not depend on the number of hybridization events in the species network but only on the *level* of the network, where the level is a measure of how much the network is "tangled". Moreover, we propose a faster algorithm solving the problem described in [4] when restricting to duplication and loss events (that is, host switching is not taken into account).

Basic notions

We start by giving some basic definitions that will be useful in the paper.

Definition 1 (Rooted phylogenetic network)*A rooted phylogenetic network N on a label set $\mathcal{X}$ is a rooted directed acyclic graph with a single root where each out-degree-0 node (the leaves) is labelled by an element of $\mathcal{X}$. The root of N, denoted by $r(N)$, has indegree* 0 *and*

* Correspondence: thu-hien.to@univ-montp2.fr
[1]ISEM - Université de Montpellier, CNRS, IRD, EPHE, Place Eugène Bataillon, 34392, Montpellier, France
Full list of author information is available at the end of the article

outdegree 2. *All other internal nodes have either indegree* 1 *and outdegree* 2 *(speciation node), or indegree* 2 *and outdegree* 1 *(hybridization node).*

Denote by $V(N)$, $I(N)$, $E(N)$, $L(N)$ and $\mathcal{L}(N)$ respectively the set of nodes, internal nodes (nodes with outdegree greater than 0), edges, leaves and leaf labels of N. The size of N, denoted by $|N|$, is equal to $|V(N)| + |E(N)|$. Given x in $V(N)$, we denote by N_x the subnetwork of N rooted at x, i.e. the subgraph of N consisting of all edges and nodes reachable from x. If x is a leaf of N, we denote by $s(x)$ the label of x in $\mathcal{L}(N)$. If x is a speciation node, we denote by $p(x)$ the only parent of x.

Given two nodes x and y of N, we say that x is *lower or equal* to y in N, denoted by $x \leq_N y$ (resp. *lower*, denoted by $x <_N y$), if and only if there exists a path (possible reduced to a single node) in N from y to x (resp. and $x \neq y$). If $x \leq_N y$, then, for every path p from y to x, denote by $length(p)$ the number of speciation nodes in N such that $x <_N z \leq_N y$. If N is a tree, then, for every two nodes u, v of N, $LCA_N(u, v)$ [10] is the lowest node of N that is above or equal to both u, v.

Given a speciation node x in N, two paths of N starting from x are said to be *separated* if each path contains a different child of x. Let x, y be two nodes of N. Denote by $\mathcal{M}_N(x, y)$ the set of nodes z of N such that there exist two separated paths in N from z to x and y. For example, in Figure 1(d), $\mathcal{M}_{N'}(x, y) = \{m_1, m_2, m_3\}$. Note that all nodes in $\mathcal{M}_N(x, y)$ are speciation nodes and, when N is a tree and x, y are not comparable, $M_N(x, y)$ contains exactly one node, which coincides with $LCA_N(x, y)$.

If every biconnected component of N has at most k hybridization nodes, we say that N is of *level-k* [11]. A rooted phylogenetic tree is a rooted phylogenetic network with no hybridization nodes, i.e. a level-0 network. In the following, we will refer to rooted phylogenetic networks and rooted phylogenetic trees simply as *networks* and *trees*, respectively. In this paper, we allow trees to contain *artificial* nodes, i.e. nodes with indegree and outdegree 1.

Let B be a biconnected component of a network N. Then B contains exactly one node $r(B)$ without ancestors in B [[12], Lemma 5.3]; we call $r(B)$ the root of B. If B is not trivial, i.e. B consists of more than one node, we can *contract* it by removing all nodes of B other than $r(B)$, then connect $r(B)$ to every node with indegree 0 created by this removal. Then the following definitions are well-posed.

Definition 2 (Tree $bc(N)$**)** *Given a network N, the tree* $bc(N)$ *is obtained from N by contracting all its biconnected components.*

For example, Figure 1(a, b) shows respectively a level-2 network N and its associated tree $bc(N)$.

Let denote by $\overset{\circ}{B}$ the node in $bc(N)$ that corresponds to a biconnected component B in N. Given two biconnected component B_i, B_j , we say that $B_i \leq_N B_j$ (resp. $B_i <_N B_j$) if and only if $\overset{\circ}{B}_i \leq_{bc(N)} \overset{\circ}{B}_j$ (resp. $\overset{\circ}{B}_i <_{bc(N)} \overset{\circ}{B}_j$). We say that B_i is the parent (resp. a child) of B_j if $\overset{\circ}{B}_i$ is the parent (resp. a child) of $\overset{\circ}{B}_j$ in $bc(N)$. We also denote by $LCA_N(B_i, B_j)$ the biconnected component corresponding to $LCA_{bc(N)}(\overset{\circ}{B}_i, \overset{\circ}{B}_j)$ in N.

Definition 3 (Elementary network) *Given a network N, each biconnected component B that is not a leaf of N defines an elementary network, denoted by* $N(B)$*, consisting of B and all cut-edges coming out from B.*

Definition 4 (Switchings of a network [13]**)** *Given a network N, a switching S of N is obtained from N by choosing, for each hybridization node, an incoming edge to* switch on *and the other to* switch off. *Once this is done, we also switch off all switched-on edges with the target node having only switched-off outgoing edges (see* Figure 1(e) *for an example). For each bi-connected component B of N, we also denote by* $S(B)$ *the subgraph of S restricted to* $N(B)$.

Switchings will be used in the next section to model gene histories for genes evolving in a species network. For example, Figure 1 presents in (*c*) a switching of the level-2 network in (*a*). We denote by $V_{on}(S)$ the set of nodes of S that are not an endpoint of any switched-off edge. A path of S is a path of N that uses only switched-on edges.

Hereafter, G will denote a tree and N a network such that there is a bijection between $L(N)$ and $\mathcal{L}(N)$ and $\mathcal{L}(G) \subseteq \mathcal{L}(N)$. In the gene tree reconciliation problem, G represents a gene tree such that each leaf corresponds to a contemporary gene and is labeled by the species containing this gene, while N is a species network such that each leaf represents an extant species. In the cophylogeny problem, G represents a parasite tree such that each leaf corresponds to a parasite species that is labeled by the species that hosts it, while N is a host network such that each leaf represents an extant species.

Reconciliations

We will now extend the definition of $\mathbb{DL}$ reconciliation in [14] to networks. In a $\mathbb{DL}$ reconciliation, each node of G is associated to a node of S and an event - a speciation ($\mathbb{S}$), a duplication ($\mathbb{D}$) or a contemporary event ($\mathbb{C}$) - under some constraints. A contemporary event $\mathbb{C}$ associates a leaf u of G with a leaf x of S such that $s(u) = s(x)$. A speciation in a node u of G is constrained to the existence of two separated paths from the mapping of u to the mappings of its two children, while the only constraint given by a duplication event is that evolution of G cannot go back in time. More formally:

Definition 5 (Reconciliation) *Given a tree G and a network N such that* $\mathcal{L}(G) \subseteq \mathcal{L}(N)$*, a* reconciliation

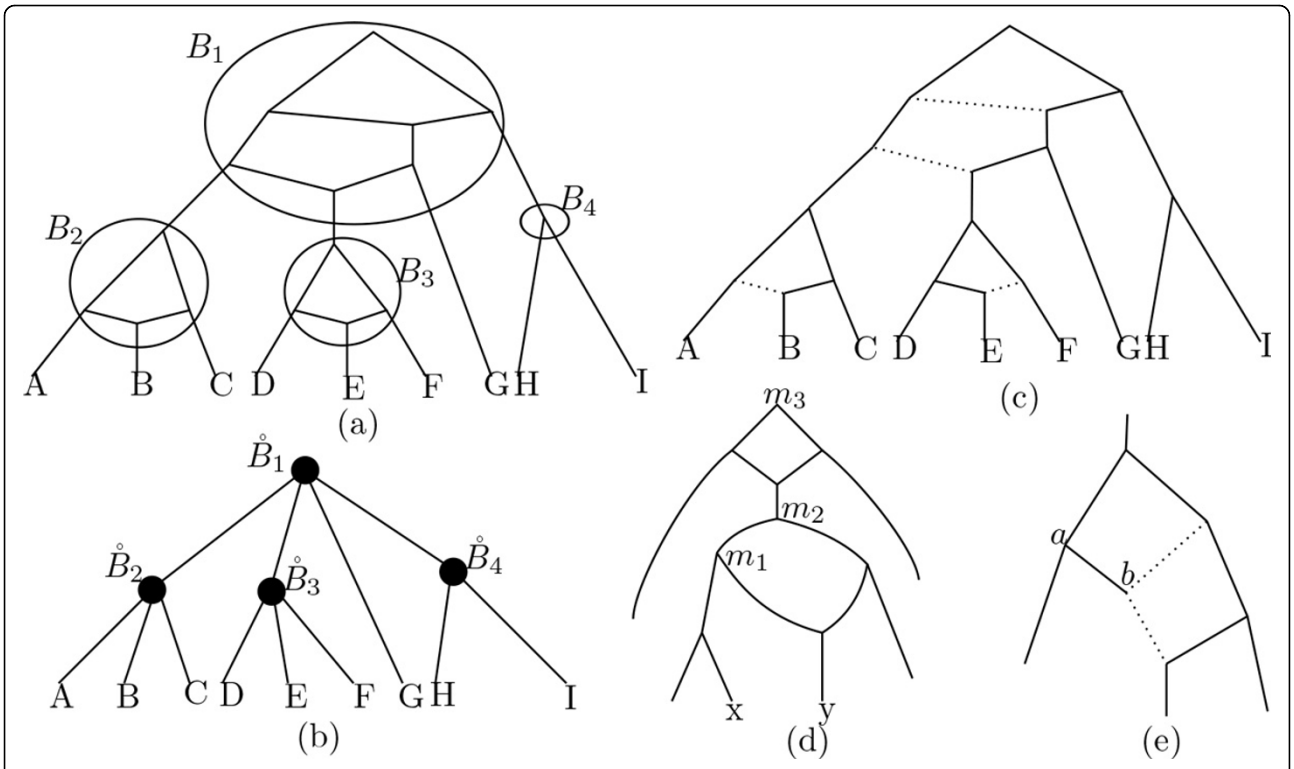

Figure 1 (*a*) An example of a level-2 network *N*, (*b*) the tree *bc*(*N*), (c) a switching of *N* where the switched-off edges are dotted, (d) a network *N'* such that $\mathcal{M}_{N'}(x, y) = \{m_1, m_2, m_3\}$, (e) another switching of a network. Note that the edge (*a*; *b*) will also be switched off since all out-going edges of *b* are off.

between G and N is a function α that maps each node u of G to a pair $(\alpha_r(u), \alpha_e(u))$ *where* $\alpha_r(u)$ *is a node of V (N) and* $\alpha_e(u)$ *is an event of type* $\mathbb{S}$ *or* $\mathbb{D}$ *or* $\mathbb{C}$*, such that:*

- $\alpha_e(u) = \mathbb{C}$ *if and only if* $u \in L(G)$, $\alpha_r(u) \in L(N)$ *and* $s(u) = s(\alpha_r(u))$;
- *for every* $u \in I(G)$ *with child nodes* $\{u_1, u_2\}$*, if* $\alpha_e(u) = \mathbb{S}$*, then* $\alpha_r(u) \in \mathcal{M}_N(\alpha_r(u_1), \alpha_r(u_2))$;
- *for any two nodes u, v of V(G) such that* $v <_G u$*, if* $\alpha_e(u) = \mathbb{D}$*, then* $\alpha_r(v) \leq_N \alpha_r(u)$*. Otherwise,* $\alpha_r(v) <_N \alpha_r(u)$.

