From gene trees to species trees through a supertree approach

Celine Scornavacca[1,2,*], Vincent Berry[2], and Vincent Ranwez[1]

[1]Institut des Sciences de l'Evolution (ISEM, UMR 5554 CNRS), Université
Montpellier II, Place E. Bataillon - CC 064 - 34095
[2] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
(LIRMM,UMR 5506, CNRS), Université Montpellier II 161, rue Ada, 34392
Montpellier Cedex 5, France,
{scornava, vberry}@lirmm.fr, Vincent.Ranwez@univ-montp2.fr

*Corresponding author

Abstract. Gene trees are leaf-labeled trees inferred from molecular sequences. Due to duplication events arising in genome evolution, gene trees usually have multiple copies of some labels, *i.e.* species. Inferring a species tree from a set of multi-labeled gene trees (MUL trees) is a well-known problem in computational biology. We propose a novel approach to tackle this problem, mainly to transform a collection of MUL trees into a collection of evolutionary trees, each containing single copies of labels. To that aim, we provide several algorithmic building stones and describe how they fit within a general species tree inference process. Most algorithms have a linear-time complexity, except for an FPT algorithm proposed for a problem that we show to be intractable.

Key words: Computational biology, evolutionary trees, graphs and graph transformation, duplications, polynomial and FPT algorithms, intractability proof.

1 Introduction

An evolutionary tree (or *phylogeny*), is a tree displaying the evolutionary history of a set of sequences or organisms. A *gene tree* is an evolutionary tree built by analyzing a gene family, *i.e.* homologous molecular sequences appearing in the genome of different organisms. Gene trees are primarily used to estimate *species trees*, *i.e.* trees displaying the evolutionary relationships among studied species. Unfortunately, most gene trees can significantly differ from the species tree for methodological or biological reasons, such as long branch attraction, lateral gene transfers, deep gene coalescence and, principally, gene duplications and losses [1]. For this reason, species trees are usually estimated from a large number of gene trees.

Inferring a species tree from gene trees is mostly done in a two-step approach. First, a micro-evolutionary model that takes into account events affecting individual sites is used to infer the gene trees. The species tree is then inferred on the basis of a macro-evolutionary model, *i.e.* minimizing the number of transfer, duplication and loss events [2–6]. To produce more biologically meaningful trees, unified models have been proposed in which the micro and macro-evolutionary dimensions are entangled [7–9]. However, it is difficult to determine how to incorporate events occurring on different spatial and temporal scales, as well as belonging to neutral and non-neutral processes, in a single model [9]. Lately, a hybrid approach has been proposed, where a first draft of a species tree is inferred with a micro-evolutionary model, the most uncertain parts of which are then corrected according to a macro-evolutionary model [9].

In this paper, we propose instead to take advantage of the very large number of gene trees present in recent phylogenomic projects to avoid entering into the detail of all possible macro-evolutionary scenarios (*e.g.* is a parsimony approach always justified? Should only the most parsimonious scenario be retained?). We propose to extract the non-ambiguous part of the topological information contained in the gene trees, *i.e.* that resulting from speciation events as opposed to duplication events, and then apply a traditional supertree method letting the weight of evidence decide in favor of one candidate species tree [10–12].

This approach is only possible when the number of gene trees is very large, and indeed this is now the case in projects such as the HOMOLENS database (`http://pbil.univ-lyon1.fr/databases/homolens.php`), storing several thousands of gene trees. In the release 04 of this database, 51% of gene families have paralogous sequences, *i.e.* sequences where duplications and losses have actually taken place. Currently, these gene families are discarded when inferring a supertree of the concerned species. Disentangling information derived from speciation events from that resulting from duplication events would thus provide more information for species tree inference.

Supertree methods combine source trees whose leaves are labeled with individual species into a larger species tree. The source trees are *single-labeled*, *i.e.* each species labels at most one leaf. Note that, by definition, the inferred supertree is also single-labeled. In contrast, gene trees are usually *multi-labeled*, *i.e.* a single species can label more than one leaf, since duplication events resulted in the presence of several copies of the genes in the species genomes. The task we therefore have to solve is to extract the largest amount of unambiguous topological information from the multi-labeled gene trees under the form of single-labeled trees. This paper presents a number of results in this direction, that all play a role in the general scheme that is fully described below. The rest of the paper details these results, though in a different order for the sake of dependancies between definitions.

