Efficient Non-Binary Gene Tree Resolution with Weighted Reconciliation Cost

Manuel Lafond[1], Emmanuel Noutahi[2], and Nadia El-Mabrouk[3]

1 **DIRO, Université de Montréal, H3C 3J7, Canada**
`lafonman@iro.umontreal.ca`
2 **DIRO, Université de Montréal, H3C 3J7, Canada**
`noutahie@iro.umontreal.ca`
3 **DIRO, Université de Montréal, H3C 3J7, Canada**
`mabrouk@iro.umontreal.ca`

Abstract

Polytomies in gene trees are multifurcated nodes corresponding to unresolved parts of the tree, usually due to insufficient differentiation between sequences. Resolving a multifurcated tree has been considered by many authors, the objective function often being the number of duplications and losses reflected by the reconciliation of the resolved gene tree with a given species tree. Here, we present PolytomySolver, an algorithm accounting for a more general model allowing for costs that can vary depending on the operation, but also on the considered genome. The time complexity of PolytomySolver is linear for the unit cost and is quadratic for the general cost, which outperforms the best known solutions so far by a linear factor. We show, on simulated trees, that the gain in theoretical complexity has a real practical impact on running times.

1998 ACM Subject Classification Biology and genetics

Keywords and phrases gene tree, polytomy, reconciliation, resolution, weighted cost, phylogeny

Digital Object Identifier 10.4230/LIPIcs.CPM.2016.14

1 Introduction

Reconstructing gene trees is a fundamental task in bioinformatics and a prerequisite for most biological studies on gene function. Consequently, a plethora of phylogenetic methods have been developed, most of them integrating measures of statistical support (e.g. by bootstrapping or jackknifing), reflecting the confidence we have on the prediction. Some of them, such as bayesian methods [9, 11] lead to non-binary trees. Moreover, weakly supported branches are often contracted and also lead to non-binary trees. Thus, although unresolved nodes in a tree may reflect a true (or *hard* [12]) simultaneous speciation or duplication event leading to more than two gene copies, they are usually artifacts (called *soft*), due to methodological reasons or to a lack of resolution between sequences.

Information for the full resolution of a gene tree may rely on the weakly exploited link between gene and species evolution. The question of resolving a non-binary gene tree by minimizing the number of duplications and losses resulting from the reconciliation of the gene tree with the species tree has first been considered in NOTUNG [2] and later by Chang and Eulenstein [1]. In 2012 [8], we developed the first linear-time algorithm for resolving a polytomy (a single unresolved node), leading to a quadratic-time algorithm for a whole tree. Recently, algorithmic results extending linearity to a whole gene tree have been obtained by Zheng and Zhang [15]. These linearity results are however restricted to the case of a unit cost for duplications and losses. On the other hand, an algorithm allowing different costs for

27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016).
Editors: Roberto Grossi and Moshe Lewenstein; Article No. 14; pp. 14:1–14:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Time-complexity results for reporting a single optimal resolution of a whole gene tree G of size $|G|$ with a species tree S of size $|S|$, where Δ is the largest degree of a node in G, δ is the cost of a duplication and λ the cost of a loss. The last column refers to the case in which each species s has its own duplication cost δ_s and loss cost λ_s.

	$\delta = \lambda = 1$	$(\delta, \lambda) \in \mathbb{R}_{>0} \times \mathbb{R}_{>0}$	$\{(\delta_s, \lambda_s)\}_{s \in V(S)}$
NOTUNG[6]	$O(\|S\|\|G\|\Delta^2)$	$O(\|S\|\|G\|\Delta^2)$	
Lafond[8]	$O(\|S\|\|G\|)$	$O(\|S\|\|G\|\Delta)$	
Zheng & Zhang[15]	$O(\|G\|)$	$O(\|G\|\Delta^2)$	
PolytomySolver	$O(\|G\|)$	$O(\|G\|\Delta)$	$O(\|G\|\|S\|\Delta)$

duplications and losses has been considered in NOTUNG [6], and further improved by Zheng and Zhang [15], using a compressed species tree idea.

In this paper, we present a new algorithm called PolytomySolver, which handles unit costs in linear time and improves the best complexity to date for more general duplication and loss cost model by a linear factor (complexity results are given in Table 1). Additionally, PolytomySolver is the first algorithm enabling to account for various evolutionary rates across the branches of a species tree, as it allows assigning each taxa its specific duplication and loss cost. This functionality may be used to reduce the effect of missing data by assigning a lower loss cost to species that are more likely to be concerned by such loss of information. It is also of practical use when biological evidence supports some particularly low or high gene duplication or loss rates in some species of interest [10]. In particular, fractionation following whole genome duplication (WGD) results in an excess of gene losses. In Section 6, we give an example showing that assigning appropriate costs to post-WGD genomes is important for an accurate inference.

The paper is subdivided as follows. First, in Section 3, we show how the linear-time algorithm developed previously by our group [8] for resolving a polytomy with unit duplication and loss cost can be extended to arbitrary costs, depending on the operation and on the genome affected by the operation. This extension is however not linear anymore but rather leads to a cubic-time algorithm. We then, in Section 4, show how using the ideas introduced by Zheng and Zhang [15] allows to reduce this time complexity to quadratic, which is the best obtained to date for the same problem. We also show how unit costs can be handled in linear time, and how PolytomySolver can be used to output all optimal resolutions, which is an advantage compared to Zheng and Zhang's algorithms. In Section 5, comparing our new algorithm with NOTUNG and Zheng and Zhang's algorithm, we show that the obtained gain in theoretical complexity actually leads to a significant gain in running times. For space reason, all proofs are given in Appendix, which is available online at `http://www-ens.iro.umontreal.ca/~lafonman/en/publications.php`.

