Formal proofs of the algorithm for "Beyond synteny: a scalable phylogenomic method for whole-genome duplication detection"

This document provides all the technical details required to achieve amortized time $O(1)$ per gene-remap entry, for the paper mentioned in the above title.

A1 Formal details of the model and algorithm

We first provide all the preliminary notions needed to understand our algorithm. In order to keep this material self-contained and avoid back-and-forth reading between the main text and this document, we repeat some of the notation from the main text. Nonetheless, the coverage of these notions is expanded by providing additional justifications and explanations for a better understanding of the principles behind the model.

The reader solely interested in the dynamic programming procedure for cost changes may jump directly to Section A2.1 (just note the notation $H_m(s) := h(F_m(s))$ not introduced in the main text).

Preliminary notions

All trees in this paper are rooted and are assumed to be binary. The root of a tree T is denoted $r(T)$ and its set of leaves is $L(T)$. The *distance* between two nodes u, v of T is the number of edges on the unique path between u and v, and is denoted $dist_T(u, v)$. The *height* of T is the maximum distance between $r(T)$ and a leaf of T and is denoted $h(T)$. A *forest* is a graph in which each connected component is a tree. We denote the set of trees that form F as $t(F)$. The set of leaves of F is $L(F) = \bigcup_{T \in t(F)} L(T)$, and the *height* of F is $h(F) = \max_{T \in t(F)} h(T)$. For $X \subseteq V(F)$, the subgraph of F induced by X is denoted $F[X]$. For $v \in V(F)$ contained in some tree $T \in t(F)$, the *height* of v is the height of the subtree of T rooted at v, and the *depth* of v is the distance between $r(T)$ and v.

A node $u \in V(F)$ is a descendant of a node $v \in V(F)$ if u and v are in the same tree T of $t(F)$, and v is on the path between $r(T)$ and u, in which case v is an ancestor of u. We write $u \preceq_F v$ if u is a descendant of v, and $u \prec_F v$ when $u \neq v$. We may drop the subscript F if it is clear from the context. If $u \prec v$ and $uv \in E(F)$, then u is a *child* of v and v is the *parent* of u. Two nodes u, v of F are *incomparable* if neither is an ancestor of the other. If $X \subseteq L(F)$ is a set of leaves that all belong to the same tree of $t(F)$, we write $\texttt{lca}_F(X)$ for the lowest common ancestor of X, that is, the node of F that is an ancestor of every element of X and that is the farthest from the root of the tree. We may write $\texttt{lca}_F(x, y)$ instead of $\texttt{lca}_F(\{x, y\})$. Note that a tree is a special case of a forest, so all the above notation also applies to trees.

Reconciliations

A *gene tree* is a tree in which the leaves contain extant genes, and a *gene forest* is a forest G of gene trees. A *species tree* is a tree S in which $L(S)$ is a set of extant species. A *leaf species map* is a function $\sigma : L(G) \to L(S)$ that assigns each extant gene to its extant species.

Given a gene forest G, a species tree S, and a leaf species map σ, a *reconciliation* is a map $m : V(G) \to V(S)$ that assigns each gene forest node to a species node. Moreover, m must satisfy the following conditions:

- *(preserve leaf maps)*: for each $u \in L(G)$, $m(u) = \sigma(u)$;
- *(time-consistency)*: for each non-root node $u \in V(G)$ with parent p_u, we have $m(u) \preceq_S m(p_u)$.

The first condition states that a reconciliation should preserve our knowledge of the species that contain extant genes, whereas the second condition states that the gene-species map should be time-consistent, i.e., a gene g cannot belong to a species that existed before the species of an ancestor of g.

The *lca-mapping* between G and S is the unique reconciliation μ that, for each internal node u of G, satisfies

$$\mu(u) = \texttt{lca}_S(\{\sigma(w) : w \in L(G) \text{ and } w \preceq u\}).$$

That is, the lca-mapping assigns each gene tree node the lowest possible species according to the time-consistency rule of reconciliation. See Figure 1 top row for an example.

Speciation, duplication, and loss

In the species tree S, each internal node x represents a species that underwent a speciation event, and the two children of the node are the species that result from this event. As for the gene forest G, each internal node u represents an ancestral gene that either underwent speciation at the same time as its species $m(u)$, in which case the two children of u should be found in the two child species of $m(u)$, or that underwent duplication, in which case the two children are the copies of u and should be in $m(u)$. Of course, genes can also be lost, and the observed children of u may be found in species that are *lower* than the expectation. Nonetheless, we can still formulate clear rules as to when a gene node represents a speciation (*Spec*) or duplication (*Dup*) event.

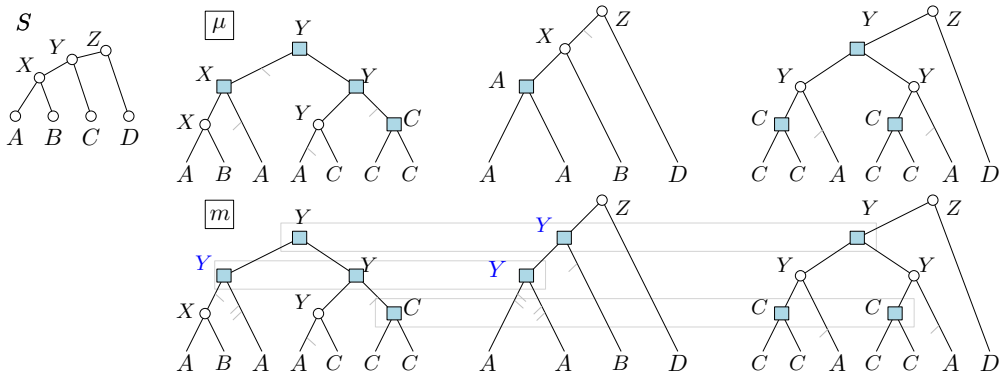

Figure 1: Two possible segmental reconciliations for the same set of gene trees. The top row shows a species tree S with a gene forest of three gene trees. The leaves are labeled by σ, that is, the species that contains them, and the internal node labels represent the lca-mapping μ. Duplications are shown as a blue square, speciations as a white circle, and losses as light gray edges. On the bottom row, an alternate reconciliation m with modified mappings in blue. This can be explained with three segmental duplications, highlighted by gray rectangles.

Let G be a gene forest, S a species tree, σ a leaf species map, and m a reconciliation. The *event labeling under* m is a function $ev_m : V(G) \setminus L(G) \to \{Spec, Dup\}$ that assigns an event to each ancestral gene as follows. Let u be an internal node of G and let u_1 and u_2 be its two children. Then:

$$ev_m(u) = \begin{cases} Spec & \text{if } m(u_1) \text{ has children } x_1, x_2 \text{ such that} \\ & m(u_1) \preceq_S x_1 \text{ and } m(u_2) \preceq_S x_2 \\ Dup & \text{otherwise.} \end{cases}$$

A node $u \in V(G)$ with $m(u) = s$ and $ev_m(u) = Dup$ is called an *s-duplication* (with respect to m). In essence, u is a speciation when its two child gene descend from the two distinct species that resulted from the speciation event. Notice that such a node could be predicted as a duplication, but it was shown that under parsimony, if a node can be a *Spec* according to m, then it should be a *Spec*. As an example, in middle gene tree of Figure 1, the node mapped to X according to μ is a *Spec* (top), but the same node mapped to Y according to m is a *Dup* (bottom).

Gene losses are counted on the branches of G by listing the species that should be present but are not. More specifically, for an edge $uv \in E(G)$, with u the parent of v, the number of losses on uv is

$$l_m(uv) = \begin{cases} dist_S(m(u), m(v)) - 1 & \text{if } ev_m(u) = Spec \\ dist_S(m(u), m(v)) & \text{if } ev_m(u) = Dup \end{cases}$$

The total number of losses under m is $l_m = \sum_{uv \in E(G)} l_m(uv)$. In the traditional DL model, the goal is to find a reconciliation that minimizes the number of *Dup* nodes plus the number of losses. We now turn to our segmental model.