Note that, if N is a tree, this definition coincides with the one given in [14].

The number of duplications of α, denoted by $d(\alpha)$ is the number of nodes u of G such that $\alpha_e(u) = \mathbb{D}$. Since in a network there can be several paths between two nodes, we count the number of losses on shortest paths, as done in [4]. In more details, given two nodes x, y of N, the distance between x and y, denoted by $dist(x, y)$, is defined as follows:

- If $y \leq_N x$, then $dist(x, y) = \min_p length(p)$ over all possible paths p from x to y;
- otherwise, $dist(x, y) = +\infty$.

Then, for every $u \in I(G)$ with child nodes $\{u_1, u_2\}$, the number of losses associated with u in a reconciliation α, denoted by $l_\alpha(u)$, is defined as follows:

- if $\alpha_e(u) = \mathbb{S}$, then $l_\alpha(u) = min\{dist(x_1, \alpha_r(u_1)) + dist(x_2, \alpha_r(u_2)), dist(x_1, \alpha_r(u_2)) + dist(x_2, \alpha_r(u_1))\}$ where x_1, x_2 are the two children of $\alpha_r(u)$;
- if $\alpha_e(u) = \mathbb{D}$, then $l_\alpha(u) = dist(\alpha_r(u), \alpha_r(u_1)) + dist(\alpha_r(u), \alpha_r(u_2))$.

The number of losses of a reconciliation α, denoted by $l(\alpha)$, is the sum of $l_\alpha(\cdot)$ for all internal nodes of G. Thus, the cost of α, denoted by $cost(\alpha)$, is $d(\alpha)\cdot\delta + l(\alpha)\cdot\lambda$, where δ and λ are respectively the cost of a duplication and a loss event. We use $cost(G,N)$ to denote the cost of the minimum reconciliations between G and N. A reconciliation having the minimum cost is called a *most parsimonious reconciliation*.

Let S be a switching of N, then a reconciliation between G and S is defined similarly to Definition 5 except that only switched-on edges are considered when defining paths, and only nodes in $V_{on}(S)$ are counted for calculating the length of the shortest path in the definition of *dist*. This is done to model the fact that, since each gene of an hybridization species is inherited from

one of its two parents, we should not count as a loss the fact that the other parent does not contribute. Moreover, note that, for every two nodes x, y of S such that $x \leq_S y$, there is a unique path from y to x in S.

When N is a tree, there is a unique reconciliation (the *LCA reconciliation*) between G and N which has minimum cost and this reconciliation can be found in $O(|G|)$ time [15-18] as follows:

- for each node u in $L(G)$, $\alpha_r(u)$ is defined as the only node x of S such that $s(u) = s(x)$, and $\alpha_e(u) = \mathbb{C}$;
- for each node u in $I(G)$ with child nodes $\{u_1, u_2\}$, $\alpha_r(u) = LCA_N(\alpha_r(u_1), \alpha_r(u_2))$; if $\alpha_r(u_1) \leq_N \alpha_r(u_2)$ or $\alpha_r(u_2) \leq_N \alpha_r(u_1)$ then $\alpha_e(u) = \mathbb{D}$; otherwise $\alpha_e(u) = \mathbb{S}$.

In the LCA reconciliation, the mapping α_e is totally defined by α_r, hence it can be omitted, and we will use α to refer to α_r. Note that the algorithms used on trees to find the LCA reconciliation can also be used on switchings, which - when only switched-on edges are considered - are actually trees. Hereafter, when we refer to *the* reconciliation between a tree and a switching, we refer to the LCA reconciliation between them. The problems in which we are interested in here are defined as follows:

Problem 1 BEST SWITCHING FOR THE $\mathbb{DL}$ MODEL

Input *A tree G, a network N such that* $\mathcal{L}(G) \subseteq \mathcal{L}(N)$, *and positive costs* δ *and* λ *for respectively* $\mathbb{D}$ *and* $\mathbb{L}$ *events.*

Output *A switching S of N such that the cost*(G, S) *is minimum over all switchings of N.*

Problem 2 MINIMUM$\mathbb{DL}$ RECONCILIATION ON NETWORKS

Input *A tree G, a network N such that* $\mathcal{L}(G) \subseteq \mathcal{L}(N)$, *and positive costs* δ *and* λ *for respectively* $\mathbb{D}$ *and* $\mathbb{L}$ *events.*

Output *A minimum reconciliation between G and N.*

Remark 1 *For the sake of simplicity, we suppose that G does not contain any internal node u such that* $|\mathcal{L}(G_u)| = 1$ *(i.e. all nodes of* G_u *are mapped to a leaf of N). If it is not the case, we can simplify the instance by replacing in G each such subtree* G_u *by a leaf labeled by the unique label in* $\mathcal{L}(G_u)$ *and compute a reconciliation for the simplified tree G'. A parsimonious reconciliation for G can be easily obtained from a parsimonious one for G'.*

Method

Best switching

We start by proving that finding the best switching for the $\mathbb{DL}$ model is NP-hard:

Theorem 1 *Problem 1 is NP-hard.*

Proof: To prove the theorem, we reduce Problem 1 to the TREE CONTAINMENT problem, which is NP-hard [19]. The TREE CONTAINMENT problem asks the following "Given a network N and a tree T, both with their leaf sets bijectively labeled by a label set $\mathcal{X}$, is there a switching S of N such that T can be obtained by S deleting all switched-off edges and nodes with indegree and outdegree 1?". Now, assume there is an algorithm $\mathcal{A}$ to solve Problem 1 in polynomial time. Then, it is easy to see that, since $\lambda, \delta > 0$, there is a solution of Problem 1 with cost 0 if and only if G is contained in N. Therefore, this method would provide a polynomial-time algorithm to solve the TREE CONTAINMENT problem, which is impossible.

In the following, we present a fixed-parameter tractable algorithm [20] in the level of the network to solve Problem 1. To do so, we need to introduce some notations.

Definition 6 (Mapping B**)** *For every* $u \in V(G)$, $B(u)$ *is defined as the lowest biconnected component B of N such that* $\mathcal{L}(N_{r(B)})$ *contains* $\mathcal{L}(G_u)$.

Then the following remark holds:

Remark 2 *If* $u \in L(G)$, *then* $B(u)$ *is the only leaf of N such that* $s(u) = s(B(u))$. *If* $u \in L(G)$, *then* $B(u) = LCA_N(B(u_1), B(u_2))$ *where* u_1, u_2 *are child nodes of u in G.*

We define by G_N the tree obtained from G by adding some artificial nodes on the edges of G and label each node of G_N by a biconnected component of N via an extension of the labeling function $B(\cdot)$ as follows:

Definition 7 (Tree G_N**)** *The tree* G_N *is obtained from G as follows: for each internal node u in G with child nodes* u_1, u_2 *such that there exist k biconnected components* $B_{i_1} >_N \ldots >_N B_{i_k}$ *strictly below* $B(u)$ *and strictly above* $B(u_1)$, *we add k artificial nodes* $v_1 > \ldots > v_k$ *on the edge* (u, u_1), *and* $B(v_j)$ *is fixed to* B_{i_j}. *We do the same for* u_2.

See Figure 2 for an example of G_N. We have the following lemma:

Lemma 1 *Let u be a node of* $I(G_N)$, *and u' be one of the children of u. If u is an artificial node, then* $B(u)$ *is the parent of* $B(u')$ *and a child of* $B(p(u))$; *otherwise,* $B(u')$ *is either equal to* $B(u)$ *or one of its children.*

Proof: Suppose that u is an artificial node. Then, by Definition 7, $B(p(u))$, $B(u)$, $B(u')$ are three consecutive nodes of G_N, thus $B(u)$ is the parent of $B(u')$ and a child of $B(p(u))$. Suppose now that u is not an artificial node, and let u'' be the child of u in G such that $u'' \leq_{Gn} u' \leq_{Gn} u$. If $B(u'') = B(u')$, then by definition, $u'' = u'$ because no artificial node is added between u and u'', and thus the claim holds. If $B(u') \neq B(u'')$, then by Definition 7, $B(u')$ is the highest biconnected component of N that is below $B(u)$ and above $B(u'')$, which is then a child of $B(u)$.

Definition 8 (G_B) *Let B be a biconnected component of N different from a leaf, then* G_B *is the set of all*

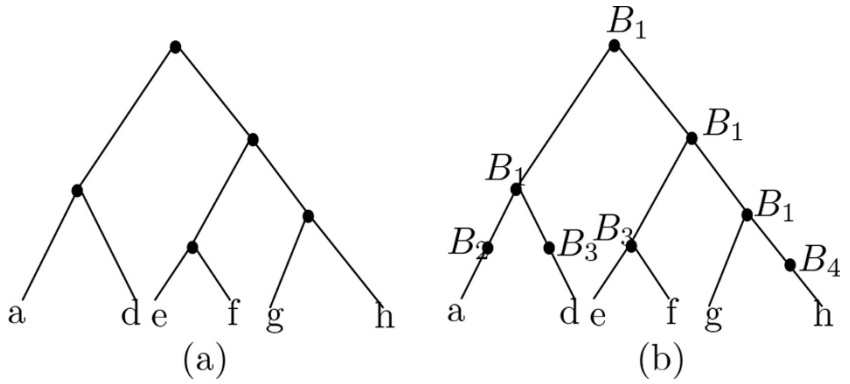

Figure 2 (a) A gene tree *G* and (b) the tree G_N along with the labeling $B(\cdot)$ of its nodes (*N* here is the network in Figure 1(a)**).**

maximal connected subgraph H of G_N such that $B(u) = B$ for every $u \in I(H)$.