2 Preliminary

All trees are considered to be rooted. Given a set X, a *tree* T *for* X has its leafset $\mathcal{L}(T)$ in bijection with X. Denote by $V(T)$ its set of nodes, $r(T)$ its root, and write $|T| = |V(T)|$. Given two nodes x and y of T, x is a *descendant* of y, and y is an *ancestor* of x, if y is on the (inclusive) path between x and $r(T)$. The *degree* $deg(x)$ of a node x is the number of edges incident to x. The maximum degree of T is $\Delta(T) = \max_{v \in V(T)} deg(v)$ (or just Δ when T is clear from the context). Given a set L of leaves, the *lowest common ancestor* of L in

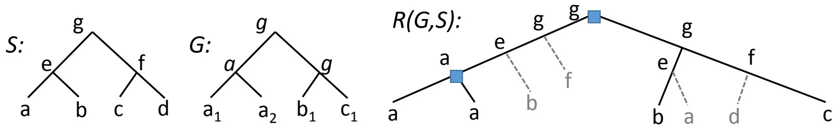


Figure 1 S is a species tree over $\Sigma = \{a, b, c, d\}$; G is a gene tree on the gene family Γ with two copies in genome a, one in genome b and one in genome c; $R(G, S)$ is a reconciliation of G with S with two duplications and four losses. Each node x of G and $R(G, S)$ is labeled by $s(x)$.

T, denoted $lca_T(L)$, is the common ancestor of L in T that is farthest from the root. A *polytomy* (or star tree) over a set L is a tree with a single internal node, which is of degree $|L|$, adjacent to each leaf of L. Finally, if x is a node of T, denote by T_x the subtree of T rooted at x, and by $T(x)$ the polytomy obtained by keeping only x and its children in T_x.

2.1 Gene Tree, Species Tree and Reconciliation

A species tree S for a set $\Sigma = \{\sigma_1, \cdots, \sigma_t\}$ of species represents an ordered set of speciation events that have led to Σ. Inside the species' genomes, genes undergo speciations when the species to which they belong do, but also duplications and losses (other events such as transfers can happen, but we ignore them here). A *gene family* is a set Γ of genes where each gene x belongs to a given species $s(x)$ of Σ. The evolutionary history of Γ can be represented as a *gene tree* G where $\mathcal{L}(G)$ is in bijection with Γ, and each internal node refers to an ancestral gene at the moment of an event (either speciation or duplication) belonging to the species $s(x) = lca_S(\{s(y) : y \in \mathcal{L}(G_x)\})$. We denote $\mathcal{S}(G) = \{s(y) : y \in \mathcal{L}(G)\}$ the set of species *represented by* G.

In this paper, we make no distinction between paralogous gene copies. In other words, a gene x is simply identified by the genome $s(x)$ it belongs to. A gene tree is therefore a tree where each leaf is labeled by an element of Σ, with possibly repeated leaf labels (Figure 1).

A *reconciliation* is an extension of the gene tree, obtained by adding lost branches, reflecting a history of duplications and losses in agreement with the species tree. Formally, an *extension* of G is a tree obtained from G by a sequence of graftings, where a *grafting* consists in subdividing an edge uv of G, thereby creating a new node w between u and v, then adding a leaf x with parent w. The new leaf x is mapped to a species $s(x)$ which is a node of S (internal or leaf). A formal definition follows (see Figure 1 for an example).

▶ **Definition 1** (Reconciled gene tree). Let G be a binary gene tree and S be a binary species tree. A *reconciliation* $R(G, S)$ of G with S is an extension of G verifying: for each internal node x of $R(G, S)$ with two children x_l and x_r, either $s(x_l) = s(x_r) = s(x)$, or $s(x_l)$ and $s(x_r)$ are the two children of $s(x)$. The node x is a duplication in $s(x)$ in the former case, and a speciation node in $s(x)$ in the latter case. A grafted leaf x corresponds to a loss in $s(x)$.

Define δ_s as the duplication cost and λ_s as the loss cost assigned to a given species s. Then, the *reconciliation cost* of $R(G, S)$ is the sum of costs of the induced duplications and losses.

2.2 Problem statement

We consider a binary species tree S and a non-binary gene tree G. The goal is to find a *binary refinement* of G, as defined below.

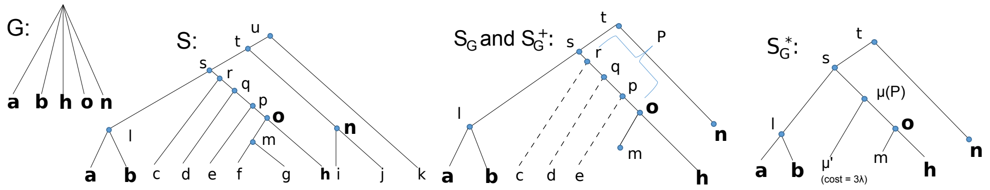


Figure 2 From left to right: a gene tree G; a species tree S; the species tree S_G linked to G is the tree illustrated by plain lines, and the augmented species tree S_G^+ linked to G is illustrated by plain and dotted lines; the compressed tree S_G^* linked to G as defined in Section 4. The leaf μ' of S_G^* has a special loss cost $\lambda_{\mu'} = 3$, as it results from the contraction of a path of length 3.