Segmental duplications

For a reconciliation m and a species $s \in V(S)$, the *duplication forest* of s (under m), denoted $F_m(s)$ is the subgraph of G induced by the duplications mapped to s:

$$F_m(s) = G[\{v \in V(G) : m(v) = s \text{ and } ev_m(v) = Dup\}].$$

A segmental duplication is a single duplication event that contains multiple genes. Since such genes must coexist, they must be incomparable in F, and it is not hard to see that the smallest possible number of segmental duplications in species s is the height of $F_m(s)$. As a shorthand, we write $H_m(s) = h(F_m(s))$ to denote that height. Note that the notation $H_m(s)$ **was not defined in the main text** but will be used often below.

The total number of segmental duplications according to m is

$$d_m = \sum_{s \in V(S)} H_m(s).$$

For example in Figure 1, $d_\mu = 5$ (two duplications in Y, one in X, A, C), whereas $d_m = 3$ (two duplications in Y, one in C).

We can finally state our formulation.

The Most Parsimonious Segmental Reconciliation
Input: a gene forest G, a species tree S, a leaf species map σ, and a duplication cost δ and loss cost λ.
Goal: find a reconciliation m between G and S that minimizes $\delta \cdot d_m + \lambda \cdot l_m$.

We note that it was shown in our earlier work that the lca-mapping μ is a solution for the problem when $\delta \leq \lambda$, but otherwise the problem is NP-hard.

A2 Efficient exploration of the solution space

Since the Most Parsimonious Segmental Reconciliation is NP-hard, we turn to algorithms that can explore the space of solutions efficiently. We propose the following basic strategy:

1) start with an initial reconciliation m (typically μ);
2) for each gene node $u \in V(G)$, and for each species s that u could possibly be mapped to, compute the change in cost in m if we remap u to s;
3) choose one of the remapping moves from the previous step according to some criteria, and apply the remapping;
4) repeat step 2 until no more moves need to be considered.

In the following, we clarify what is meant by "possibly mapped to" and "applying the remapping", and later on discuss concrete strategies to choose moves and stop exploring. In what follows, we let G be a gene forest, S a species tree, and m is a reconciliation on which we want to apply a move. We consider the following types of moves.

Up-moves. Let $u \in V(G)$ and let $s \succ m(u)$. A node $u \in V(G)$ can be mapped to any ancestor of $\mu(u)$. Such a remapping may create time inconsistencies, for instance if we remap u to a species that pre-existed $m(p_u)$, where p_u is the parent of u. We therefore support remapping moves that can also remap ancestors to preserve time-consistency. See Figure 2 left box for an example, where remapping u to Z enforces remapping all the ancestors in Y.

The *up-remapping of u to s* is another mapping denoted $m[u \to s]$ in which:

- $m[u \to s](w) = s$ for every ancestor w of u such that $m(w) \prec s$ (including u);
- $m[u \to s](w) = m(w)$ for every other node of G.

The astute reader may ask whether up-moves that remap parents are necessary, as it would be simpler to only consider moves that do not remap the parent. Let us call this a *simple* up-move. The middle gene

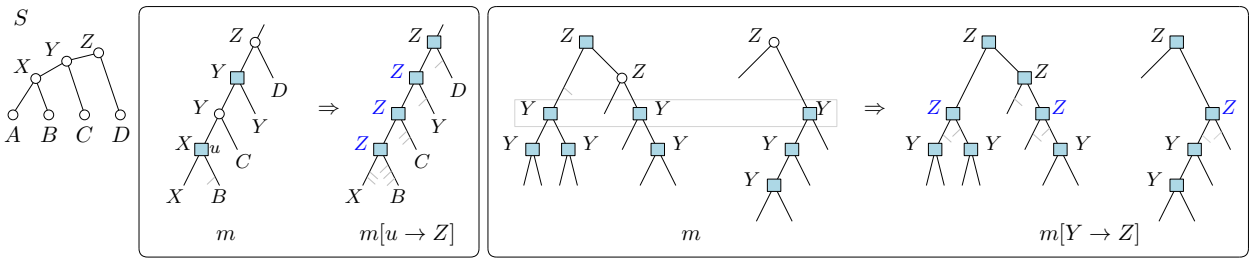

Figure 2: An illustration of up-moves and bulk up-moves. Left: the species tree S. Middle: a portion of a gene tree undergoing an up-move that remaps u from X to Z. To satisfy time-consistency, the parent and grand-parent of u must also be remapped to Z, which creates a chain of duplications in Z and additional losses. Right: a bulk up-move that takes the earliest segmental duplication in Y spanning two gene trees, which consists of the roots of the trees in $F_m(Y)$, and remaps them to the species Z.

tree in Figure 1 shows the advantage of the more complex type of up-moves. In that tree, we would like the A-duplication to be remapped to Y to save one duplication, but with simple moves we would need to remap X to Y first. However, that moves does not decrease the number of segmental duplications, but increases the number of losses, so it is strictly worse. The more complex remap from A to Y lets us find an improved reconciliation directly, without having to go through a worse reconciliation first.

Bulk up-moves. We may need to remap several genes upwards to save a segmental duplication. One way is to take the genes in the same segmental duplication in some species s, and move that duplication to an ancestor of s. For $s \in V(S)$ and $t \succ s$, the *bulk up-remapping from s to t* is another mapping $m[s \to t]$ in which, for every root u of the forest $F_m(s)$ in any order, we apply the up-move that remaps u to t.

Note that such bulk up-moves always reduce the height of $F_m(s)$, although it can increase the duplication height of other species. This is illustrated in Figure 2, where the roots of the Y duplication forest are all remapped to Z, thereby reducing the number of segmental duplications in Y. Also note that such a move is guaranteed to increase the number of losses. Therefore, they can only be advantageous if they reduce the height of $F_m(s)$ without increasing another height.

Down-moves. A down-move remaps $u \in V(G)$ to a species s below $m(u)$. Since $\mu(u)$ is the lowest possible, such an s must satisfy $\mu(u) \preceq s \prec m(u)$, and we may need to remap descendants of u to s to avoid time-inconsistencies. The *down-remapping of u to s* is another mapping $m[u \to s]$ in which

- $m[u \to s](w) = s$ for every descendant w of u such that $m(w) \succ s$ (including u);
- $m[u \to s](w) = m(w)$ for every other node of G.

Notice that the notation $m[u \to s]$ does not distinguish between up-moves and down-moves, but that distinction can be deduced from s. Also note that unlike up-moves, applying a down-move to u may make it change from a *Dup* node to a *Spec*, and even its parent p_u can become a *Spec*.

Bulk down-moves. As for up-moves, several genes may be remapped "downwards" at once to save a duplication (or more). Here, we take the deepest segmental duplication in s, and move it to a lower species.

For $s \in V(S)$ and $t \prec s$, the *bulk remapping from s to t* is another mapping $m[s \to t]$ that takes each deepest leaf u of $F_m(s)$, that is, every u of depth $H_m(s)$ in $F_m(s)$, and applies the down-remapping of u to t. If such a down-move is not possible because $\mu(u) \preceq t \prec s$ does not hold, then $m[s \to t]$ is *invalid*. Note that valid bulk down-remappings always reduce the number of losses, and hence they are advantageous if they do not increase the number of segmental duplications.

Exploration strategies and bottlenecks

We implemented standard *greedy* and *stochastic* strategies. In the greedy strategy, we (1) start from the lca-mapping μ; (2) compute the change in cost of every possible up, down, bulk-up, and bulk-down move; (3) apply the move that reduces the cost by a maximum amount, and repeat the second step. We stop when, at step 3, every move strictly increases the reconciliation cost.