For example, in Figure 3, G_{B_1} consists of one binary tree, G_{B_2} consists of one edge, and G_{B_3} contains 1 tree and 1 edge.

Lemma 2 *Let B be a biconnected component of N different from a leaf, then we have the following:*

(i) *for every $H \in G_B$, H is either a binary tree or an edge whose upper extremity is an artificial node. Moreover, for every leaf u of H, $B(u)$ is a child of B.*

(ii) *if $B = B(r(G))$, then G_B consists of one binary tree.*

Proof: **(i)** First, suppose that $I(H)$ contains an artificial node u. Then this artificial node is the only internal node of H; indeed, by Lemma 1, the value of $B(\cdot)$ for the parent and the child of u are both different from B. Thus, H consists of only one edge whose upper extremity is u. If $I(H)$ does not contain any artificial node, it follows that H is a binary tree. Moreover, by Lemma 1 and Definition 8, $B(u)$ is a child of B for every leaf u of H.

(ii) Suppose now that $B = B(r(G))$, and G_B contains at least two components H_1, H_2, rooted at two different nodes r_1, r_2 where $B(r_1) = B(r_2) = B$. Let $r = LCA_{G_N}(r_1, r_2)$, then, by Definition 6, $B(v) = B$ for every node v on the two paths from r to r_1 and to r_2, because $B(r(G)) = B(r_1) = B(r_2) = B$ and $r_1,\ r_2 \leq_{G_N} r \leq_{G_N} r(G)$. But this contradicts the maximality of H_1 and H_2 since they can both be extended. Hence G_B contains only one component. Suppose that this component is an edge; thus, its upper extremity is an artificial node that has been added on the path from a node u to a node v of G where u is strictly higher than $r(G)$. But this is not possible, because $r(G)$ is the highest node of G. Therefore, G_B contains one binary tree.

Given a biconnected component B_i different from a leaf, denote by $G^t_{B_i}$ the set of binary trees of G_{B_i}, and $G^e_{B_i}$ the set of edges of G_{B_i}. Let S_i be a switching of $N(B_i)$, and let H be a tree in $G^t_{B_i}$. By Lemma 2, for every $u \in L(H)$, $B(u)$ is a child of B_i and thus $r(B(u))$ is a leaf of $N(B_i)$, which is also a leaf of S_i. Hence, we can define a reconciliation between H and S_i, denoted by $\beta^H_{S_i}$, such that each leaf u of H is mapped to the leaf $r(B(u))$ of S_i.

Lemma 3 *Let S be a switching of N, and let α be the reconciliation between G and S. For every $u \in I(G)$, there exists $H \in G^t_{B(u)}$ such that $u \in I(H)$, and $\alpha(u) = \beta^H_{S(B(u))}(u)$.*

The proof of this lemma is deferred to the appendix. The following definition will be useful later on.

Definition 9 (*cost*(H,S_i)) *Let B_i be a biconnected component of N different from a leaf, and S_i a switching of $N(B_i)$; then cost(H,S_i) is defined as follows:*

- $\forall H \in G^t_{B_i}$, $cost(H, S_i) = cost(\beta^H_{S_i})$ *if* $B_i = B(r(G))$, *and* $cost(H, S_i) = cost(\beta^H_{S_i}) + \lambda\, dist(r(S_i), \beta^H_{S_i}(r(H)))$ *otherwise;*
- $\forall H \in G^e_{B_i}$, $cost(H, S_i) = \lambda\, dist(r(S_i), \beta^H_{S_i}(r(B(u))))$ *where u is the only leaf of H.*

As we will see later, this cost corresponds to the contribution of H to a reconciliation between G and any

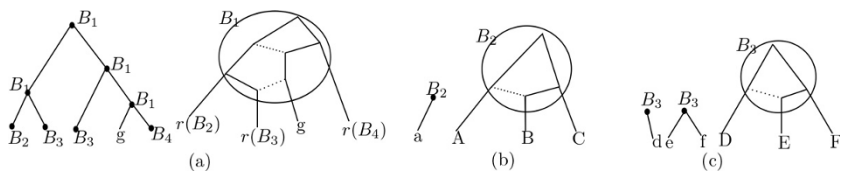

Figure 3 An example of execution of Algorithm 1 on the tree *G* in Figure 2(a) **and the network *N* in** Figure 1(a). (a) G_{B_1} and a switching of $N(B_1)$ on which its reconciliation with G_{B_1} contains 1 duplication and 2 losses. (b) G_{B_2} and a switching of $N(B_2)$ on which the reconciliation with G_{B_2} contains one loss. (c) G_{B_3} and a switching of $N(B_3)$ on which the reconciliation with G_{B_3} contains 2 losses.

switching of N that contains S_i. For example, in Figure 3 (b), H is the edge (B_2, a) and, if S_i is the switching on the left, then $cost(H,S_i) = \lambda$.

The next lemma is a central one, and it permits to solve Problem 1 independently per each biconnected component of N :

Lemma 4 *Let $\{B_1,..., B_p\}$ be the biconnected components of N that are not leaf nodes, and let S be a switching of N where each elementary network $N(B_i)$ has S_i as switching. Then cost* $(G, S) = \sum_{i=1}^{p} \sum_{H \in G_{B_i}} cost(H, S_i)$.

Proof: Let α be the reconciliation between G and S. Denote by $d_\alpha(S_i)$ the number of nodes in $I(S_i)$ associated to a duplication by α. By Remark 1, no duplication happens at leaves of S. Additionally, $\cup_{B_i} \cup_{H \in G^t_{B_i}} I(H) = I(G)$ and the sets of internal nodes of $G^t_{B_i}$ are disjoint. Hence, we have $d(\alpha) = \sum_{i=1}^{p} d_\alpha(s_i) = \sum_{i=1}^{p} \sum_{H \in G^t_{B_i}} d(\beta^H_{S_I})$ because $\beta^H_{S_i}(u) = \alpha(u)$ for every internal node u of H (Lemma 3).

Let us now consider the loss count. Note that a node/edge is on (resp. off) in S_i if and only if it is also on (resp. off) in S. Let x, y be two nodes of S such that $y \leq_S x$. Then we define $dist_{S_i}(x, y)$ as the number of nodes z in $V_{on}(S_i) \setminus L(S_i)$ such that $y <_S z \leq_S x$. Then, for each internal node u of G, we define the number of losses associated with u in S_i, denoted by $l_\alpha(u, S_i)$, similarly to $l_\alpha(u)$ but using the function $dist_{S_i}$ instead of $dist$. Then, $l_\alpha(u) = \sum_{i=1}^{p} l_\alpha(u, S_i)$. Now, let u_1, u_2 be two children of u in G, then $l_\alpha(u, S_i) > 0$ only if the path from $\alpha(u)$ to either $\alpha(u_1)$ or $\alpha(u_2)$ in S contains at least a node of B_i. Therefore, we have three possible cases: (1) $\alpha(u)$ is in is in B_i, (3) $\alpha(u)$ is above $r(B_i)$ while either $\alpha(u_1)$ or $\alpha(u_2)$ is in a biconnected component below B_i. In case (1), by Lemma 3, there exists a binary tree H of $G^t_{B_i}$ such that $u \in I(H)$, and $\alpha(v) = \beta^H_{S_i}(v)$ for every $v \in I(S_i)$, thus $l_\alpha(u, S_i) = l_{\beta^H_{S_i}}(u)$. Now, let suppose S_i that case (2) holds for u_1, then u_1 must be the root of a binary tree H_1 of G^t , and the contribution of u_1 to $l_\alpha(u, S_i)$ is $dist\left(r(S_i), \beta^{H_1}_{S_i}(r(H_1))\right)$. Note that in this case, $u_1 \neq r(G)$. Finally, let suppose that case (3) holds for u_1 and let u_a, u_b be the artificial nodes added on the edge (u, u_1) of G such that $B(u_a) = B_i$ and $B(u_b)$ is a child of B_i. Hence, $(u_a, u_b) \in G^e_{B_i}$, and the contribution of u_1 to $l_\alpha(u, S_i)$ is $dist(r(S_i), r(B(u_b)))$. Let call V^1_i, V^2_i, V^3_i the set of nodes u in the first, second, and third case. By construction, V^1 is disjoint from V^2_i and V^3_i. Moreover, V^2_i and V^3_i are disjoint because if a node u has two children u_1, u_2 such that u_1 is in B_i, and u_2 is below B_i, then u must be in B_i, i.e. cannot be above $r(B_i)$. Thus,

$$l(\alpha) = \sum_{u \in I(G)} l_\alpha(u) = \sum_{u \in I(G)} \sum_{i=1}^{p} l_\alpha(u, S_i) = \sum_{i=1}^{p} \sum_{u \in V^1_i \cup V^2_i \cup V^3_i} l_\alpha(u, S_i).$$

Therefore,

$$cost(G, S) = \delta \cdot d(\alpha) + \lambda \cdot l(\alpha) = \sum_{i=1}^{p} (\delta \cdot d_\alpha(S_i) +$$

$$\lambda \cdot \sum_{u \in I(G)} l_\alpha(u, S_i)) = \sum_{i=1}^{p} (\delta \cdot \sum_{H \in G^t_{B_i}} d(\beta^H_i) +$$

$$\lambda \cdot \sum_{u \in V^1_i \cup V^2_i \cup V^3_i} l_\alpha(u, S_i) = \sum_{i=1}^{p} ([\lambda \cdot \sum_{u \in V^3_i} (u, S_i)] +$$

$$+[\delta \cdot \sum_{H \in G^t_{B_i}} d(\beta^H_{S_i}) + \lambda \cdot \sum_{u \in V^1_i} l_\alpha(u, S_i) +$$

$$\lambda \cdot \sum_{u \in V^2_i} l_\alpha(u, S_i)]).$$

As proved above (in case (3)), the first term between squared brackets is equal to $\sum_{H \in G^t_{B_i}} cost(H, S_I)$ by Definition 9. In the second term between squared brackets, the sum of the two first factors is exactly $\sum_{H \in G^t_{B_i}} cost(\beta^H_{S_i})$ (as proved in case (1)), while the last factor is equal to $\lambda \cdot \sum_{H \in G^t_{B_i}} dist(r(S_i), \beta^H_{S_i}(r(H)))$ (as proved in case (2)). Note that as mentioned in case (2), only nodes that are not the root of G are considered. Hence, the second term between squared brackets corresponds to $\sum_{H \in G^t_{B_i}} cost(H, S_i)$ by Definition 9.