▶ **Definition 2** (binary refinement). A *binary refinement* $B = B(G)$ of G is a binary tree such that $V(G) \subseteq V(B)$ and for every $x \in V(G)$, $\mathcal{L}(G_x) = \mathcal{L}(B_x)$.

The objective function taken for choosing among all possible binary refinements is the reconciliation cost.

▶ **Definition 3** (Resolution). A *resolution* of G with respect to S is a reconciliation $R(B, S)$ between a binary refinement B of G and S. The set of all possible resolutions of a tree G is denoted $\mathcal{R}(G)$.

We are now ready to state our optimization problem.

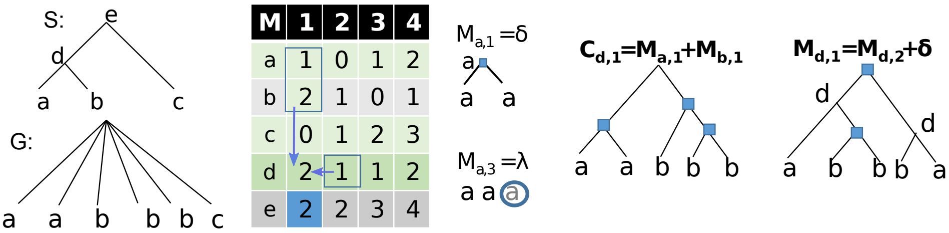
Minimum Resolution Problem
Input: A binary species tree S and a non-binary gene tree G.
Output: A *Minimum Resolution* of G with respect to S (or simply *Minimum Resolution of* G), e.g. a resolution of G of minimum reconciliation cost with respect to S.

It has been previously shown [1] that each polytomy of G can be considered independently. In particular, a minimum resolution of G can be obtained by a depth-first procedure that solves each polytomy $G(x)$ iteratively, for each internal node x of G. Thus, in the following, we focus on a single polytomy $G = G(x)$.

Some parts of the species tree can be ignored in the process of refining G. Define the *species tree linked to* G, denoted by S_G, as the tree obtained from the subtree of S rooted at the lowest common ancestor of $\mathcal{S}(G)$, by removing all nodes that have no descendant in $\mathcal{S}(G)$ (Figure 2). The algorithms with the best known complexity results (Table 1) are obtained by using a compressed version S_G^* of this tree, which is defined in Section 4. We first begin, in Section 3, by describing the refinement strategy by using an *augmented species tree linked to* G, denoted S_G^+, obtained from S_G by adding to every node of degree two its missing child in S. It is known (c.f. [8, 15]) that resolving G with either S or S_G^+ leads to the same reconciliation cost. Intuitively, S_G^+ contains every node of S that may appear in a resolution of G, whether as a loss, a duplication or a speciation.

3 A dynamic programming approach

We present a dynamic programming approach for the Minimum Resolution Problem for a single polytomy G. It is a generalization of that presented in [8]. While the previous algorithm was developed for a unit cost of duplications and losses, the one we present here



M	1	2	3	4
a	1	0	1	2
b	2	1	0	1
c	0	1	2	3
d	2	1	1	2
e	2	2	3	4

Figure 3 A polytomy G and a species tree S. The corresponding table M is obtained for $\delta_s = \lambda_s = 1$ for all species. Squares on trees illustrate duplications. To the right of table M, the forests corresponding to an $(a, 1)$ and $(a, 3)$-resolution are given, where the circled a illustrates a singleton loss. We illustrate the $(d, 1)$-resolution, rooted at a speciation node, corresponding to $C_{d,1} = 3$ (obtained from the vertical arrow in table M), and an optimal $(d, 1)$-resolution, obtained from a $(d, 2)$-resolution (horizontal arrow in M).

holds for a more general reconciliation cost, where each $s \in \Sigma$ has its own duplication cost δ_s and loss cost λ_s. In this section, we assume that $S = S_G^+$.

The recursion is made on the subtrees of S. Define the multiplicity $m(s)$ of $s \in V(S)$ in G as the number of times it appears in G, i.e. $m(s) = |\{x \in \mathcal{L}(G) : s(x) = s\}|$. An (s, k)*-resolution* of G is a forest of k reconciled gene trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ such that, for each $1 \leq i \leq k$, $s(r(T_i)) = s$, and each leaf x of G with $s(x)$ being a descendant of s is present as a leaf of some tree of $\mathcal{T}$ (see Figure 3 for an example). All leaves of trees in $\mathcal{T}$ that are not in $\mathcal{L}(G)$ represent losses. Also, some trees of $\mathcal{T}$ may be restricted to a single node which is either a child x of $r(G)$ with $s(x) = s$, or a singleton loss in s. The cost of $\mathcal{T}$, denoted $c(\mathcal{T})$, is the sum of reconciliation costs of all T_is. Notice that since $S = S_G^+$, a resolution of G is an $(r(S), 1)$*-resolution.*

Denote by $M_{s,k}$ the minimum cost of an (s, k)-resolution for a given node s of S and a given integer $k \geq 1$ (and $M_{s,k} = \infty$ for $k < 1$). The final cost of a minimum resolution of G is given by $M_{r(S),1}$. The table M is computed, line by line, for all nodes of S, in a bottom-up traversal. For now, k is unlimited, but we show in the complexity section that there is no need to consider more than $|G| - 1$ columns.