The stochastic strategy uses a simulated annealing algorithm by assigning a probability from the Boltzmann distribution to each move, based on its change in cost. The process is as follows:

1. start with $m = \mu$ as the lca-mapping;
2. for each possible move that results in a new reconciliation m', compute $r = \delta(d_{m'} - d_m) + \lambda(l_{m'} - l_m)$, which is the change in reconciliation cost if we apply the move;
3. assign the weight $w_{m'} = e^{r/(kT)}$, where k is the Boltzmann constant and T is a user-specified *temperature* parameters that specifies how likely the most cost-reducing moves are taken;
4. obtain a vector of weights $(w_{m_1}, \ldots, w_{m_l})$ from the list of computed moves. Normalize the vector so that its entries sum to one, and use it as a probability distribution to sample an element from the vector, and apply the chosen move to obtain an updated reconciliation;
5. repeat Step 2 until convergence (i.e., no improvement in cost was made for a specified number of iterations), or until a specified maximum number of iterations is reached.

The main bottleneck of this procedure is Step 2, the computation of cost changes of each move. In order to perform as many iterations as possible, our goal is to compute all such entries as efficiently as possible. However, a purely naive algorithm would be too slow. Consider the set of all possible up-moves, for example. Let us note that every node $u \in V(G)$ must be mapped to an ancestor s of $\mu(u)$, and so there are $O(h(S))$ possibilities, and therefore $O(|V(G)|h(S))$ entries to compute. For each such entry, the naive algorithm would recalculate the forest heights in every gene tree in time $O(|V(G)| + |V(S)|)$ (which is actually difficult to avoid, because chains of remapping may affect several duplication heights). If we assume that $|V(G)| > |V(S)|$, we get a total time of $O(|V(G)|^2 h(S))$ to compute every entry, per iteration.

Since our goal is to scale to gene forests with thousands of trees, and thus tens of thousands of nodes, a quadratic dependency on $|V(G)|$ is not acceptable to perform large numbers of iterations. The complexity can be reduced by maintaining the forest heights per species in each gene tree separately, so that remappings only involve recalculations in the current tree, but the complexity per entry would still be linear in the size of that tree, plus the number of trees and $|V(S)|$. In the following, we aim for an amortized time of $O(1)$ per entry.

A2.1 Computing the change in cost of up-moves

We now describe in detail how the change in cost of each up-move can be computed efficiently. Consider a valid up-move $m[u \to s]$ that remaps u to $s \succ m(u)$. For any $t \in V(S)$, we denote $\Delta(u, s, t) = H_{m[u\to s]}(t) - H_m(t)$, which is the change in duplication height in species t after remapping u to s. Note that $\Delta(u, m(u), t) = 0$, since remapping u to $m(u)$ changes nothing. Then denote

$$\Delta(u, s) = \sum_{t \in V(S)} \Delta(u, s, t)$$

which is actually equal to $d_{m[u\to s]} - d_m$, the *total* change in duplication heights after remapping u to s.

We also need to account for losses, and denote by $\Lambda(u, s)$ the total change in the number of losses after remapping u to s. Recalling that l_m is the number of losses in m, we have

$$\Lambda(u, s) = l_{m[u\to s]} - l_m.$$

Changes in duplication costs

Let us first focus on $\Delta(u, s)$ entries first. We first show that if we calculate those in pre-order on G (i.e, parents are handled before children), then we can re-use the information from the parents and only require a constant number of entries.

Before we can get to the recurrences, we first show that only the changes in $m(u)$ and s need specific calculation, as *Dup* nodes in other species are the same whether we remap u to s or its parent p_u to s.

Lemma 1. *Let $u \in V(G)$ be a non-root node with parent p_u, let $s \in V(S)$ be an ancestor of $m(p_u)$, and let $t \in V(S) \setminus \{s, m(u)\}$. Then $F_{m[u\to s]}(t) = F_{m[p_u\to s]}(t)$, that is, the set of Dup nodes mapped to t is the same under $m[u \to s]$ or $m[p_u \to s]$.*

Proof. Assume as stated that $s \succeq m(p_u)$, and observe that for a gene forest node $w \neq u$, w is an ancestor of u with $m(w) \prec s$ if and only if w is an ancestor of p_u with $m(w) \prec s$, because u and p_u have the same ancestors except u itself. Let R_u (resp. R_{p_u}) be the ancestors of u (resp. p_u) such that $m(w) \prec s$, noting that $R_{p_u} = R_u \setminus \{u\}$. Note that R_u contains precisely the nodes that get remapped to s in $m[u \to s]$ (the R stands for "remapped"). Also note that R_{p_u} could be empty if $s = m(p_u)$.

Let $t \in V(S) \setminus \{s, m(u)\}$. Let $w \in F_{m[u\to s]}(t)$, so that w is a t-duplication under $m[u \to s]$. Then $w \notin R_u$, since such ancestors get remapped to s in $m[u \to s]$. Hence $m(w) = m[u \to s](w) = t$. Also, $w \notin R_{p_u}$, so it does not get remapped in $m[p_u \to s]$ as well, and thus $m[p_u \to s](w) = t$. We must also show that w is a *Dup* node under $m[p_u \to s]$. To see this, consider the children w_1, w_2 of w in G (which exist since *Dup* nodes are internal). If no child of w is in R_u, then no child of w is in R_{p_u}, implying that $m[u \to s](w_i) = m[p_u \to s](w_i)$ for $i = 1, 2$. Since *Dup* nodes are determined by the mapping of their children, we have that w is a *Dup* node under $m[p_u \to s]$ as well. It is also possible that one child w_i of w is in R_u (but not two, since R_u induces a path). Suppose without loss of generality that child w_1 of w is in R_u. If w_1 is also in R_{p_u}, we have $s = m[u \to s](w_1) = m[p_u \to s](w_1)$. The other child w_2 of w is not in R_u and does not change its map, and we get the same conclusion. If that child w_1 is not in R_{p_u} but is in R_u, then $w_1 = u$ is the only possibility. Hence $w = p_u$. But this cannot be, because $p_u \notin R_{p_u}$ is only possible if $s = m(p_u)$, whereas we have $m(w) = t$. Hence, we deduce that w is a *Dup* node mapped to t if we remap p_u to s, and thus $F_{m[u\to s]}(t) \subseteq F_{m[p_u\to s]}(t)$.

Conversely, let $w \in F_{m[p_u\to s]}(t)$ be a *Dup* node under $m[p_u \to s]$ such that $m[u \to s](w) = t$. Similarly as before, $w \notin R_{p_u}$ and thus $m(w) = t$. Moreover, $t \neq m(u)$ implies that $w \neq u$. Therefore, $w \notin R_u$, and thus $m[u \to s](w) = m(w) = t$. As before, consider the children w_1, w_2 of w. If none of them is in R_{p_u}, then none of them is in R_u (since $w \neq p_u$), and if one of them is in R_{p_u}, that child is also in R_u. Thus the children of w has the same map and it is also a *Dup* node mapped to t under $m[u \to s]$. Hence $F_{m[p_u\to s]}(t) \subseteq F_{m[u\to s]}(t)$. □

Lemma 1 allows us to simplify the computation of changes in cost.