Therefore, $cost(G, S) = \sum_{i=1}^{p} \sum_{H \in G_{B_i}} cost(H, S_i)$ and this concludes the proof.

Algorithm 1 computes the switching of N for which its reconciliation with G has the smallest cost, by analyzing each biconnected component of N independently. See Figure 3 for an example of application of this algorithm to the species network in Figure 1(a) and the gene tree in Figure 2(a).

Algorithm 1 Solving Problem 1

1: **Input:** A species network N and a gene tree G such that $\mathcal{L}(G) \subseteq \mathcal{L}(N)$, and positive costs δ, λ for duplication and loss events, respectively.

2: **Output:** A switching S of N that is optimal in the sense of Problem 1.

3: Compute the tree G_N and its labeling function $B(\cdot)$;

4: Compute G_{B_i} for each biconnected component B_i of N that is not a leaf;

5: **for** each biconnected component B_i of N **do**

6: **for** each switching S^j_i of $N(B_i)$ **do**

7: $cost_j = \sum_{H \in G_{B_i}} cost\left(H, S^j_i\right)$;

8: $S_i^m \leftarrow$ the switching of $N(B_i)$ with the lowest value of $cost_j$ over all j;

9: Return the switching S of N in which each elementary network $N(B_i)$ has S_i^m as switching.

Theorem 2 *Let N be a level-k network with p biconnected components. Algorithm 1 runs in $O(|N| + 2^k \cdot p \cdot |G|)$ time and returns a switching S of N such that cost (G,S) is minimum.*

Proof: **Complexity:** It is well-known that all biconnected components of N can be computed in linear time, i.e. $O(|N|)$, by using depth-first-search [10]. After a linear preprocessing, *LCA* operations on a tree can be performed in constant time [21,22]. Thus, from Remark 2, the mapping $B(\cdot)$ can be computed in $O(|G_N|)$. Hence, the tree G_N can be constructed in times $O(|G_N| + |N|)$.

By a simple traversal of G_N which takes time $O(|G_N|)$, we can compute G_{B_i} for all B_i. Each biconnected component B_i of N has at most k hybridization nodes, then $N(B_i)$ has at most 2^k switchings. For each switching S_i^j of $N(B_i)$, it takes $O(|G_B|)$ to compute $cost_j$ [23,24]. The overall size of all G_{Bi} is the size of G_N . Therefore, the total complexity of the loop at lines 5 - 8 is $O(2^k \cdot |G_N|)$. Each edge of G can have at most p artificial nodes added to it. Hence in the worst case $O(|G_N|) = O(|G| \cdot p)$, i.e. the total complexity of Algorithm 1 is $O(|N| + 2^k \cdot p \cdot |G|)$.

Correctness: Let S be a switching of N where each elementary network $N(B_i)$ has S_i as the switching. By Lemma 4, $cost(G,S) = \sum_{i=1}^{p} \sum_{H \in G_{B_i}} cost(H, S_i)$. Hence $cost(G,S)$ is minimum if and only if $\sum_{H \in G_{B_i}} cost(H, S_i)$ is minimum for every S_i. Lines 5-8 in Algorithm 1 search, for each network $N(B_i)$, the tree S_i^m such that $\sum_{H \in G_{B_i}} cost(H, S_i)$ is minimum, which implies the correctness of the algorithm.

Minimum Reconciliation on Networks

Given a tree G and a network N, in [4] the authors prove that reconciling G on N can be solved in polynomial time, when host switchings (or transfer events, in the DTL reconciliation terminology) are also accounted for. In their model, each node of N is dated by a single date while each node of G is dated by a set of dates, and they search for a parsimonious reconciliation between G and N, i.e. one that has minimum cost, under the constraint that an event associated to a node u of G can only happen at a node/edge of N whose date is contained in the set of possible dates of u. Although the algorithm complexity stays polynomial, it is very high: $O(\tau^3 \cdot |G| \cdot |N|^5)$ for a binary tree and a binary network, where τ is the number of possible dates of the nodes of G and N, which is at most $O(|G| + |N|)$. A drawback of this model is that it requires information on the dates that is often not available. Moreover, transfers are not always possible in all parts of Tree of Life. Here, we take into account only speciation, duplication and loss events, and consider G and N as undated (see Problem 2 and Definition 5 for more details). Using a similar dynamic algorithm on this simpler model, and by some further analyses, we provide an algorithm that is a generalization of the LCA algorithm to networks that has a much smaller complexity than the algorithm in [4], namely $O(h^2 \cdot |G| \cdot |N|)$, where h is the number of hybridization nodes of N.

Let x, y be two nodes of N. Denote by $\mathcal{M}in_N(x, y)$ the subset of $\mathcal{M}_N(x, y)$ such that, for every $z \in \mathcal{M}in_N(x, y)$, there does not exist any $z' \in \mathcal{M}in_N(x, y)$ such that every path from z to x and to y passes through z'. For example, in Figure 1(d), m_1, m_2, m_3 are in $\mathcal{M}_N(x, y)$ but only m_1 and m_2 are in $\mathcal{M}_N(x, y)$ because every path from m_3 to x, y must pass m_2.

For the sake of simplicity, we consider only reconciliations without duplications on hybridization nodes: indeed, since losses are not counted at hybridization nodes, a duplication on such nodes can be moved to its only child without changing the total cost of the reconciliation.

Lemma 5 *Let α be a reconciliation of minimum cost between G and N, then for every node u of G that has two children u_1, u_2, we have:*

(i) if $\alpha_e(u) = \mathbb{S}$, then $\alpha_r(u) \in \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$;
(ii) if $\alpha_e(u) = \mathbb{D}$, then either $\alpha_r(u_1) \leq_N \alpha_r(u_2)$ and $\alpha_r(u) = \alpha_r(u_2)$, or $\alpha_r(u_2) \leq_N \alpha_r(u_1)$ and $\alpha_r(u) = \alpha_r(u_1)$.

Proof: (i) Suppose that $\alpha_e(u) = \mathbb{S}$, then by definition $\alpha_r(u)$ must be a node of $\mathcal{M}_N(\alpha_r(u_1), \alpha_r(u_2))$. Let x_1, x_2 be two children of $\alpha_r(u)$ such that $l_\alpha(u) = dist(x_1, \alpha_r(u_1)) + dist(x_2, \alpha_r(u_2))$. Suppose that $\alpha_r(u) \notin \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$, then there exists a node y in $\mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$ such that every path from $\alpha_r(u)$ to $\alpha_r(u_1)$ (resp. to $\alpha_r(u_2)$) passes for y. Let y_1, y_2 be the two children of y.

First, we suppose that the shortest path from x_1 to $\alpha_r(u_1)$ passes through y, y_1, while the one from x_2 to $\alpha_r(u_2)$ passes y, y_2. Consider the reconciliation α' such that $\alpha'_r(v) = \alpha_r(v)$ and $\alpha'_e(v) = \alpha_e(v)$ for every $v \neq u$, while $\alpha'_r(u) = y$ and $\alpha'_e(u) = \mathbb{S}$. It is easy to see that α' respects Definition 5, and that $d(\alpha) = d(\alpha')$. Denote by $f = dist(\alpha_r(u), y)$, then $f > 0$. The numbers of losses in α and α' are different on those associated with u and $p(u)$ (if u is not the root of G). We thus have $l_\alpha(p(u)) \geq l_{\alpha'}(p(u)) - f$ if $u \neq r(G)$. Moreover, $l_\alpha(u) = dist(x_1, \alpha_r(u_1)) + dist(x_2, \alpha_r(u_2)) = dist(x_1, y) + 1 + dist(y_1, \alpha_r(u_1)) + dist(x_2, y) + 1 + dist(y_2, \alpha_r(u_2)) \geq dist(\alpha_r(u), y) + dist(y_1, \alpha'(u_1)) + dist(\alpha_r(u), y) + dist(y_2, \alpha'(u_2))) \geq 2f + l_{\alpha'}(u)$. Hence, whether u coincides with $r(G)$ or not, $l(\alpha) > l(\alpha')$, i.e. $cost(\alpha) > cost(\alpha')$, a contradiction.

Suppose now that both the shortest paths from x_1 to $\alpha_r(u_1)$ and to $\alpha_r(u_2)$ pass through y, and then pass through y_1 (or y_2). This means that y_1 is above both $\alpha_r(u_1)$ and $\alpha_r(u_2)$. Let y' be one of the lowest nodes below or equal to y_1 that is above both $\alpha_r(u_1)$ and $\alpha_r(u_2)$. Let y'_2, y'_2 be the two children of y', and sup-1 2 pose, without loss of generality, that y'_1 is above or equal to $\alpha_r(u_1)$ and y'_2 is above or equal to $\alpha_r(u_2)$. Hence, the shortest path from x_1 to $\alpha_r(u_1)$ must pass, in the order, through y, y_1, y', and y', while the shortest path from x_2 to $\alpha_r(u_2)$ must pass through y, y_1, y', and y'_1. By defining the reconciliation α'as done above apart for $\alpha'(u)$, which is fixed to y', we arrive at a contradiction by a similar argument.