The following lemma gives the base case for the leaves of S. It follows from the fact that, if k is larger than the number of available leaves, then additional leaves have to be added (called *singleton losses*); otherwise leaves have to be joined under duplication nodes. As an illustration, in Figure 3, this lemma is used to compute the three first lines of M.

▶ **Lemma 4** (Base case). *For a leaf node s of S, if $k > m(s)$ then $M_{s,k} = \lambda_s \cdot (k - m(s))$; otherwise $M_{s,k} = \delta_s \cdot (m(s) - k)$.*

The rest of this section focuses on the computation of a line M_s of M for an internal node s of S, from the lines M_{s_l} and M_{s_r}, where s_l and s_r are the two children of s in S. We require an intermediate cost table $C_{s,k}$, defined for internal nodes of S, accounting only for speciation events. That is, $C_{s,k}$ represents the minimum cost of an (s, k)-resolution in which every tree is rooted at a speciation node with two children (these two children may both be losses), or consists of a singleton node that is a child of $r(G)$ already mapped to s. For $k > m(s)$, such an (s, k)-resolution of cost $C_{s,k}$ can only be obtained from an $(s_l, k - m(s))$-resolution and an $(s_r, k - m(s))$-resolution by creating $k - m(s)$ speciation nodes, each joining a pair of (s_l, s_r) trees, then adding the $m(s)$ singleton trees mapped to s. No other scenarios are possible, since (s, k)-resolutions are reconciled trees, and each non-singleton root is a speciation in s

that must have genes mapped to s_l and s_r as children. See for example the $(d, 1)$-resolution corresponding to $C_{d,1}$ in Figure 3. Note that if instead $k \leq m(s)$, such an (s, k)-resolution cannot exist, since $m(s)$ trees are required for the children of $r(G)$ mapped to s, plus at least another tree containing the genes in a descendant of s. Thus we define:

$$C_{s,k} = M_{s_l,k-m(s)} + M_{s_r,k-m(s)} \text{ if } k > m(s) \text{ and } C_{s,k} = +\infty \text{ otherwise} \qquad \textbf{(1)}$$

It is readily seen that $M_{s,k} \leq C_{s,k}$. A recurrence for computing $M_{s,k}$ follows.

▶ **Lemma 5.** *For an internal node s of S, $M_{s,k} = \min(M_{s,k-1} + \lambda_s, M_{s,k+1} + \delta_s, C_{s,k})$.*

This recurrence cannot be used as such to compute C and M, as it induces both a left and right dependency. That is, $M_{s,k}$ depends on $M_{s,k+1}$ and vice-versa, leading to a chicken-and-egg problem as to which value should be computed first. In the case of a unit cost $\delta_s = \lambda_s = 1$ for all s, we have shown in [8] that this dependency can be avoided by considering a strong property on lines of M. Indeed, each line M_s is characterized by two values k_1 and k_2 such that, for any $k_1 \leq k \leq k_2$, $M_{s,k}$ is minimum, for any $k \leq k_1$, $M_{s,k-1} = M_{s,k} + 1$, and for any $k \geq k_2$, $M_{s,k+1} = M_{s,k} + 1$. In other words, M_s has a slope of -1 until k_1, a slope of 0 until k_2, then a slope of 1. In particular, M_s can be treated as a convex function fully determined by k_1, k_2 and its minimum value γ. We then say M_s has a *minimum plateau* between k_1 and k_2. For example, line M_d in Figure 3 is fully determined by $k_1 = 2$ and $k_2 = 3$.

Here, we extend these results by first showing, in Lemma 7, that both C and M are still convex, albeit having less predictable changes in the slopes. Nevertheless, this allows to first compute the bounds k_1 and k_2 of the functions' minimum plateau, and then extend to the left and to the right from this plateau.

We first recall the formal definition of a discrete convex function, then state the convexity result for C and M and finally give the recurrences of the dynamic programming algorithm in Theorem 8.

▶ **Definition 6** (Convex function)**.** A discrete function f is convex if and only if, for any integer $n > 1$, the two following statements, which are equivalent, are true.

- $f(n+1) + f(n-1) - 2f(n) \geq 0$;
- for any integers $\epsilon_1, \epsilon_2 > 0$ and any integer $n > \epsilon_1$, $f(n - \epsilon_1) + f(n + \epsilon_2) - 2f(n) \geq 0$.

▶ **Lemma 7.** *Both M_s and C_s are convex.*

▶ **Theorem 8** (Recurrence 2)**.** *Let k_1 and k_2 be the smallest and largest values, respectively, such that $C_{s,k_1} = C_{s,k_2} = \min_k C_{s,k}$. Then,*

$$M_{s,k} = \begin{cases} C_{s,k} & \textit{if } k_1 \leq k \leq k_2 \\ \min(C_{s,k}, M_{s,k+1} + \delta_s) & \textit{if } k < k_1 \\ \min(C_{s,k}, M_{s,k-1} + \lambda_s) & \textit{if } k > k_2 \end{cases}$$

Theorem 8 provides the way for computing a row M_s for an internal node s of S: for each k, compute $C_{s,k}$ using recurrence **(1)** and keep the two columns k_1 and k_2 setting the bounds of the convex function's plateau. Extend to the left of k_1 using $M_{s,k} = \min(C_{s,k}, M_{s,k+1} + \delta_s)$, and to the right of k_2 using $M_{s,k} = \min(C_{s,k}, M_{s,k-1} + \lambda_s)$. These recurrences, with the base case for S leaves given in Lemma 4, describe the dynamic programming algorithm, that we call PolytomySolver, for computing the cost $M_{r(S),1}$ of a minimum resolution of the polytomy G with respect to S. We refer the reader to [8] for the reconstruction of a solution from M in linear time, which is accomplished using a standard backtracking procedure.