Lemma 2. *Let $u \in V(G)$ and let $s \in V(S)$ be an ancestor of $m(u)$. If u is a root, or if u has parent p_u satisfying $s \prec m(p_u)$, then*

$$\Delta(u, s) = \Delta(u, s, s) + \Delta(u, s, m(u)).$$

Otherwise, u has a parent p_u satisfying $s \succeq m(p_u)$, in which case

$$\begin{aligned}\Delta(u, s) = \Delta(p_u, s) \ &- \Delta(p_u, s, s) - \Delta(p_u, s, m(u)) \\ &+ \Delta(u, s, s) + \Delta(u, s, m(u)).\end{aligned}$$

Proof. Suppose that u is a root of G. Since up-moves can only remap ancestors of u, it follows that u is the only node that gets remapped in $m[u \to s]$. Hence, aside from u, all the nodes have the same map as well as their children, and therefore *Dup* nodes mapped to $t \notin \{m(u), s\}$ are the same in either m or $m[u \to s]$, and thus $\Delta(u, s, t) = 0$ for each such t. Therefore, the summation for $\Delta(u, s)$ can only be affected by $\Delta(u, s, s)$ and $\Delta(u, s, m(u))$, as stated.

If u is not a root but $s \prec m(p_u)$, it is still true that all nodes except u keep the same map under $m[u \to s]$, including p_u. Moreover, the children of all such nodes have the same map, except p_u whose child u has changed. Thus all *Dup* nodes in $t \neq s, m(u)$ are the same before and after, except possibly p_u. If p_u is a *Dup* under m, it is easy to check that this also holds under $m[u \to s]$. If p_u was a *Spec*, then $m(p_u)$ has children x_1, x_2 and $m(u) \preceq x_1$ (and the other child descends from x_2). Since $m(u) \preceq s \prec m(p_u)$, we have $s \preceq x_1$ as well and p_u is still a *Spec* under $m[u \to s]$. Hence, duplication forests in t are also preserved in this case. It again follows that only $\Delta(u, s, s)$ and $\Delta(u, s, m(u))$ contribute to $\Delta(u, s)$.

So assume that $s \succeq m(p_u)$. By Lemma 1, for $t \notin \{m(u), s\}$, the *Dup* nodes in t are identical under $m[u \to s]$ and $m[p_u \to s]$. It follows that $H_{m[u\to s]}(t) = H_{m[p_u\to s]}(t)$, and therefore that $\Delta(u, s, t) = \Delta(p_u, s, t)$. We

may conclude that

$$\begin{aligned}\Delta(u,s) &= \sum_{t\in V(S)} \Delta(u,s,t)\\ &= \Delta(u,s,s) + \Delta(u,s,m(u)) + \sum_{t\in V(S)\setminus\{s,m(u)\}} \Delta(u,s,t)\\ &= \Delta(u,s,s) + \Delta(u,s,m(u)) + \sum_{t\in V(S)\setminus\{s,m(u)\}} \Delta(p_u,s,t)\\ &= \Delta(u,s,s) + \Delta(u,s,m(u)) + \Delta(p_u,s) - \Delta(p_u,s,s) - \Delta(p_u,s,m(u))\end{aligned}$$

which, after reordering the terms, yields the claimed equality. □

As a consequence of Lemma 2, using a top-down approach, one can obtain $\Delta(u,s)$ by reusing $\Delta(p_u,s)$ and computing only four additional values, at most. Still, these are not trivial to obtain in $O(1)$ time, and we need auxiliary data structures to achieve this efficiently. Later on we discuss how to maintain this information while still achieving amortized time $O(1)$ per entry.

- For a *Dup* node $u \in V(G)$ under m, denote by $h_m(u)$ the height of u in $F_m(m(u))$. That is, $h_m(u)$ is the number of segmental $m(u)$-duplications that contain u or one of its descendants.
- For $s \in V(S)$ and integer k, denote by $B[s,k]$ the set of nodes of G whose height is k in $F_m(s)$. That is, $B[s,k] = \{v \in V(G) : m(v) = s, h_m(v) = k\}$. Note that this set could be empty, which occurs in particular if $k = 0$.
- For $u \in V(G)$ and $s \succeq m(u)$, define $c_m(u,s)$ as the number of ancestors w of u satisfying $s \succeq m(w)$. This quantity is the length of the duplication chain that ends at u if we remap it to s.

Assuming we compute $\Delta(u,s,t)$ values in pre-order, the value of $\Delta(p_u,s,s)$ needed in Lemma 2 is easy to obtain: if $m(p_u) = s$, then it is 0, and otherwise that value is necessary to compute $\Delta(p_u,s)$, and so it will have been computed and stored when handling p_u. The next lemma states how the other three required values can be calculated.

Lemma 3. *Let $u \in V(G)$ and let $s \in V(S)$ be a strict ancestor of $m(u)$. Then the following holds:*

1. $\Delta(u,s,s) = \max(c_m(u,s) - H_m(s), 0)$.
2. *If u has a parent p_u satisfying $s \succeq m(p_u)$, then*

$$\Delta(p_u,s,m(u)) = \begin{cases} 0 & \text{if } m(u) \prec m(p_u)\\ \Delta(p_u,s,m(p_u)) & \text{otherwise.}\end{cases}$$

3. *for the value of $\Delta(u,s,m(u))$, the following cases arise:*
   (a) *if u is a root or $m(u) \neq m(p_u)$, then*

$$\Delta(u,s,m(u)) = \begin{cases} -1 & \text{if } B[m(u), H_m(m(u))] = \{u\}\\ 0 & \text{otherwise}\end{cases}$$

   (b) *if $m(u) = m(p_u)$, then let $h' = H_{m[p_u\to s]}(m(u))$, which is the height in $m(u)$ if we remap p_u to s. Then*

$$\Delta(u,s,m(u)) = \Delta(p_u,s,m(u)) + \begin{cases} 0 & \text{if } h' \neq h_m(u)\\ -1 & \text{if } h' = h_m(u) \text{ and } B[m(u),h'] = \{u\}\\ 0 & \text{if } h' = h_m(u) \text{ and } B[m(u),h'] \neq \{u\}\end{cases}$$

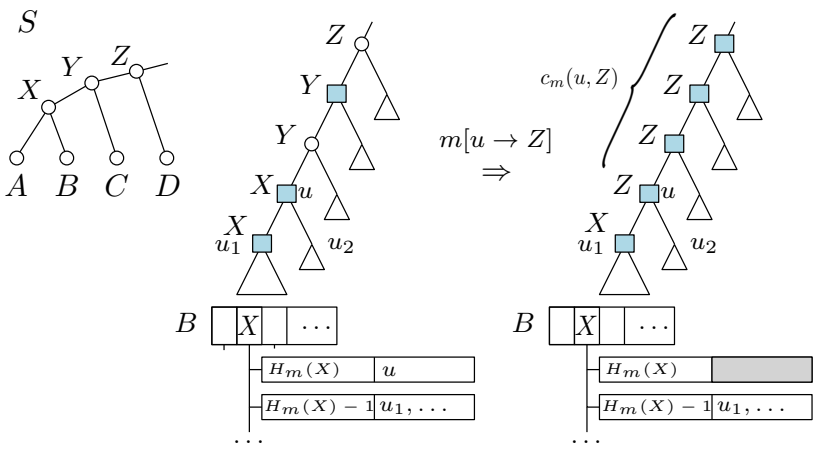

Figure 3: Illustration of Cases 1 and 3.(a) from Lemma 3. For Case 1, $\Delta(u, s, s) = \Delta(u, Z, Z)$, and the length of the chain of Z-duplications ending at u after remapping is $c_m(u, Z) = 4$. Either this chain is the maximum height in Z-duplications after remapping, or some other untouched subtree still has the maximum height. For Case 3.(a), $\Delta(u, s, m(u)) = \Delta(u, Z, X)$, a representation of B is shown at the bottom. If $B[s, H_m(s)] = B[X, H_m(X)]$ only contains u, then after remapping u that entry becomes empty and a child u_1 of u is now a root of a highest X-duplication subtree (of height $H_m(X) - 1$).

Proof sketch. The complete proof is somewhat tedious, but the main ideas are relatively simple, so we first sketch them.