(ii) Suppose that $\alpha_e(u) = \mathbb{D}$, and $\alpha_r(u_1)$, $\alpha_r(u_2)$ are not comparable. Hence, $\mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$ is not empty. If $\alpha_r(u) \notin \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$, then there exists a node $\gamma \in \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$ such that every path from $\alpha_r(u)$ to $\alpha_r(u_1)$ and to $\alpha_r(u_2)$ must pass through y. Similarly to case (i), we have that the reconciliation α' that coincides with α apart for the fact that $\alpha'_r(u) = \gamma$ and $\alpha'_e(u) = \mathbb{S}$ has smaller cost than α, a contradiction. Hence, $\alpha_r(u)$ 2 $\mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$. Considering now the reconciliation α' that coincides with α but for $\alpha'(u)$, which is fixed to $\mathbb{S}$. Hence $d(\alpha) = d(\alpha')+1$, and $l_\alpha(p(u)) = l_{\alpha'}(p(u))$ if $u \neq r(G)$. Let x_1, x_2 be the two children of $\alpha_r(u)$. If the shortest path from $\alpha_r(u)$ to $\alpha_r(u_1)$ passes through x_1 while the shortest path from $\alpha_r(u)$ to $\alpha_r(u_2)$ passes through x_2, then $l_\alpha(u) = 2 + l_{\alpha'}(u)$. Therefore, $cost(\alpha) > cost(\alpha')$, a contradiction. Thus, both the two shortest paths from $\alpha_r(u)$ to $\alpha_r(u_1)$ and $\alpha_r(u_2)$ must pass through x_1 (or x_2). Let y be one of the lowest nodes located on both of these paths. Then, $y \in \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$. By the same argument as in the previous case, the reconciliation α'' coinciding with α but for $\alpha''_r(u) = y$ and $\alpha''_e(u) = \mathbb{S}$ must have smaller cost than α, a contradiction.

Hence, either $\alpha_r(u_1) \leq_N \alpha_r(u_2)$ or $\alpha_r(u_2) \leq_N \alpha_r(u_1)$. Suppose that the first case holds (the second case is similar), but $\alpha_r(u) \neq \alpha_r(u_2)$, i.e. $\alpha_r(u_2) <_N \alpha_r(u)$. Let x_1, x_2 be two children of $\alpha_r(u)$. If the shortest path from $\alpha_r(u)$ to $\alpha_r(u_1)$ passes through x_1 while the shortest path from $\alpha_r(u)$ to $\alpha_r(u_2)$ passes through x_2, then by replacing $\alpha_e(u)$ by an $\mathbb{S}$ event, we obtain a reconciliation with a smaller cost. Thus, both shortest paths from $\alpha_r(u)$ to $\alpha_r(u_1)$ and $\alpha_r(u_2)$ must pass through x_1 (or x_2). Let y be a node that is located on both paths such that there is no other node below y on these two paths. Then, $y \neq \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$. By the same argument as in the case (i), the reconciliation α' such that $\alpha'_r(v) = \alpha_r(v)$, $\alpha'_e(v) = \alpha_e(v)$ for every $v \neq u$, $\alpha'_r(u) = y$, $\alpha'_e(u) = \mathbb{S}$ must have smaller cost than α, a contradiction.

Now, we are ready to describe a dynamic algorithm to compute a reconciliation of minimum cost between G and N. Let α be a reconciliation between G and N. For every $u \in V(G)$, denote by $cost_\alpha(u)$ the cost of the reconciliation of α restricted to G_u. Hence, if α is a most parsimonious reconciliation, then $cost_\alpha(u)$ is the minimum cost among all reconciliations between G_u and N that maps u to $\alpha_r(u)$. Algorithm 2 aims at computing, for each u, the set $\mathcal{C}(u)$ containing all pairs (x,c) such that c is the minimum cost among all reconciliations between G_u and N mapping u to x. It is straightforward to see that the cost of a most parsimonious reconciliation between G and N is the minimum cost involved in a pair in $\mathcal{C}(r(G))$.

The function $merge(L_1, L_2)$ used in Algorithm 2 takes as input two lists of pairs (x,c) - where x is a node of N and c is a real number - and merges them keeping, for each x, the pair (x,c) with the smallest value of c. The method $computeMin(y, z)$ used in Algorithm 2 is detailed in Algorithm 3. This method computes, for two nodes y, z of N, the set $\mathcal{M}in_N(y, z)$ by using two breath-first-searches (BFS) starting respectively from y and z up to the root of N (note that, to perform the breath-first-searches, the edges are considered as directed in the inverse order). For this, it labels each node v in such a way that, if v is not strictly above y and z, then $label(v) = \emptyset$. Otherwise, $label(v)$ is the lowest node such that every path from v to y and z passes through it. This method also computes the value of the function $dist$ between y (resp. z) and each node visited in the corresponding breath-first-search.

Algorithm 2 Solving Problem 2

1: **Input:** A network N and a tree G such that $\mathcal{L}(G) \subseteq \mathcal{L}(N)$, and positive costs δ, λ for duplication and loss events, respectively.

2: **Output:** The set $\mathcal{C}(u)$ of pairs (x,c) for every $u \in V(G)$.

3: **for** each node u of G in post-order **do**
4: $\mathcal{C}(u) \leftarrow \emptyset$;
5: **if** u is a leaf **then**
6: Let x be the leaf of S such that $s(x) = s(u)$;
7: $\mathcal{C}(u) \leftarrow \{(x, 0)\}$;
8: **else**
9: Let u_1, u_2 be the two children of u;
10: **for** each $(y, c_1) \in \mathcal{C}(u_1)$ and each $(z, c_2) \in \mathcal{C}(u_2)$ **do**
11: $computeMin(y, z)$;
12: $C \leftarrow \emptyset$;
13: **for** each $x \in \mathcal{M}in_N(y, z)$ **do**
14: Let x_1, x_2 be the two children of x in N;
15: $c = c_1 + c_2 + \lambda \cdot min\{dist(x_1, y) + dist(x_2, z), dist(x_2, z) + dist(x_1, y)\}$;
16: $C \leftarrow C \cup \{(x, c)\}$;
17: **if** $y \leq_N z$ **then**
18: $c = \delta + \lambda \cdot dist(z, y) + c_1 + c_2$;
19: $C \leftarrow C \cup \{(z, c)\}$;

20: **else if** $z \leq_N y$ **then**
21: $c = \delta + \lambda \cdot dist(y, z) + c_1 + c_2$;
22: $C \leftarrow C \cup \{(y, c)\}$;
23: $\mathcal{C}(u) = merge(\mathcal{C}(u), C)$;
24: Return $\mathcal{C}$.

The following theorem proves the correctness of Algorithm 2:

Theorem 3 *Algorithm 2 returns a matrix $\mathcal{C}$ such that, for every $u \in V(G)$, (x,c) is contained in $\mathcal{C}(u)$ if and only if there exists a most parsimonious reconciliation between G_u and N mapping u to x with cost c.*

Algorithm 3 *computeMin*(y, z)

1: Proceed a BFS from y, store the ordered list of visited nodes in $BFS(y)$ and compute, for each u in $BFS(y)$, $dist(y,v)$;
2: Do the same from z;
3: **for** each node $v \in BFS(y)$ **do**
4: **if** $v = y$ **or** $v = z$ **or** v is not in $BFS(z)$ **then** $label(v) = \emptyset$
5: **else if** v has only one child v_1 that is in $BFS(z)$ or $BFS(y)$ **then**
6: $label(v) = label(v_1)$;
7: **else**
8: Let v_1, v_2 be the two children of v;
9: **if** $label(v_1) = label(v_2) \neq \emptyset$ **then**
10: $label(v) = label(v_1)$;
11: **else**
12: $label(v) = \{v\}$; Add v into $\mathcal{M}in_N(y, z)$;

Proof: For each $u \in V(G)$, we need to prove that, if α' is a reconciliation of minimum cost between G_u on N, then $(\alpha'(u), cost_{\alpha'})$ is contained in $\mathcal{C}(u)$. This is obviously true for every leaf of G (by lines 5-7). Let now u be an internal node having two children u_1, u_2. Then, following Lemma 5, $\mathcal{C}(u)$ can be computed from $\mathcal{C}(u_1)$ and $\mathcal{C}(u_2)$ by using the information contained in $\mathcal{M}in_N$ and $dist$. Lines 10-23 in Algorithm 2 computes $\mathcal{C}(u)$ following this lemma.

It remains now to prove that Algorithm 3 correctly computes $\mathcal{M}in_N(y, z)$. For every node v that is above y, z, denote by $low(v)$ the lowest node such that every path from v to y, z must pass through this node. There is only one such node. Indeed, suppose that there are two such nodes m, m', then every path from v to y must pass through both m, m', i.e. either $m <_N m'$ or $m' <_N m$, contradicting the lowest property of m, m'. To prove the claim, we need to show that if v is a node of $BFS(y)$, then:

(i) if v is not strictly above y and z, then $label(v) = \emptyset$;
(ii) otherwise, $label(v) = low(v)$ and line 12 of Algorithm 3 is performed if and only if v is actually in $\mathcal{M}in_N(y, z)$.

Let v be a node of $BFS(y)$, then (i) holds by line 4. Now consider the case where v is strictly above y and z. We will prove (ii) by recursion on v following the order of the nodes in $BFS(y)$. The recursion begins from the set of lowest nodes that are strictly above y and z, i.e. the set of nodes of v in $\mathcal{M}in_N(y, z)$ such that there is not any node in $\mathcal{M}in_N(y, z)$ that is below v. Let v_1, v_2 be two children of v, then by hypothesis v_1, v_2 are not strictly above y, z, i.e. $label(v_1) = label(v_2) \neq \emptyset$; and $low(v) = v$. Thus, due to line 12, $label(v) = v = low(v)$. Now let v be a node strictly above y, z such that (ii) is correct for each node below v which is strictly above or equal to y, z. If v is a hybridization node, then it is evident that $low(v) = low(v_1)$ where v_1 is the only child of v. Moreover, since $label(v_1) = low(v_1)$ (by the hypothesis of recurrence), then $label(v) = label(v_1) = low(v_1) = low(v)$. If v is a speciation node having two children v_1, v_2 such that v_2 is not in either $BFS(y)$ or $BFS(z)$, then we also have $low(v) = low(v_1)$. Hence, due to lines 5 - 6, we have $label(v) = label(v_1) = low(v_1) = low(v)$. Now consider the last case, i.e. v is a speciation node having two children v_1, v_2 that are both in $BFS(y)$ and $BFS(z)$. If there exists a node $q = low(v_1) = low(v_2)$, then $low(v) = q$ because every from v to y, z must pass either through v_1 or v_2, i.e. always pass through q. Following line 10, we fix $label(v) = label(v_1) = q = low(v)$. Moreover, in this case v can not be in $\mathcal{M}in_N(y, z)$ because every path from v to y, z passes through a node q below v. In the last case, we have $label(v_1) \neq label(v_2)$. Since both v_1, v_2 are above y, z, then there exists a node $q_1 = label(v_1)$, and a different node $q_2 = label(v_2)$. We will prove that $v \in \mathcal{M}in_N(y, z)$. Indeed, suppose otherwise, then there is a node q such that every path from v to y, z must pass through q. Hence, every path from v_1 (resp. v_2) to y, z must also pass through q, so $q_1 \leq_N q$ (resp. $q_2 \leq_N q$). It means that there is a path from v_1 to q to q_2 and then to y, z that does not pass through q_1, a contradiction. Hence v is in $\mathcal{M}in_N(y, z)$, and thus by definition the only node that every path from v to y, z must pass through is v itself (line 12).