Complexity

The following lemma states that there is no reason to explore more gene copies of a given species than the size of the polytomy, in other words, the size of a line of M can be bounded by $|G|$. This fact may seem obvious to the accustomed, but in [6] it was equally “obvious" that only $m^* = \max_{s \in V(S)} m(s)$ columns needed to be considered, which turns out to be wrong [1]. In fact, this Lemma requires a surprising amount of care in the details (see Appendix).

▶ **Lemma 9.** *Only the values of M and C for columns k between 1 and $|G| - 1$ need to be computed.*

It follows from Lemma 4, Theorem 8 and Lemma 9 that each row of C and M can be computed in time $O(|G|)$, and the whole table in time $O(|S||G|)$.

Now suppose that H is a general tree with p polytomies, where Δ is the largest degree of a polytomy. According to the depth-first procedure described at the end of Section 2, G can be resolved in time $O(p|S|\Delta)$, which is less than $O(|H||S|\Delta)$. In the next section, we improve this to $O(|H|\Delta)$ in the case of distinct costs δ and λ that are shared across all species, and $O(|H|)$ in the case of equal costs $\delta = \lambda$.

4 A faster algorithm using species tree compression

Assume that all species have the same duplication cost δ and the same loss cost λ. We call it *unit cost* if $\delta = \lambda$, and *general cost* otherwise. Again we assume that G is a polytomy.

In the previous section, results have been obtained using the augmented linked species tree S_G^+. As observed by Zheng and Zhang [15], S_G^+ contains many “useless" nodes that do not provide any meaningful information with regards to the resolution of G. This idea allowed them to optimize their refinement algorithm for the unit cost, leading to a linear-time algorithm. However, their algorithm does not apply to the general cost. For such a cost, their optimisation idea was rather applied to the NOTUNG's algorithm, which is less efficient. Here, we use a similar idea to optimize PolytomySolver. More precisely, we show how a compressed version of the linked species tree S_G can be used to reduce the complexity for refining a general tree G to $O(|G|\Delta)$ for the general cost, and to $O(|G|)$ for the unit cost.

We first need some definitions. Let T be a tree. Call P a *path in* T if P is a sequence of non-root adjacent vertices of degree two in T. *Contracting* P in T consists in replacing P by a single node $\mu = \mu(P)$. Now, let U be the set of non-root vertices of degree two of S_G that are not in $\mathcal{S}(G)$. We call U the set of “useless nodes" of S_G. Notice that $S_G[U]$, the graph obtained from S_G by keeping only nodes of U and edges with both endpoints in U, corresponds to a set of disjoint paths in S_G. The *compressed tree* S_G^* is the tree obtained from S_G by contracting every path P of $S_G[U]$ to $\mu = \mu(P)$, then adding a leaf child μ' to every such μ (see Figure 2 for an example). Moreover, we set a special loss cost $\lambda_{\mu'} = \lambda|P|$ to μ' (and duplication cost δ as every other node). This special loss cost ensures that a loss in μ' is counted as a loss in every node in P. Notice that some internal nodes of S_G that are included in $\mathcal{S}(G)$ may still have only one child. Thus S_G^* is finally obtained by adding to each remaining node having only one child a new leaf child (duplication of cost δ and loss cost λ). The following Theorem ensures that S_G^* does not change the solution space.

[1] The complexity reported in Table 1 is not the one reported by NOTUNG, as dependency is not given on Δ but instead on m^*. However, it can be shown that considering m^* columns is not enough on some examples.

▶ **Theorem 10.** *Let T be a binary refinement of G. Then the reconciliation cost of T is the same whether we reconcile it with S_G^+ or S_G^* and their corresponding duplication/loss costs.*

Thus, using S_G^+ or S_G^* leads to the same minimum resolution for G. We show that using S_G^* leads to reduction in time complexity of the algorithm.

▶ **Theorem 11.** *Given a gene tree H,* PolytomySolver *can run in time $O(\Delta|H|)$.*

4.1 The case of a unit cost

In [8], we showed how, in the case of a unit cost $\delta = \lambda$, each line M_s of M can be computed in constant time. However, in order to take advantage of the compressed species tree $S = S_G^*$, we need to account for special leaves μ' with loss cost $\lambda_{\mu'} > 1$, since they make the cost not unitary anymore. The following theorem allows us to extend the result to this specific case. It leads to the computation of M in time $O(|S_G^*|) = O(|G|)$ for a polytomy G. The complexity for a gene tree H is thus reduced to $O(|H|)$, which results in a reduction of the previous complexity by a factor of Δ.

▶ **Theorem 12.** *Suppose $S = S_G^*$. Then for $s \in V(S)$,*

1. *if s is a leaf with loss cost $\lambda = 1$, then $M_{s,k} = |k - m(s)|$;*
2. *if s is a leaf with loss cost $\lambda_s > 1$, then $M_{s,k} = k \cdot \lambda_s$;*
3. *if s is an internal node, there exist 3 integers k_1, k_2 and γ_s such that*

$$M_{s,k} = \begin{cases} \gamma_s & \text{if } k_1 \leq k \leq k_2 \\ \gamma_s + k_1 - k & \text{if } k < k_1 \\ \gamma_s + k - k_2 & \text{if } k > k_2 \end{cases}$$

Moreover, k_1, k_2 and γ_s can be computed in constant time.