For 1., one can show that the longest s-duplication path in $m[u \to s]$ either starts at u and goes upwards, in which case the length of the path is $c_m(u, s)$, or the longest s-duplication path was already present and there is 0 change. For 2., if $m(u) \prec m(p_u)$, then remapping p to s does not affect any node mapped to $m(u)$, and otherwise only $m(u) = m(p_u)$ is possible. For 3.(a), the height in $m(u)$ cannot increase, and the only way to decrease it is if u was the sole root of a maximum-height tree in $F_m(m(u))$. This holds if and only if $B[m(u), H_m(m(u))] = \{u\}$, and the height can only drop by 1 since the children of u keep the same map.

For 3.(b), we rely on h', the height in $m(u)$ if we remap p_u to s. In $m[p_u \to s]$, one of three things can happen: u is not the root of a maximum height subtree in $m(u)$ (in which case $h' \neq h_m(u)$ and $\Delta(u, s, m(u)) = \Delta(p_u, s, m(u))$); u is the sole root of a maximum height subtree, in which case $h' = h_m(u)$ and $B[m(u), h'] = \{u\}$ and now remapping u decreases the duplication height; or there are multiple roots of maximum height subtrees, in which case there is no change in heights. The expression for $\Delta(u, s, m(u))$ considers these three cases. □

Proof. For the detailed proof, we argue the three values separately.

Case 1: $\Delta(u, s, s)$. First note that under $m[u \to s]$, u is a *Dup* node in s and none of its children is mapped to s. Consider the path P that starts at u and, going upwards, includes the ancestors of u mapped to s in $m[u \to s]$. Then P includes u, plus any ancestor v of u that got remapped (so $s \succ m(v)$), plus any ancestor v such that $s = m(v) = m[u \to s](v)$. Thus by definition, the number of nodes of P is equal to $c_m(u, s)$.

We note that $H_{m[u \to s]}(s) \geq H_m(s)$, because the $m[u \to s]$ has strictly more *Dup* nodes mapped to s than m, namely because of u. Now let Q be a root-to-leaf path of maximum length in $F_{m[u \to s]}(s)$, so that the number of nodes in Q is $H_{m[u \to s]}(s)$. If Q contains u, then $P = Q$ and $\Delta(u, s, s) = c_m(u, s) - H_m(s)$. Since this is at least 0, our max expression is correct in this case.

If Q does not contain u, then $c_m(u, s) \leq H_{m[u \to s]}(s)$. We also note that Q could not contain an ancestor v of u such that $s \succ m(v)$. Indeed, such a node v has no child mapped to s under m, and under $m[u \to s]$ it only has a child mapped to s that leads to u. Thus any maximal *Dup* path in s containing such a v ends in u. Thus, Q does not contain any node that got remapped from m to $m[u \to s]$, and so that *Dup* path in s was also present under m. We thus get $H_{m[u \to s]}(s) \leq H_m(s)$, and because the duplication height in s cannot decrease, equality holds and $\Delta(u, s, s) = 0$. In that case, $c_m(u, s) \leq H_{m[u \to s]}(s)$ implies that our max expression will return 0. In all cases $\Delta(u, s, s)$ is correct.

Case 2: $\Delta(p_u, s, m(u))$. If $m(u) \prec m(p_u)$, then notice that no ancestor of p_u is mapped to $m(u)$. Therefore,

remapping p_u to s, which is an ancestor of $m(p_u)$ and thus of $m(u)$, cannot affect the duplication height in $m(u)$. In this case, $\Delta(p_u, s, m(u)) = 0$ as stated. If instead $m(u) \succeq m(p_u)$, then by time-consistency, only $m(u) = m(p_u)$ is possible, hence $\Delta(p_u, s, m(u)) = \Delta(p_u, s, m(p_u))$.

Case 3: $\Delta(u, s, m(u))$. We note that the height of the duplication forest in $m(u)$ cannot increase when remapping u to s, since this remapping reduces the number of nodes mapped to $m(u)$. Suppose that u is a root or $m(u) \neq m(p_u)$. Suppose that $B[m(u), H_m(m(u))] = \{u\}$. Then there is only one subtree of $F_m(m(u))$ of height $H_m(m(u))$ and its root is u. When we remap u to s, the largest possible height in $m(u)$ is then $H_m(m(u)) - 1$, which is achieved by one of the children of u. In this case, the duplication height in $m(u)$ is reduced by 1, and so putting $\Delta(u, s, m(u)) = -1$ in this case is correct.

Assume instead that no entry $B[m(u), H_m(m(u))]$ exists, which can occur if there are no $m(u)$-duplications. Then remapping u from $m(u)$ to s cannot create $m(u)$-duplications, and putting $\Delta(u, s, m(u)) = 0$ is correct. So assume that $B[m(u), H_m(m(u))]$ contains some node v other than u. Then $m(u) \neq m(p_u)$ implies that v is not p_u nor any of its ancestors. Thus v is incomparable with u, and thus the duplication subtree rooted at v is still present in $m[u \to s]$. Hence the height has not reduced, and since it cannot decrease, $\Delta(u, s, m(u)) = 0$ is the correct value to apply in this case.

Next suppose that $m(u) = m(p_u)$. Let $h' = H_{m[p_u \to s]}(m(u))$ be the height of the duplication forest in $m(u)$ under $m[p_u \to s]$. If $h' \neq h_m(u)$, then some subtree of height h' in $m(u)$ is present under $m[p_u \to s]$ and is not rooted at any ancestor of p_u (because those are mapped to s or above). It is also not rooted at u since $h' \neq h_m(u) = h_{m[p_u \to s]}$. It is therefore not rooted at an ancestor of u and so this subtree is also present under $m[u \to s]$. It follows that $H_{m[u \to s]}(m(u)) \geq h'$. On the other hand, $H_{m[u \to s]}(m(u)) \leq h'$, since $m[u \to s]$ has strictly less nodes mapped to $m(u)$ than $m[p_u \to s]$. Thus $H_{m[u \to s]}(m(u)) = h'$, and we get

$$H_{m[u \to s]}(m(u)) - H_m(m(u)) = h' - H_m(m(u)) = \Delta(p_u, s, m(u))$$

which is the correct value we give to $\Delta(u, s, m(u))$.

So assume that $h' = h_m(u)$. Then under $m[p_u \to s]$, u is the root of a *Dup* tree in $m(u)$ of maximum height. If $B[m(u), h'] = \{u\}$, then u is the only subtree with height h' under m. Noting that remapping p_u and some ancestors to s cannot create new *Dup* subtrees in $m(u)$ with that same height, it follows that u is also the only subtree with that height under $m[p_u \to s]$ as well. Since the only difference between $m[u \to s]$ and $m[p_u \to s]$ is that u is remapped to s, the height of that subtree is one less under $m[u \to s]$ than under $m[p_u \to s]$ (achieved by one of the children of u, as before). Thus putting $\Delta(u, s, m(u)) = \Delta(p_u, s, m(u)) - 1$ is correct in this case.

Finally, suppose that $B[m(u), h']$ contains some node v other than u. Because under m, u and v both root a *Dup* subtree in $m(u)$ of the same height h', v cannot be an ancestor or descendant of u and is thus incomparable with u. Hence, the v subtree is present under either $m[p_u \to s]$ or $m[u \to s]$. Thus, $H_{m[u \to s]}(m(u)) \geq h'$ and, using the previous argument, $H_{m[u \to s]}(m(u)) \leq h'$. This implies equality and then $\Delta(u, s, m(u)) = \Delta(p_u, s, m(u))$. □

Changes in loss costs

Recall that $\Lambda(u, s) = l_{m[u \to s]} - l_m$ is the total change in the number of losses after remapping u to s. For a mapping m', we define:

$$i_{m'}(u) = \begin{cases} 0 & \text{if } u \text{ is duplication under } m' \\ 2 & \text{if } u \text{ is speciation under } m' \end{cases}$$

Assuming that s is a strict ancestor of $m(u)$, we also define an auxiliary value $\lambda(u, s)$ as the total change in the number of losses between u and its children u_l, u_r, that is:

$$\begin{aligned} \lambda(u, s) = \; & dist(s, m(u_l)) - dist(m(u), m(u_l)) + \\ & dist(s, m(u_r)) - dist(m(u), m(u_r)) + i_m(u) \end{aligned}$$

The reason for adding the $i_m(u)$ term is that if u is a *Spec* under m, it will be a *Dup* under $m[u \to s]$ and require additional loss on each child branch.