We now present some intermediate results that will be useful to prove the complexity of Algorithm 2.

We extend the definition of $\mathcal{M}_N$ to a subset of leaves. Let L be a subset of $L(N)$. If $|L| = 1$, then $\mathcal{M}_N(L) = L$. Otherwise, $\mathcal{M}_N(L)$ is the set of nodes m of N such that m is above all leaves in L and there exist at least two separated paths from m to two distinct leaves of L.

Given a node u of G, $L_N(u)$ is defined as the set of leaves of N to which α maps the leaf set of G_u, i.e. $L_N(u) = \{x \in L(N) \mid \exists u \in L(G_u) \text{ and } s(u) = s(x)\}$.

Lemma 6 *Let α be a most parsimonious reconciliation between G and N, then, for every node u of G, $\alpha_r(u) \in \mathcal{M}_N(L_N(u))$.*

Proof: It is true for every leaf u of G. Let u now be an internal node having two children u_1, u_2.

If u_1, $u_2 \in L(G)$, then following Remark 1, $L_N(u)$ consists of two distinct nodes $\alpha_r(u_1)$, $\alpha_r(u_2)$. By

Lemma 5, $\alpha_e(u) = \mathbb{S}$, i.e. there exist two separated paths from $\alpha_r(u)$ to $\alpha_r(u_1)$ and $\alpha_r(u_2)$. It means that $\alpha_r(u) \in \mathcal{M}_N(L_N(u))$.

Let $u_1 \notin L(G)$, then following Remark 1, $|L_N(u_1)| \geq 2$, then there always exist two distinct leaves x, y of $L_N(u)$ such that $x \in L_N(u_1)$, $y \in L_N(u_2)$, i.e. x is below $\alpha_r(u_1)$ and y is below $\alpha_r(u_2)$. If $\alpha_e(u) = \mathbb{S}$, then following Lemma 5, $\alpha_r(u) \in \mathcal{M}in_N(\alpha_r(u_1), \alpha_r(u_2))$, i.e. there exist two separated paths from $\alpha_r(u)$ to $\alpha_r(u_1)$, $\alpha_r(u_2)$. By extending these two paths from $\alpha_r(u_1)$ to x, and from $\alpha_r(u_2)$ to y, we have to two separated paths from $\alpha_r(u)$ to x, y. In other words, $\alpha_r(u) \in \mathcal{M}_N(L_N(u))$. If $\alpha_e(u) = \mathbb{D}$ and suppose that $\alpha_r(u_2) \leq_N \alpha_r(u_1)$, then $\alpha_r(u) = \alpha_r(u_1)$ following Lemma 5, and $\alpha_r(u)$ is not a leaf of N following Remark 1. Let u' be the highest node such that $u' \leq_G u_1$ and $\alpha_e(u') = \mathbb{S}$. Then following Lemma 5, $\alpha_r(u') = \alpha_r(u)$, and there exists two separated paths from $\alpha_r(u')$, i.e. from $\alpha_r(u)$, to two distinct leaves of $L_N(u')$, i.e. two distinct leaves of $L_N(u)$. We can prove the claim similarly for the case when $\alpha_r(u_1) \leq_N \alpha_r(u_2)$.

Lemma 7 *If N is a network that contains h hybridization nodes, then for every subset L of $L(N)$, $|\mathcal{M}_N(L)| \leq h + 1$ holds.*

The proof of this lemma is deferred to the appendix. We are now ready to state the complexity of Algorithm 2.

Theorem 4 *The time complexity of Algorithm 2 is $O(h^2 \cdot |G| \cdot |N|)$ where h is the number of hybridization nodes of N.*

Proof: For every $u \in V(G)$, $|\mathcal{C}(u)|$ is equal to the possible nodes of N that u can be mapped to, which is bounded by $|\mathcal{M}_N(L_N(u))|$ by Lemma 6, and so by $O(h)$ following Lemma 7.

The *for* loop at lines 3 - 23 is performed $|G|$ times, and, at each iteration, the *for* loop at lines 10-23 is performed $O(h^2)$ times. In each iteration of the second loop, the operation *computeMin*, as detailed in Algorithm 3, requires two breath-first-search traversals, which can be performed in time $O(|N|)$. Moreover, for every node x of $\mathcal{M}in_N(y, z)$, by definition there exists two separated paths from x to y, z, which can be extended to be two separated paths from x to two distinct leaves l_1, l_2 of $L_N(u)$ where l_1 is a leaf below y of and l_2 is a leaf below z. This is always possible because, by Remark 1, $|L_N(u)| > 1$. Hence, x must be in $\mathcal{M}_N(L_N(u))$ by definition, i.e. $\mathcal{M}in_N(y, z) \subseteq \mathcal{M}_N(L_N(u))$, and thus $|\mathcal{M}in_N(y, z)| \leq |\mathcal{M}_N(L_N(u))| \leq (h + 1)$. Therefore, the loop at lines 13 - 16 can be performed in time $O(h)$. The operation *merge*(L_1, L_2) at lines 23 for two lists of size $O(h)$ can be implemented in times $O(h)$, if we know that the resulting list is also of size $O(h)$. Hence, it takes $O(|N| + h) = O(|N|)$ times for each iteration of the loop 10 - 23. Therefore, the total complexity is $O(h^2 \cdot |G| \cdot |N|)$.

Finally, a reconciliation of minimum cost between G and N can be then obtained by a standard back-tracking of the matrix $\mathcal{C}$, starting from any pair (x,c) of $\mathcal{C}(r(G))$ such that c is the minimum value over all pairs in $\mathcal{C}(r(G))$.

Conclusions

In this paper, we have studied two variants of the reconciliation problem between a gene tree and a species network. In particular, for the problem of finding the "most parsimonious" switching of the network, even though the number of switchings can be exponential with respect to the number of hybridization nodes, we proposed an algorithm that is exponential only with respect to the level of the network, which is often low for biological data. Moreover, the problem of finding a reconciliation between a gene tree and a network, which was solved in [4] for a more general model but with a very high complexity, was re-studied here for a simpler model, which is more pertinent for same parts of the Tree of Life, and an algorithm with a much smaller complexity was provided. In a further work, we intend to implement the algorithms presented in this paper and apply them to biological data.

Appendix

Proof of lemma 3

By Definition 8, for every $u \in I(G)$, there must exist $H \in G_{B(u)}$ such that $u \in I(H)$. If H is an edge, then u is the only internal node of H, which must be an artificial node by Lemma 2. But this is not possible because nodes of G cannot be artificial. Hence, H must be a binary tree. Let denote $B_i = B(u)$, and $S_i = S(B(u))$. We will prove now that $\alpha(u) = \beta^H_{S_i}(u)$ by recursion on the height of u.

Let u be an internal node of G that has two children u_1, u_2 in G, and let H be the binary tree of G_{B_i} such that $u \in I(H)$. Denote $B_{i_1} = B(u_1)$, $B_{i_2} = B(u_2)$, and $S_{i_1} = S(B(u_1))$, $S_{i_2} = S(B(u_2))$. For $j = 1, 2$, if u_j is a leaf, let H_j be equal to u_j, otherwise H_j is the binary tree of $G_{B_{i_j}}$ such that $u_j \in I(H_j)$. For the sake of convenience, if u_j is a leaf, we also denote by $\beta^{H_j}_{S_{i_j}}$ the reconciliation that maps the only leaf u_j to the only leaf x of N such that $s(x) = s(u_j)$. Note that $s(x) = \alpha(u_j)$.

We now suppose that $\alpha(u_j) = \beta^{H_j}_{S_{i_j}}(u_j)$ for $j = 1, 2$ (which is evidently true if u_j is a leaf), and we will show that this implies that the claim is true for u.

Let u'_1(resp. u'_2) be the child of u in G_N such that $u_1 \leq_{G_N} u'_1 <_{G_N} u$ (resp. $u_2 \leq_{G_N} u'_2) < G_N u$). We respectively denote $B_{i'_1} = B(u'_1)$ and $B_{i'_2} = B(u'_2)$. By definition of the LCA reconciliation, we have $\beta^H_{S_i}(u) = LCA_{S_i}(\beta^H_{S_i}(u'_1), \beta(u'_2))$.

(i) If $B_{i_1} = B_{i_2}$, then $B_i = B_{i_1} = B_{i'_1} = B_{i2} = B_{i'_2}$, and $H = H_1 = H_2$. This implies that $u'_1 = u_1$, because otherwise H

will contain an artificial node. The same holds for u'_2 and u_2. Thus, $\alpha(u) = LCA_S(\alpha(u_1), \alpha(u_2)) = LCA_S(\beta^{H_1}_{S_{i_1}}(u_1), \beta^{H_2}_{S_{i_2}}(u_2)) = LCA_S(\beta^H_{S_i}(u_1), \beta^H_{S_i}(u_2)) = LCA_S(\beta^H_{S_i}(u_1), \beta^H_{S_i}(u_2)) = LCA_S(\beta^H_{S_i}(u'_1), \beta^H_{S_i}(u'_2)) = \beta^H_{S_i}(u)$.