4.2 Constructing all minimum resolutions

After computing table M, it remains to compute $(r(S), 1)$-resolutions, i.e. all resolutions of minimum cost. Without any increase in the theoretical time complexity of the algorithm, a simple pass through table M leads to one minimum resolution (see [8] for the details). Here we rather show how to recover all minimum resolution.

Denote by $\mathcal{P}_{s,k}$ the set of all minimum (s, k)-resolutions of a polytomy G. By setting $s = r(S)$ and $k = 1$, we exhibit the following recursive algorithm that finds $\mathcal{P}_{r(S),1}$. To do so, we define three intermediate solution sets $\mathcal{P}_{s,k}^{dup}, \mathcal{P}_{s,k}^{loss}$ and $\mathcal{P}_{s,k}^{spec}$, which respectively correspond to (s, k)-resolutions containing a duplication root, a singleton loss and only speciation roots (it turns out that these three cases are disjoint).

We show in the Appendix that this algorithm eventually terminates, and does find every solution. The essential reason that this algorithm finishes is because of the convexity of M_s, which allows avoiding circular dependencies between say $\mathcal{P}_{s,k}$ and $\mathcal{P}_{s',k'}$.

It can be shown that this algorithm takes time $O(|S| \cdot |\mathcal{P}_{r(S),1}|)$, which may be exponential. Methods for outputting solutions iteratively, each in polynomial time, seem possible, but are not immediately obvious. Notice that Zheng and Zhang's algorithms [15] can only output a subset of $\mathcal{P}_{r(S),1}$. As for NOTUNG, it takes time $O(|S|\Delta \cdot (|\mathcal{P}_{r(S),1}| + \Delta))$ to construct every optimal solution [2].

```
procedure COMPUTE $\mathcal{P}_{s,k}$
    if $s$ is a leaf and $m(s) = k$ then
        return $k$ singleton trees mapped to $s$
    Let $\mathcal{P}^{dup}_{s,k} = \emptyset, \mathcal{P}^{loss}_{s,k} = \emptyset, \mathcal{P}^{spec}_{s,k} = \emptyset$
    if $M_{s,k} = M_{s,k+1} + \delta_s$ then
        Compute $\mathcal{P}_{s,k+1}$
        for every forest $\mathcal{T}$ in $\mathcal{P}_{s,k+1}$, and for every pair of distinct trees $T_1, T_2 \in \mathcal{T}$ do
            Add to $\mathcal{P}^{dup}_{s,k}$ the $(s,k)$-resolution obtained by joining $r(T_1)$ and $r(T_2)$
    if $M_{s,k} = M_{s,k-1} + \lambda_s$ then
        Compute $\mathcal{P}_{s,k-1}$
        for every forest $\mathcal{T}$ in $\mathcal{P}_{s,k-1}$ do
            Add to $\mathcal{P}^{loss}_{s,k}$ the $(s,k)$-resolution obtained adding a singleton loss in $s$ in $\mathcal{T}$
    if $s$ is an internal node with children $s_1, s_2$ and $M_{s,k} = M_{s_1,k-m(s)} + M_{s_2,k-m(s)}$ then
        Compute $\mathcal{P}_{s_1,k-m(s)}$ and $\mathcal{P}_{s_2,k-m(s)}$
        for each pair $(\mathcal{T}_1, \mathcal{T}_2)$ in $\mathcal{P}_{s_1,k-m(s)} \times \mathcal{P}_{s_2,k-m(s)}$, and for every bijection $f : \mathcal{T}_1 \longrightarrow \mathcal{T}_2$ do
            Add to $\mathcal{P}^{spec}_{s,k}$ the $(s,k)$-resolution $\mathcal{T}$ obtained by joining $r(T_1)$ with $r(f(T_1))$ for every $T_1 \in \mathcal{T}_1$, then adding the $m(s)$ children of $G$ mapped to $s$ as singleton trees
    Let $\mathcal{P}_{s,k} = \mathcal{P}^{dup}_{s,k} \cup \mathcal{P}^{loss}_{s,k} \cup \mathcal{P}^{spec}_{s,k}$, and return $\mathcal{P}_{s,k}$
end procedure
```

5 Results on simulated data

We compare the running time of our algorithm to Zheng and Zhang's algorithms [15] and NOTUNG, on simulated datasets for both cases of unit and general costs. We implemented PolytomySolver and Zheng and Zhang's algorithms in python and used the latest stable version (v2.6)[2] of NOTUNG. Our implementations are available at `https://github.com/UdeM-LBIT/profileNJ`. Run times are reported for single outputs of the algorithms.

We first simulated species trees with n leaves using a birth-death process. For each species tree, gene trees of fixed size $(1.5 \times n)$ and branch support picked from a standard uniform distribution, were simulated using a simple Yule process [13]. In order to mimic a gene family history with a high number of events (duplications and losses), we labeled each leaf of the gene tree with a uniformly chosen species from the set of leaves of the species tree. Non-Binary gene trees were then obtained by contracting edges of the gene trees with support lower than a fixed threshold r (0.2, 0.4, 0.6 and 0.8).