Lemma 4. *Let $u \in V(G)$ and let $s \in V(S)$ be a strict ancestor of $m(u)$. Then:*

1. *If u is a root, then $\Lambda(u,s) = \lambda(u,s)$.*
2. *if u has a parent p_u and $s \preceq m(p_u)$, then*

$$\Lambda(u,s) = \lambda(u,s) + \; dist(m(p_u),s) - dist(m(p_u),m(u)) + \\ i_m(p_u) - i_{m[u\to s]}(p_u).$$

3. *if u has a parent p_u and $s \succ m(p_u)$, then*

$$\Lambda(u,s) = \Lambda(p_u,s) + \lambda(u,s) - dist(s,m(u))$$

Proof. If u is a root, the only edges of G on which the number of losses changes are those adjacent to u, which are the edges leading to the children of u. In that case, the expression $\lambda(u,s)$ is easily seen to represent the number of changes in losses.

Suppose that u is not a root but $s \preceq m(p_u)$. Then p_u does not get remapped, and so the changes in number of losses only occur on the edges incident to u. This includes the edges leading to the children of u, which again are handled by $\Lambda(u,s)$, but also the edge (p_u,u). If p_u is a *Dup* under m, then it must be a *Dup* under $m[u \to s]$ as well since $s \succ m(u)$. In that case, the number of losses on (p_u,u) under m is $dist(m(p_u),m(u))$ and under $m[u \to s]$ it is $dist(m(p_u),s)$. Since $i_m(p_u) = i_{m[u\to s]}$ in this case, our value of $\Lambda(u,s)$ is correct in this case.

If p_u is a *Spec* under m, it could be a *Spec* or a *Dup* under $m[u \to s]$. Suppose it remains a *Spec*. Then the difference in the number of losses is

$$dist(m(p_u),s) - 1 - (dist(m(p_u),m(u)-1))$$

and our value of $\Lambda(u,s)$ is again correct. If p_u becomes a *Dup*. Let v be the child of p_u other than u. Note the change from *Spec* to *Dup* adds an extra loss on (p_u,v). Then the difference in the number of losses is

$$dist(m(p_u),s) + 1 - (dist(m(p_u),m(u)-1))$$

and, since $i_m(p_u) - i_{m[u\to s]}(p_u) = 2$, our value is correct.

Finally, assume that $s \succ m(p_u)$. In this case, p_u and some of its ancestors gets remapped. In this case, one can see this remapping u to s in two phases: first remap p_u to s, which introduces $\Lambda(p_u,s)$ losses, then remap u to s. Note that no losses remain on the (p_u,s) edge after the second phase, so the losses on that edge counted in the first phase need to be subtracted. Because p_u is a *Dup* after remapping it to s, there are $dist(s,m(u))$ losses on (p_u,s) to subtract. The only other change introduced by this second phase are the $\lambda(u,s)$ losses created on the children of u. This justifies our expression for $\Lambda(u,s)$. □

Wrapping up: complexity of computing every entry

We have gathered all the necessary steps to compute $\Delta(u,s)$ for every $u \in V(G)$ and $s \succ m(u)$. To achieve amortized $O(1)$ time per entry, we must argue that the set of all entries can be computed in total time $O(|V(G)|h(S))$. Recall that we need access to values $h_m(u), B[s,k]$, $c_m(u,s)$, and $H_m(s)$. We first assume that we have computed and stored the lca-mapping, that we may query in time $O(1)$ whether a node is an ancestor of the other, and that we have access to $dist_S(x,y)$ in $O(1)$ time (this pre-processing is only needed once before the first iteration and can be achieved in time $O(|V(G)| + |V(S)|)$, detailed omitted here).

To compute the necessary values, at the start of an iteration we proceed as follows:

1. the values $h_m(u)$ can be calculated in time $O(|V(G)|)$ through a post-order traversal of G. If u is a speciation, then $h_m(u) = 0$ because it is not even part of $F_m(m(u))$. If u is a duplication, then $h_m(u) = 1 + \max_{v\in C(u)} h_m(v)$, where $C(u)$ is the set of children of u mapped to $m(u)$ (the maximum is defined as 0 if $C(u)$ is empty). Note that checking whether u is a *Spec* or *Dup* can be done in $O(1)$ time by comparing m with μ.

2. for the the sets $B[s,k]$, assuming $O(1)$ time for searching and inserting into hash tables, each $B[s,k]$ can be represented as a set implemented as a hash table. Then B can be filled in time $O(|V(G)|)$ by adding each u to $B[m(u), h_m(u)]$. Note that in doing so, for each s we may store the maximum k for which $B[s,k]$ is non-empty without any computation overhead.

3. the values $H_m(s)$ can easily be obtained while computing the previous step, as $H_m(s)$ is the maximum k such that $B[s,k]$ is non-empty.

4. for the values $c_m(u,s)$, we compute them in time $O(|V(G)|h(S))$ through a pre-order traversal of G, where for each $u \in V(G)$ we iterate over every $s \succeq \mu(u)$, noting that constant time per c_m entry can be done using the fact that $c_m(u,s) = 1$ if u is a root of G or $s \prec m(p_u)$, and otherwise $c_m(p_u,s) + 1$.

Overall, the pre-processing step takes time no more than $O(|V(G)|h(S))$. Once this is done, we can compute $\Delta(u,s)$ and $\Lambda(u,s)$ for each u and each s. By computing those in pre-order, one can check that the expressions from Lemma 2 and Lemma 3 can all be computed in time $O(1)$, if we have access to the entries of p_u when arriving at a node u. We thus get the following.

Theorem 1. *Given a gene forest G, a species tree S, and a reconciliation m, the value of each possible entry for up-moves $\Delta(u,s)$ and $\Lambda(u,s)$ can be computed in total time $O(|V(G)|h(S))$.*

Remark. The above assumes a worst-case situation where the auxiliary data structures and each Δ and Λ values must be recalculated after each iteration, implying a time of $O(|V(G)|h(S))$ per iteration. However, applying an up-move only affects one gene tree, so the data structures can be updated efficiently after the first iteration. For instance, when applying a remapping from u to s, the $c_m(u,s)$ and $h_m(u)$ entries only need to be updated for the nodes of the tree containing u, and likewise updating the $B[s,k]$ entries only needs to relocate the nodes in that same tree. Moreover, applying an up-move can only remap $h(G)$ genes, and thus affect the duplication height of $O(h(G))$ species, meaning that recalculating all $\Delta(u,s)$ may not be necessary for every s. Developing a formal analysis of the optimal time-per-update is a complicated matter, and we leave it for future work.

A2.2 Computing the change in cost of down-moves

The computation of down-moves uses similar ideas: pre-process G and S in time $O(|V(G)|h(S))$ to obtain values of interest, and use them to compute every $\Delta(u,s)$ entry in constant time. Because down-moves remap descendants, we calculate the entries in a post-order fashion and use the information from the children through the iterations. One major difference is that for a gene tree node u with children u_1, u_2, the computations performed at u_1 did not "see" the u_2 subtree, and vice-versa. For example, if both the u_1 and u_2 subtrees contain a maximum height subtree in some species t, applying a down-move on either u_1 or u_2 does not alter the duplication height in t. However, a down-move on u may destroy both of these subtrees, a fact that is difficult to recover from the local information stored at u_1 or u_2. Our pre-processing must therefore store the largest t-duplication subtree not contained below u, for each u and each t. It is challenging to obtain this in time $O(|V(G)|h(S))$, but we show that it can be done, and after that we can apply techniques that are similar to up-moves.