(ii) If $B_{i_1} <_N B_{i_2}$, then $B_i = B_{i_2}$, and $H = H_2$. As in point (*i*), this implies $u'_2 = u_2$ and thus $\alpha(u) = LCA_S(\alpha(u_1), \alpha(u_2)) = LCA_S(\beta^{H_1}_{S_{i_1}}(u_1), \beta^H_{S_i}(u'_2))$. If $u'_1 = u_1$, then we have $LCA_S(\beta^{H_1}_{S_{i_1}}(u_1), \beta^H_{S_i}(u'_2)) = LCA_S(\beta^H_{S_i}(u'_1), \beta^H_{S_i}(u'_2)) = \beta^H_{S_i}(u)$. Otherwise, u'_1 is an artificial node which is a leaf of H that is mapped to $r(B'_{i_1})$ by $\beta^H_{S_i}$. By Lemma 1, $B_{i_1} \leq_N B_{i'_1} <_N B_i$, so all nodes of B_i are above all nodes of B_{i_1} and above $r(B_{i'_1})$. This implies that $LCA_S(\beta^{H_1}_{S_{i_1}}(u_1), \beta^H_{S_i}(u'_2)) = LCA_S(r(B_{i'_1}), \beta^H_{S_i}(u'_2)) = LCA_S(\beta^H_{S_i}(u'_1), \beta^H_{S_i}(u'_2)) = \beta^H_{S_i}(u)$. Similarly for the case where $B_{i_2} <_N B_{i_1}$.

(iii) Suppose now that B_{i_1}, B_{i_1} are not comparable and that $u'_1 \neq u_1$ and $u'_2 \neq u_2$ (the other cases can be shown reusing the arguments of point (*ii*)). Then, similarly as in point (*ii*), $B_{i_1} \leq_N B_{i'_1} <_N B_i$, $B_{i_2} \leq_N B_{i'_2} <_N B_i$, and u'_1, u'_2 are leaves of H mapped respectively to $r(B_{i'_1})$ and $r(B_{i'_2})$ by $\beta^H_{S_i}$. Since $\beta^{H_1}_{S_{i_1}}(u_1)$ is a node of B_{i_1}, $\beta^{H_2}_{S_{i_2}}(u_2)$ is a node of B_{i_2}, then $LCA_S(\beta^{H_1}_{S_{i_1}}(u_1), \beta^{H_2}_{S_{i_2}}(u_2)) = LCA_S(r(B_{i'_1}), r(B_{i'_1})) = LCA_S(\beta^H_{S_i}(u'_1), \beta^H_{S_i}(u'_2)) = \beta^H_{S_i}(u)$.

Therefore, in all cases we always have $\alpha(u) = \beta^H_S(u)$, and thus the same is true for every $u \in I(G)$ by recursion.

Proof of lemma 7

We will prove this lemma by recursion on the number of hybridization nodes. See the figure 4 for an example.

If there is no hybridization node, then N is a tree, and it is evident that $\mathcal{M}_N(L)$ contains exactly one node.

Suppose that the claim is correct for every network having h hybridization nodes. Let N now be a network that has $h + 1$ hybridization nodes. Let (a, b) be an edge of N having as target a hybridization node (namely, b) such that it does not exist any hybridization node above a (such a node always exists because N is a directed acyclic graph). Let N' be the network obtained by removing (a, b) from N (and also removing all nodes of indegree 1 and outdegree 1 created by this removal), then N' has exactly h hybridization nodes. Let $Q = \mathcal{M}_N(L) \setminus \mathcal{M}_{N'}(L)$, then every node q in Q must be above a. Indeed, if q is not above a, then every path from q to every node of L does not contains (a, b), thus q is in $\mathcal{M}_{N'}(L)$, a contradiction. Moreover, by hypothesis, there is no hybridization node above a, hence all nodes of Q must be contained in a path leading a, and this path does not contain any hybridization to node. Let enumerate the nodes in Q as $q_1, \ldots q_m$ from the lowest to the highest one.

If $|Q| \leq 1$, then $|\mathcal{M}_N(L)| = |\mathcal{M}_{N'}(L)| + |Q| \leq h + 1 + 1 = h + 2$, we are done.

Suppose now that $|Q| = m > 1$. In the following, we will define $m - 1$ edges of N having as target a hybridization node such that if N'' is the network obtained from N' by removing these edges, then $\mathcal{M}_{N'}(L) = \mathcal{M}_{N''}(L)$.

Denote by L^* the set of nodes in L that are below q_m in N'. Hence $L \setminus L^*$ is not empty since otherwise q_m would be in $\mathcal{M}_{N'}(L)$, a contradiction. For every q_i, $i = 1, \ldots, m$, let q'_i, q''_i be the two children of q_i such that q'_i is above or equal to a. Hence, q''_i, is not above or equal to a since there is no hybridization node above a, thus every path from q_i to $L \setminus L^*$ must pass through q'_i and a, b. By definition, there must exist at least two separated paths from q_i to two leaves of L. Hence, for every i, there exists always a path from q_i to a node of L^* that passes through q''_i. Denoted this node by $l(q_i)$.

Denote by L_m the set of nodes in L^* such that, for every $l \in L_m$, there is a path from q to l that passes through q''_m. As explained above, L_m is not empty. Recursively, for every $i < m$, let L_i be the set of nodes in $L^* \setminus L_m \setminus \ldots L_{i+1}$ such that, for every $l \in L_i$, there is a path from q_i to l that passes q''_i. Note that this set may be empty if $i < m$, and for every $i \neq j$, $L_i \cap L_j = \emptyset$.

We will define, for each $i < m$, an index $c(i)$ that is strictly greater than i such that $L_{c(i)} \neq \emptyset$, together with two paths p_i (resp. p'_i) from q_i (resp. $q_{c(i)}$) to a node of $L_{c(i)}$ as follows: if $l(q_i)$ is not in L_i, then by the definition, there exists a unique j such that L_j contains $l(q_i)$, and $j > i$. We fix $c(i) = j$. Next, we define p_i (resp. p'_i) as a path from q_i (resp. $q_{c(i)}$) to $l(q_i)$ that passes q''_i (resp. $q''_{c(i)}$). If $l(q_i)$ is in L_i, then let $c(i)$ be the smallest number that is greater than i and $L_{c(i)} \neq \emptyset$ (such an index always exists because $L_m \neq \emptyset$). Let l' be a node of $L_{c(i)}$. Since q_i is in $M_N(L)$, then there must exist a path from q_i to l', and we define p_i as this path. The path p'_i is defined as the one from $q_{c(i)}$ to l' that passes through $q''_{c(i)}$. Note that p_i, p'_i must contain at least one common hybridization node since they start from different nodes and end at a same leaf of L. Denote by h_i the highest common hybridization node of p_i and p'_i. Hence, all h_is are distinct since each p_i starts at a different node q_i, and each p'_i starts at a node $q_{c(i)}$ that is strictly greater than q_i. We define (a_i, b_i) recursively in increasing order of i from 1 to m - 1 as follows. If $i = 1$, then b_i is the highest hybridizatition node on p_i. If $i > 1$, then b_i is the highest hybridization node on p_i and different from all b_k for every $k < i$. There exists always such a node b_i, for example h_i. Therefore, all b_is are distinct. Denote by a_i the parent of b_i on the path p_i.

Denote by N'' the network obtained from N' by removing all edges (a_i, b_i). For every node x in $\mathcal{M}_{N'}(L)$, we will prove that x is also in $\mathcal{M}_{N''}(L)$. Denote by x', x'' the two children of x. By definition, for every $l \in L$, there exists a path, denoted by $f'(l)$, in N' from x to l such that at least one path among them passes through x' and one other passes through x''. To prove that x is in $\mathcal{M}_{N''}(L)$, we will now construct another set of paths in N'' (i.e. in N' and does not contain any (a_i, b_i)) from x