For each species tree and each algorithm, we measured the average running time on 40 non-binary trees (10 simulated gene trees for each contraction rate). All software were run on the same computer and with the same costs for duplications and losses.

We first considered the unit cost ($\lambda = \delta = 1$), for which both PolytomySolver and Zheng and Zhang's algorithm (LZZ) are linear. Figure 4a shows the results for values of n ranging from 500 to 10000, and Figure 4b shows results for n between 10000 and 100000. As expected, the two linear algorithms exhibit very similar run time in all cases, and are significantly

[2] Notice that an improved version of NOTUNG v2.8 became available after these tests were performed.

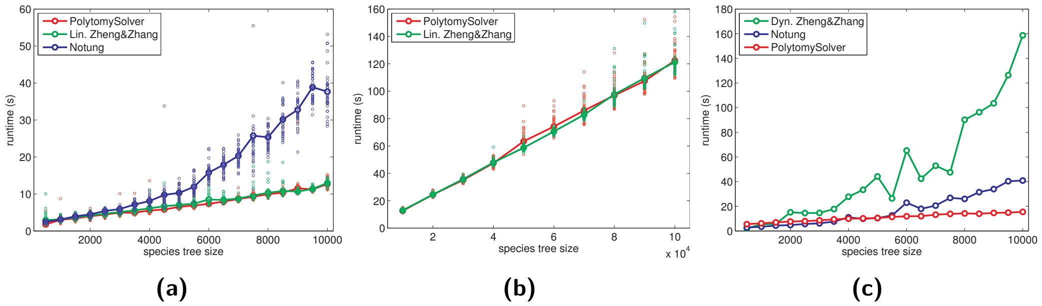


Figure 4 Running times comparisons between all algorithms for species trees of increasing size n and gene trees of size $1.5 \times n$. a Running times of PolytomySolver, LZZ (linear Zheng and Zhang's algorithm) and NOTUNG, using unit cost, for species trees of increasing size ranging from 500 to 10000. b Running times of PolytomySolver and LZZ for unit cost on larger species trees (n in the range of 10000 to 100000). c Running times of PolytomySolver, DZZ (Dynamic Zheng and Zhang's algorithm) and NOTUNG using $\delta = 3$ and $\lambda = 2$.

faster than NOTUNG, which could not be included in Figure 4b. Indeed, on those trees, NOTUNG took a considerable amount of time, and in some cases we could not get a result after many hours.

We then considered a non-unit cost, using $\delta = 3$ and $\lambda = 2$. Recall that PolytomySolver is quadratic in this case. As for the algorithm proposed by Zheng and Zhang for these costs, that we refer to by DZZ (for Dynamic Zheng and Zhang's algorithm), it is (essentially) cubic (see Table 1). Figure 4c gives the results for species trees of size ranging between 500 and 10000. As expected, PolytomySolver is faster than DZZ and NOTUNG. Surprisingly, NOTUNG turns out to be faster than DZZ, which rather expected to improve over NOTUNG as it uses the species tree compression idea. This could be due to the fact that NOTUNG is a well optimized program. Moreover, the error in NOTUNG of using m^* instead of Δ (see footnote in this Section 3), may accelerate the process, as m^* is usually much smaller than Δ.

6 A practical use of PolytomySolver

As handling species specific costs is one of the major contribution of this paper, we conclude our presentation by providing a biological example for which taking advantage of this flexibility of PolytomySolver leads to better accuracy.

We first downloaded the orthogroup of the yeast gene REG1, a regulatory subunit of type 1 protein phosphatase Glc7p, involved in negative regulation of glucose-repressible genes, from the Fungal Orthogroups Repository (http://www.broadinstitute.org/regev/orthogroups/). We then reconstructed the gene tree with PolytomySolver, using the same species tree as [14] and a unit cost for both λ and δ. Two equally parsimonious solutions with a reconciliation cost of 2 were obtained (Figures 5B, 5C).

It has been shown that the yeast *Saccharomyces cerevisiae* arose from an ancient whole-genome duplication (WGD) [4, 5, 7]. This WGD was immediately followed by a massive gene loss period, during which most of the duplicated gene copies were lost [7]. There is also evidence of lineage-specific loss of paralogous genes. In particular, *C. glabrata* and *S. castellii* appear to have lost several hundred paralogs [3, 5]. This is reflected in their total gene count, which are the lowest among the post-WGD genomes [14].

Whereas the solution shown in Figure 5C is in agreement with this WGD event, the alternative gene family history in Figure 5B places the duplication much lower in the tree, with