We now formalize these ideas, starting with the pre-processing. Since the high-level ideas are similar to up-moves, our proofs are more expeditive, and the recurrences needed to the computation are provided inside the proofs.

For $u \in V(G)$, we write $[u]$ to denote the set of descendants of u (including itself). For a duplication forest $F_m(s)$, we use $F_m(s) - [u]$ to denote the forest obtained from $F_m(s)$ and removing all of its nodes that belong to $[u]$. If a node of $[u]$ is not in $F_m(s)$, it has no effect. For a gene forest node u and species node s, we let $g(u,s)$ be the height of the forest $F_m(s) - [u]$. These values are crucial to compute down-moves efficiently, and we show that they can be handled in a pre-processing step.

Lemma 5. *There is a pre-processing algorithm that runs in time $O(|V(G)|)$ and that allows accessing $g(u,s)$ in time $O(1)$, for any $u \in V(G)$ and $s \in V(S)$.*

Proof. Let u be a gene tree node and s a species tree node. We describe the process of computing all $g(u,s)$ entries and analyze the complexity afterwards. If $H_m(s) = 0$, then $g(u,s)$ is clearly 0 since removing vertices

$[u]$ from the empty $F_m(s)$ cannot increase the number of segmental duplications. So assume that $H_m(s) > 0$. Let v_s be any element of $B[s, H_m(s)]$, so that v_s is the root of a maximum-height tree T in $F_m(s)$. Let v'_s be a deepest leaf of T. Notice that if u is incomparable with v'_s or if $u \prec v'_s$, then we have $g(u, s) = H_m(s)$, since removing $[u]$ from $F_m(s)$ does not affect the longest path of the duplication subtree rooted at v_s. Given u and v'_s, these cases can be handled in $O(1)$ time by checking ancestry relations between u and v'_s. It now suffices to consider the case where $v'_s \preceq u$.

If $u = v_s$, we can obtain $g(u, s) = g(v_s, s)$ by removing from the B data structure every s-duplication that descends from v_s (by doing this in pre-order starting from v_s, each removal takes constant time). As a result, $g(v_s, s)$ is the highest k such that the modified $B[s, k]$ is non-empty, which can then be obtained in constant time. We store this value and then reinsert the removed elements from B to retrieve its original state.

For u such that $v'_s \preceq u \prec v_s$, then either the longest duplication path in $F_m(s) - [u]$ uses p_u, or not. If p_u is used, then $g(u, s) = dist_T(v_s, p_u) + 1 + h_{F_m(s)}(u')$, where u' is the sibling of u, i.e., the child of p_u other than u. The latter height is 0 if u' is not a duplication mapped to s, or otherwise it is $h_m(u')$, both cases verifiable in constant time. If p_u is not used, then the longest duplication path in $F_m(s) - [u]$ does not use the sibling u' either, since if this was the case this path could be extended to include p_u (which is mapped to s since it is between v'_s and v_s). Therefore, this path does not use p_u nor any of its descendants, meaning that it is also in $F_m(s) - [p_u]$, and thus $g(u, s) = g(p_u, s)$. Either case can be calculated in constant time (if we process the nodes of the $v_s - v'_s$ path top-down and store the $g(u, s)$ values in order), and so we take the maximum between $dist_T(v_s, p_u) + 1 + h_{F_m(s)}(u')$ and $g(p_u, s)$.

Finally, it remains to compute $g(u, s)$ for $u \succ v_s$. These values are computed starting from the parent of v_s and going upwards. For each encountered u, we remove from the B data structure all the s-duplications that descend from u that have not been removed so far, and put $g(u, s)$ as the maximum k such that $B[s, k]$ is non-empty. We do not reinsert the s-duplications after processing u — instead we process the parent of u and remove the new s-duplications that need removal. Only once we have processed the root we reinsert all the removed s-duplications. This can be achieved in time proportional to the number of s-duplications as follows. First we remove every s-duplication that descends from v_s (again). Then, the other elements of B to remove at each u can be enumerated quickly, as they are the s-duplications w satisfying $lca(w, v_s) = u$. The nodes that satisfy this condition can be computed and stored in a pre-processing step by iterating over all the s-duplications w, and adding w to a list of nodes associated with $lca(v_s, w)$ (if $w \notin V(T)$, we ignore it).

Overall, if we let n_s denote the number of s-duplications under m, obtaining every $g(u, s)$ entry requires the following:
- we find v_s and v'_s in time $O(n_s)$ by going through the s-duplications in $F_m(s)$ (if multiple choices arise, choose arbitrarily);
- nodes u incomparable to v'_s or below v'_s take $O(1)$ time. There is nothing to compute or store for this case, as we check for incomparability or descendance when $g(u, s)$ is queried;
- we put $u = v_s$, and spend time $O(n_s)$ to remove and reinsert entries in B;
- for $v'_s \preceq u \prec v_s$, the recurrence based on $g(p_u, s)$ given above takes time $O(1)$, and there are $O(n_s)$ such entries;
- for $u \succ v'_s$, we delete entries from B. Each entry to delete takes $O(1)$ time to find and delete, using the lca strategy from above. Since we go upwards from v_s, each s-duplication in T gets deleted at most once during the upwards traversal, so computing $g(u, s)$ for the ancestors of v_s takes total time $O(n_s)$.

Since each of the five above steps takes time $O(n_s)$, it follows that for each s, obtaining every $g(u, s)$ takes time $O(n_s)$. Iterating over every $s \in V(S)$, the total time is proportional to the sum of the n_s's, which is $O(|V(G)|)$. □

We now assume that we have applied a pre-processing step that allows access to any $g(u, s)$ value in time $O(1)$.

Theorem 2. *Given a gene forest G, a species tree S, and a reconciliation m, the value of each possible entry for down-moves $\Delta(u, s)$ and $\Lambda(u, s)$ can be computed in total time $O(|V(G)|h(S))$.*

Proof. Denote by μ the lca-mapping between G and S, which we again assume has been computed in a pre-processing step. First notice that any gene tree node can only be remapped to a species s with

$\mu(u) \preceq s \prec m(u)$. Hence there are $O(h(S))$ possibilities, and thus $O(|V(G)|h(S))$ entries to compute. We calculate the values $\Delta(u,s)$ for each u in post-order, and for each possible s in a top-down manner (so, starting with s as the chilld of $m(u)$ and going downwards to $\mu(s)$.

Let $u \in V(G)$ and $s \in V(S)$ with $\mu(u) \preceq s \prec m(u)$. We assume that u has a parent p_u and sibling u' (the child of p_u other than u). If u is a root of G, if suffices to ignore the cases that include p_u in the rest of the proof.

Consider the possible species t for which the set of t-duplications could possibly change in $m[u \to s]$. The obvious cases are $t = s$ and $t = m(u)$. Aside those, a gene tree node could incur change because it gets remapped, or because it was a duplication and became a speciation (we let the reader verify that down-moves cannot turn speciations into duplications). In $m[u \to s]$, the only nodes that get remapped are descendants of u mapped to a node on the path between s and $m(u)$, including $m(u)$ but excluding s. In other words, t-duplications may get remapped, where $s \prec t \preceq m(u)$. The nodes that can become a speciation are descendants of u that get remapped to s, as the previous case, or p_u could become a speciation. In the latter case, s and $m(u')$ must be incomparable and $m(p_u) = lca_S(s, m(u'))$. Note that $m(u)$ must be on the path between $m(p_u)$ and s, and if $m(u)$ were a strict descendant of $m(p_u)$, then p_u would be a *Spec* under μ. Thus we must have $m(p_u) = m(u)$, as p_u is assumed to be a *Dup*. Hence only an $m(u)$-duplication could be lost to a *Spec* in that latter case.