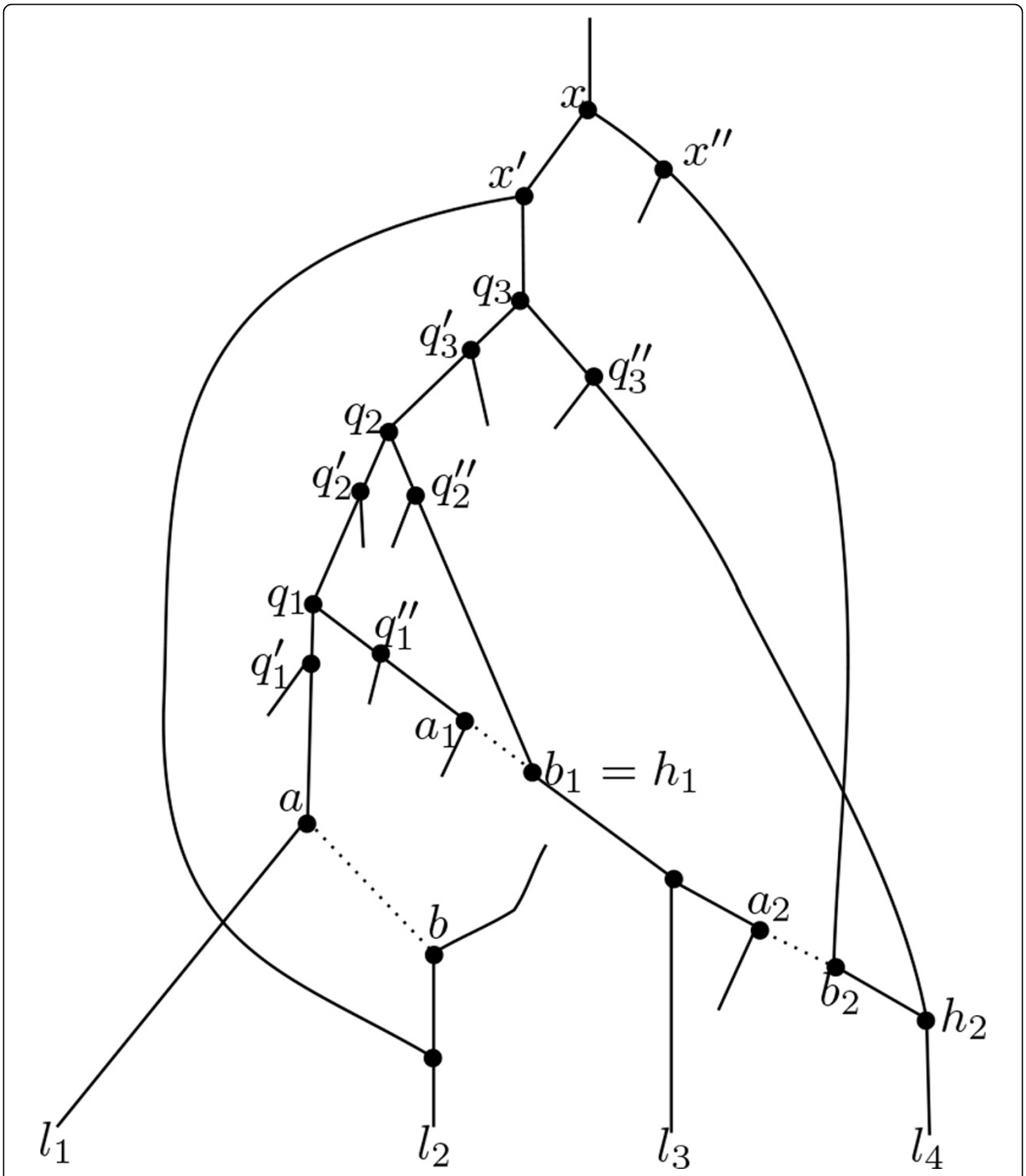

Figure 4 An illustration for the proof of Lemma 7. In this example: $L = \{l_1, l_2, l_3, l_4\}$, $Q = \{q_1, q_2, q_3\}$, $L^* = \{l_1, l_3, l_4\}$, $L_3 = \{l_4\}$, $L_2 = \{l_3\}$, and $L_1 = \emptyset$. For $i = 1$, let $l(q_1) = l_3$; thus, we have $c(1) = 2$ and the path p_1 starts from q_1, passes through q_1'', b_1 to l_3, while the path p_1' starts from q_2, passes through q_2'', b_1 to l_3. In this case, we have $h_1 = b_1$. For $i = 2$, let $l(q_2) = l_3$; thus we have $c(2) = 3$ and the path p_2 starts from q_2, passes through q_2'', b_1, b_2, h_2 to l_4, while the path p_2' starts from q_3 passes through q_3'' to l_4. Let x be a node in $\mathcal{M}_{N'}(L)$, then x is also in $\mathcal{M}_{N''}(L)$ where N'' is obtained from N' by removing all edges (a_i, b_i).

to each leaf l of L, denoted by $f''(l)$, such that at least one path among them passes through $x'$and one other passes through x''.

Consider first the case that x is above q_m (as shown in the figure 4). Without loss of generality, suppose that x' is above or equal to q_m while x'' is not. Suppose that $f'(l)$ contains q_m, then $l \in L^*$ and $f'(l)$ must pass through x' because there is no hybridization node above q_m. Suppose that $l \in L_1 \cup \ldots L_m$. Let k be the index such that $l \in L_k$, then we can choose a path $f''(l)$ in N' from x to l that does not contain any (a_i, b_i) as follows. This path starts from x, passes through x', goes down to q_k , then takes the path from q_k to l that contains q''. Note that this path does not include any p_i since, by construction, every path p_i starts from q_i and goes to a node in $L_{c(i)}$ that is different from L_i, while this path passes through q_k and goes to a node in L_k . Moreover, this path and p_i can not have any common hybridization node above a_i because b_i is the highest hybridization node on p_i. Hence, it can not pass through (a_i, b_i) for any i. If l is not in $L_1 \cup \ldots L_m$, it means that every path from q_m to l must pass through q'_1, then we take the path starting from x, going down to q'_i, then continuing to l. It is evident that this path does not include any p_i, and it cannot have with p_i any common hybridization node above a_i because b_i is the highest hybridization node on p_i. Hence it does not pass through any (a_i, b_i). If $f'(l)$ does not contain q_m, then we fix $f''(l) = f'(l)$. It is easy to see that $f'(l)$ does not contain any edge (a_i, b_i) because otherwise $f'(l)$ and p_i must have at least a common hybridization node above a_i (since there is no hybridization node above q_i). But this is not possible because b_i is the highest hybridization node on p_i. Remark that at least one of the paths $f'(l)$ in this case must pass through x'' since all paths in the first case must pass through x'. Hence at least two of the paths $f''(l)$ are separated, thus x is in $\mathcal{M}_{N''}(L)$ by definition.

Consider now the case that x is not above q_m, then similarly as in the previous case where $f'(l)$ does not contain q_m, we deduce that $f'(l)$ does not contain any (a_i, b_i) for every l. Hence, by choosing $f''(l) = f'(l)$ for every l, we are done.

Therefore, we have $\mathcal{M}_{N'}(L) = \mathcal{M}_{N''}(L)$.

The network N'' contains h - $|Q|$ + 1 hybridization nodes, then following the hypothesis of recurrence, $|\mathcal{M}_{N''}(L) \leq h - |Q| + 2$. This implies that $|\mathcal{M}_{N'}(L)| = |\mathcal{M}_{N''}(L)| \leq h - |Q| + 2$, thus $|\mathcal{M}_N(L)| = |\mathcal{M}_{N'}(L)| + |Q| \leq h - |Q| + 2 + |Q| = h + 2$.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
Both authors contributed to design the models, algorithms and to write the paper.

Acknowledgements
This work was partially funded by the French Agence Nationale de la Recherche Investissements d'Avenir/ Bioinformatique (ANR-10-BINF-01-02, Ancestrome). The publication charges of this article were funded by the French Grant Agence Nationale de la Recherche: Investissements d'Avenir/ Bioinformatique (ANR-10-BINF-01-02, Ancestrome).
This article has been published as part of *BMC Genomics* Volume 16 Supplement 10, 2015: Proceedings of the 13th Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics: Genomics. The full contents of the supplement are available online at http://www.biomedcentral.com/bmcgenomics/supplements/16/S10.

Authors' details
[1]ISEM - Université de Montpellier, CNRS, IRD, EPHE, Place Eugène Bataillon, 34392, Montpellier, France. [2]Institut de Biologie Computationnelle (IBC), 95 rue de la Galera, 34095, Montpellier, France.

Published: 2 October 2015

References
1. Doyon J-p, Ranwez V, Daubin V, Berry V: **Models, algorithms and programs for phylogeny reconciliation.** *Briefings in bioinformatics* 2011, **12(5)**:392-400, doi:10.1093/bib/bbr045.
2. Maddison WP: **Gene trees in species trees.** *Systematic Biology* 1997, **46(3)**:523-536 [http://sysbio.oxfordjournals.org/content/46/3/523.full.pdf+html], doi:10.1093/sysbio/46.3.523.
3. Huson D, Rupp R, Scornavacca C: **Phylogenetic Networks.** Cambridge University Press; 2010.
4. Libeskind-Hadas R, Charleston MA: **On the computational complexity of the reticulate cophylogeny reconstruction problem.** *JCB* 2009, **16(1)**:105-117.
5. Page RDM: **Parallel phylogenies: reconstructing the history of host-parasite assemblages.** *Cladistics* 1994, **10(2)**:155-173.
6. Ronquist F: **Reconstructing the history of host-parasite associations using generalized parsimony.** *Cladistics* 1995, **11(1)**:73-89.
7. Charleston M: **Jungles: a new solution to the hostparasite phylogeny reconciliation problem.** *Math Biosci* 1998, **149(2)**:191-223.
8. Merkle D, Middendort M: **Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information.** *Theory Biosci* 2005, **123(4)**:277-299.
9. Doyon JP, Scornavacca C, Gorbunov KY, Szöllősi GJ, Ranwez V, Berry V: **An effcient algorithm for gene/species trees parsimonious reconciliation with losses duplications and transfers.** In *Research in Computational Molecular Biology: Proceedings of the 14th International Conference on Research in Computational Molecular Biology (RECOMB). Volume 6398.* LNCS, Springer, Berlin/Heidelberg, Germany; 2010:93-108, Software downloadable at http://www.atgc-montpellier.fr/Mowgli/.
10. Cormen TH, Stein C, Rivest RL, Leiserson CE: *Introduction to Algorithms.* 2 edition. McGraw-Hill Higher Education; 2001.
11. Choy C, Jansson J, Sadakane K, Sung W-K: **Computing the maximum agreement of phylogenetic networks.** *Theoretical Computer Science* 2005, **335(1)**:93-107.
12. Fischier M, van Iersel L, Kelk S, Scornavacca C: **On Computing the Maximum Parsimony Score of a Phylogenetic Network.** *SIAM Journal on Discrete Mathematics - SIDMA* 2015.
13. Kelk S, Scornavacca C, Van Iersel L: **On the elusiveness of clusters.** *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 2012, **9(2)**:517-534.
14. Doyon J, Chauve C, Hamel S: **Space of gene/species trees reconciliations and parsimonious models.** *Journal of Computational Biology* 2009, **16(10)**:1399-1418.
15. Zmasek CM, Eddy SR: **A simple algorithm to infer gene duplication and speciation events on a gene tree.** *Bioinformatics* 2001, **17(9)**:821-828.
16. Goodman M, Czelusniak J, Moore GW, Romero-Herrera AE, Matsuda G: **Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences.** *Systematic Biology* 1979, **28(2)**:132-163.
17. Page RDM: **Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas.** *Systematic Biology* 1994, **43(1)**:58-77.

18. Chauve C, El-Mabrouk N: **New perspectives on gene family evolution: Losses in reconciliation and a link with supertrees.** *Research in Computational Molecular Biology, 13th Annual International Conference, RECOMB 2009, Tucson, AZ, USA, May 18-21, 2009. Proceedings* 2009, 46-58.
19. Kanj Ia, Nakhleh L, Than C, Xia G: **Seeing the trees and their branches in the network is hard.** *Theoretical Computer Science* 2008, **401**:153-164, doi:10.1016/j.tcs.2008.04.019.
20. Downey RG, Fellows MR: **Fundamentals of Parameterized Complexity.** *Springer* 2013, **4.**
21. Gabow HN, Tarjan RE: **A linear-time algorithm for a special case of disjoint set union.** *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing* 1983, 246-251.
22. Harel D, Tarjan RE: **Fast algorithms for finding nearest common ancestors.** *SIAM Journal on Computing* 1984, **13(2)**:338-355.
23. Zhang L: **On a mirkin-muchnik-smith conjecture for comparing molecular phylogenies.** *Journal of Computational Biology* 1997, **4**:177-187.
24. Bender M, Farach-Colton M: **The lca problem revisited.** *LATIN 2000: Theoretical Informatics. Lecture Notes in Computer Science Springer* 2000, **1776**:88-94.

doi:10.1186/1471-2164-16-S10-S6
Cite this article as: To and Scornavacca: **Efficient algorithms for reconciling gene trees and species networks via duplication and loss events.** *BMC Genomics* 2015 **16**(Suppl 10):S6.