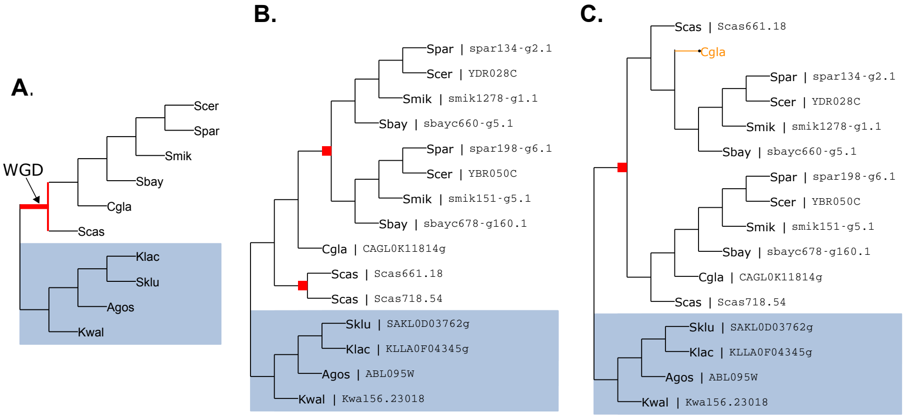


Figure 5 **A.** Phylogeny of ten Hemiascomycota fungi, including *S. cerevisiae (Scer), S. paradoxus (Spar), S. mikatae (Smik), S. bayanus (Sbay), C. glabrata (Cgla), S. castellii (Scas), K. waltii (Kwal), K. lactis (Klac), S. kluyveri (Sklu) and A. gossypii (Agos).* The whole-genome duplication (WGD) event in yeast is indicated. The species that did not went through the WGD are shadowed in light blue. **B.** and **C.** Two minimally resolved gene trees of the phosphatase Glc7p gene family. Duplication nodes are depicted by a red square and lost genes are shown in orange.

and additional duplication in *S. castellii* instead. By assigning to *C. glabrata* and *S. castellii* a loss cost lower than for all other species, the only solution returned by PolytomySolver is the one shown in Figure 5C. Using appropriate species dependant costs might therefore allow to filter the solution space with additional relevant information.

7 Conclusion

PolytomySolver is the most efficient algorithm to date for refining an unresolved gene tree. In contrast to previous methods, this algorithm is flexible enough to handle general reconciliation costs, allowing for instance to account for different costs over the branches of a species tree. Moreover, all topologies of optimal trees can be output by PolytomySolver. Notice that here we made no distinction between paralogous genes, which are simply referred to by their genome of origin. If we rather consider the specificity of each gene copy then, for a given topology obtained by PolytomySolver, an appropriate method shall be considered to distribute gene copies on leaves. We are presently investigating the possibility of introducing a Neighbor-Joining principle in the resolution process.

The gain in running time attained with PolytomySolver allows to perform exhaustive corrections of all trees contained in a large gene tree dataset such as Ensembl. Moreover, compared with NOTUNG, running time is independent upon the largest degree of a node, which makes the algorithm efficient enough to resolve highly unresolved trees. The next step will be to perform such a large scale gene tree dataset correction.

References

1 W.C. Chang and O. Eulenstein. Reconciling gene trees with apparent polytomies. In D.Z. Chen and D. T. Lee, editors, *Proceedings of the 12th Conference on Computing and Combinatorics (COCOON)*, volume 4112 of *Lecture Notes in Computer Science*, pages 235–244, 2006.

2 K. Chen, D. Durand, and M. Farach-Colton. Notung: Dating gene duplications using gene family trees. *Journal of Computational Biology*, 7:429–447, 2000.

3 Paul F Cliften, Robert S Fulton, Richard K Wilson, and Mark Johnston. After the duplication: gene loss and adaptation in saccharomyces genomes. *Genetics*, 172(2):863–872, 2006.

4 Fred S Dietrich, Sylvia Voegeli, Sophie Brachat, Anita Lerch, Krista Gates, Sabine Steiner, Christine Mohr, Rainer Pöhlmann, Philippe Luedi, Sangdun Choi, et al. The ashbya gossypii genome as a tool for mapping the ancient saccharomyces cerevisiae genome. *Science*, 304(5668):304–307, 2004.

5 Bernard Dujon, David Sherman, Gilles Fischer, Pascal Durrens, Serge Casaregola, Ingrid Lafontaine, Jacky De Montigny, Christian Marck, Cécile Neuvéglise, Emmanuel Talla, et al. Genome evolution in yeasts. *Nature*, 430(6995):35–44, 2004.

6 D. Durand, B.V. Haldórsson, and B. Vernot. A hybrid micro-macroevolutionary approach to gene tree reconstruction. *Journal of Computational Biology*, 13:320–335, 2006.

7 Manolis Kellis, Bruce W Birren, and Eric S Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast saccharomyces cerevisiae. *Nature*, 428(6983):617–624, 2004.

8 M. Lafond, K.M. Swenson, and N. El-Mabrouk. An optimal reconciliation algorithm for gene trees with polytomies. In *LNCS*, volume 7534 of *WABI*, pages 106-122, 2012.

9 Nicolas Lartillot and Hervé Philippe. A bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. *Molecular Biology and Evolution*, 21(6):1095–1109, Jun 2004. doi:10.1093/molbev/msh112.

10 Michael Lynch and John S Conery. The evolutionary demography of duplicate genes. *Journal of structural and functional genomics*, 3(1-4):35–44, 2003.

11 F. Ronquist and J.P. Huelsenbeck. MrBayes3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19:1572- 1574, 2003.

12 J.B. Slowinski. Molecular polytomies. *Molecular Phylogenetics and Evolution*, 19(1):114-120, 2001.

13 Mike Steel and Andy McKenzie. Properties of phylogenetic trees generated by yule-type speciation models. *Mathematical biosciences*, 170(1):91–112, 2001.

14 Ilan Wapinski, Avi Pfeffer, Nir Friedman, and Aviv Regev. Natural history and evolutionary principles of gene duplication in fungi. *Nature*, 449(7158):54–61, 2007.

15 Y. Zheng and L. Zhang. Reconciliation with non-binary gene trees revisited. In *Lecture Notes in Computer Science*, volume 8394, pages 418-432, 2014. Proceedings of RECOMB.