To summarize, only the duplication heights that can change are those in species on the $s - m(u)$ path, inclusively. Thus we have

$$\Delta(u,s) = \Delta(u,s,s) + \Delta(u,s,m(u)) + \sum_{s \prec t \prec m(u)} \Delta(u,s,t).$$

We argue that each of the three terms (counting the summation as a single term) can be handled in constant time.

For the summation, notice that for any t with $s \prec t \prec m(u)$, we have $\Delta(u,s,t) = g(u,t) - H_m(t)$, because in $m[u \to s]$, there is no t-duplication above u, and every t-duplication that descends from u is remapped to s. Thus, the remaining height in t-duplications is the same as in $F_m(t) - [u]$. We would like to obtain the term $\sum_{s \prec t \prec m(u)} \Delta(u,s,t)$ in constant time. It is 0 when $s = m(u)$ or when s is a child of $m(u)$, since there is no t to enumerate. Otherwise, consider t with $s \prec t \prec m(u)$, and the parent p_s of s in S. If $p_s \prec t \prec m(u)$ also holds, then by the previous paragraph, $\Delta(u,s,t) = g(u,t) - H_m(t) = \Delta(u,p_s,t)$. That is, there is equality because whether we remap u to s or p_s, the same set of t-duplications are remapped. We get that

$$\sum_{s \prec t \prec m(u)} \Delta(u,s,t) = \Delta(u,s,p_s) + \sum_{p_s \prec t \prec m(u)} \Delta(u,p_s,t)$$

The value $\Delta(u,s,p_s)$ is $g(u,p_s) - H_m(p_s)$, as argued before (this applies because we assume that s is not $m(u)$ or its child, and thus $p_s \neq m(u), s$)). Therefore, as we process the values of $\Delta(u,s)$ for each s top-down, at each s we can store the value of the summation $\sum_{s \prec t \prec m(u)} \Delta(u,s,t)$. When processing the next s, we have access to the value of $\sum_{p_s \prec t \prec m(u)} \Delta(u,p_s,t)$ in time $O(1)$. We can then add the term $\Delta(u,s,p_s)$ to it and store it, adding time $O(1)$. Therefore, the summation part adds only $O(1)$ overhead.

To compute $\Delta(u,s,m(u))$, we consider two cases. If $m(p_u) \neq m(u)$, then no ancestor of u is a $m(u)$-duplication. Therefore, the only $m(u)$-duplications that get removed descend from u, and it follows that $\Delta(u,s,m(u)) = g(u,m(u))$. Assume that $m(p_u) = m(u)$. If p_u remains a $m(u)$-duplication in $m[u \to s]$, then again only $m(u)$-duplications below u disappear, and $\Delta(u,s,m(u)) = g(u,m(u))$ [[MINUS $H_m(m(u))$??]]. Finally, if p_u is a speciation in $m[u \to s]$, then u' is certainly not also mapped to $m(u)$. Thus, the $m(u)$-duplications that disappear in $m[u \to s]$ all descend from p_u, in which case $\Delta(u,s,m(u)) = g(p_u,m(u))$. Note that checking which case applies can be done in time $O(1)$, and so computing $\Delta(u,s,m(u))$ also takes constant time.

It only remains to compute $\Delta(u,s,s)$. We need one last pre-processed data structure. For each $w \in V(G)$ such that w can be remapped to s, define $c'_m(w,s)$ as the height of the s-duplication subtree rooted at w in $m[w \to s]$ (where $s = m(w)$ is possible). If w is a speciation in $m[w \to s]$, then $c'_m(w,s) = 0$. Otherwise w is an s-duplication in $m[w \to s]$. In that case, $c'_m(w,s) = 1 + \max_{w'} c'_m(w',s)$, where w' iterates over the children w' of w such that $m(w') \succeq s$ (if no such child exists, $c'_m(w,s) = 1$). One can see that the set of all

c'_m entries can be computed in total time $O(|V(G)|h(S))$ in a bottom-up fashion. Then we get

$$\Delta(u, s, s) = \max(c'_m(u, s) - H_m(s), 0)$$

because either the highest s-duplication subtree becomes rooted at u, or some other subtree already present under m takes that role (this works because the height of the largest subtree cannot decrease).

To sum up, we have decomposed the computation of $\Delta(u, s)$ into a sum of three elements, each obtainable in time $O(1)$ after pre-processing time $O(|V(G)|h(S))$. This takes care of the changes in duplication costs.

For changes in loss costs, the computation is easier and we only provide the high-level idea. We apply a post-order processing and compute $\Lambda(u, s)$, where u has children u_1, u_2. First consider $\ell = \sum_{u'} \Lambda(u', s)$, where u' iterates over the children u' of u such that $m(u') \succeq s$. Since branching incident to a vertex descending from u' children are disjoint, ℓ counts the change in the number of losses when each such child gets mapped to s. We only need to alter ℓ according to the number of losses incident to u when it gets remapped to s. This requires subtracting losses counted in ℓ incident to uu' branches and adding new losses created between u and its parent p_u, if present. We also need to check whether p_u and/or u becomes a *Spec* and adjust the counts accordingly. Since this computation relies on the $\Lambda(u', s)$ at the children of u, and on losses on branches incident to u, the computation of $\Lambda(u, s)$ can be carried out in constant time, which proves the theorem. □

Estimating bulk up-moves and bulk down-moves

It is quite challenging to compute efficiently the exact change in cost of every possible bulk move. Instead, we focus on finding moves that are guaranteed to be advantageous, and use upper bounds on their change in cost.

Recall that in a bulk up-move $m[s \to t]$, we remap all the roots of the s-duplication subtrees of $F_m(s)$ to t. We can easily iterate over all these roots since they are stored in $B[s, H_m(s)]$. The move reduces the height in s-duplications by 1, but might increase the height in other species. If the latter happens, no reduction in segmental duplications occurs and because up-moves create losses, the move will not be advantageous and we ignore it. To detect this, we check whether $\max_{g \in B[s, H_m(s)]} \Delta(g, t) > 0$. If so, the move is ignored, but otherwise the bulk remapping will not incur new duplications and we add the move to the list of candidates. We estimate the change in duplications to $\max_{g \in B[s, H_m(s)]} \Delta(g, t) - 1$ (where the -1 accounts for the reduction in s). We then estimate the number of losses created by remapping every root to t with $\sum_{g \in B[s, H_m(s)]} \Lambda(g, t)$. This is only an upper bound because of the following special case: if we remap two nodes u, v from the same gene tree, and both end up remapping their common ancestor, the losses incurred on that common ancestor will be double-counted. Nonetheless, we believe that such cases do not affect dramatically our choice of moves.

In terms of complexity, for each $s \in V(S)$ and $t \succ s$, there are $|B[s, H_m(s)]|$ gene tree nodes to enumerate, and on which to take the max of $\Delta(g, t)$ terms and the sum of $\Lambda(g, t)$ terms (which we assume are stored by the current iteration). The total time is $\sum_{s \in V(S)} |B[s, H_m(s)]|h(S) = h(S) \cdot \sum_{s \in V(S)} |B[s, H_m(s)]| = O(h(S)|V(G)|)$, which is the same time as before.

For bulk down-moves, we use a similar idea. When considering a move $m[s \to t]$ with $t \prec s$, we take all the deepest nodes in the forest $F_m(s)$ (these can be stored in a data structure analogous to B, we omit the details), and try to remap those to t. If this is not allowed for one such node, or if such a remapping would increase the number of segmentnal duplications, again taking the max as before, the move is considered inadmissible.