First of all, we propose to separately preprocess MUL trees in order to remove their redundant parts with respect to speciation events. For this purpose, we extend the tree isomorphism algorithm of [13] making it applicable to MUL trees while preserving a linear running time (section 5). This algorithm is then applied to the pairs of subtrees hanging from duplication nodes in MUL trees. This preprocess lowers the number of duplication nodes in gene trees. We also give in passing a linear time algorithm to identify duplication nodes in MUL trees

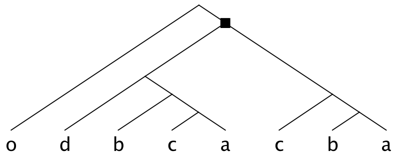

Fig. 1. An example of MUL tree with one odn indicated by a black square.

(section 4). For the gene trees that still have duplication nodes, we define a set $\mathcal{R}$ of triplets (binary rooted trees on three leaves [14, 12]) containing the topological information of a MUL tree that can be thought of as being unambiguously related to speciation events. We show that this set of triplets can be computed in $O(|\mathcal{R}|)$ time (section 2). When this set is compatible, the MUL tree contributes a coherent topological signal to build the species tree. In such a case, we can replace the MUL tree with a single-labeled tree representing its associated set of triplets by using the BUILD algorithm [14] (section 3). When a MUL tree is not auto-coherent, we propose to extract a maximum subtree that is both auto-coherent and free of duplication events. Surprisingly, this optimization problem can be solved in linear time (section 6). When extracting largest single-labeled subtrees from MUL trees it is possible to obtain an incompatible collection, when a compatible collection could have been obtained by choosing subtrees of MUL trees in a coordinated way. However, solving this problem is computationally harder, as we show by providing an NP-completeness proof (section 7).

2 Preliminaries

In this paper we focus on rooted binary multi-labeled (MUL) trees. Let M be a MUL tree and v a vertex of M. We denote by **s(v)** and **s'(v)** the two sons of v and by **sons(v)** the set {s(v),s'(v)}. We define by **subtree(v)** the subtree with v as root and by **L(v)** the multiset of labels of subtree(v). We denote by $L(M)$ the multiset $L(root(M))$.

Definition 1. *A node v of M is called an* ***observed duplication node*** *(odn) if the intersection of $L(s(v))$ and $L(s'(v))$ is not empty.*

Note that, for an odn v, $L(v)$ will always contain some label more than once. We denote by $\mathcal{D}(M)$ the set of odn. A label $l \in L(M)$ is a *repeated label* for M iff the label l occurs more than once in $L(M)$. We say that f is a *repeated leaf* for M iff $L(f)$ is a repeated label. For every three leaves we can have three different rooted tree binary shapes, called *triplets*. We denote by $AB|C$ the rooted tree that connects the pair of taxa (A, B) to C via the root.

We denote by $\mathcal{R}(M)$ the set of triplets of a (single/multi)labeled evolutionary tree M *i.e.* $\mathcal{R}(M) = \{ab|c$ *s.t.* there exist three leaf nodes $x, y, z \in M : L(x) = a$, $L(y) = b$, $L(z) = c$ and $lca(x, y) \neq (lca(x, z) = lca(y, z))\}$[1].

Definition 2. *Let M be a MUL tree. We define by $\mathcal{R}_{wd}(M)$ ($\mathcal{R}(M)$ without duplications) the set of triplets $ab|c$ of M* s.t. *there exist three leaf nodes $x, y, z \in M : L(x) = a$, $L(y) = b$, $L(z) = c$ and $lca(x, y) \notin \mathcal{D}(M)$, $lca(x, y, z) \notin \mathcal{D}(M)$, $lca(x, y) \neq (lca(x, z) = lca(y, z))$.*

Considering the MUL tree in Fig. 1 for example, $\mathcal{R}_{wd}(M) = \{ac|b, ac|d, ab|d, bc|d, ac|o, ab|o, ad|o, bc|o, cd|o, bd|o, ab|c\}$. Hence, not all the triplets of $\mathcal{R}(M)$ are kept. This is due to the fact that, once a duplication event occurred in a gene's history, the two copies of the gene evolved independently. The history of each copy is influenced by the species history but, considering them simultaneously may produce information unrelated to the species evolution. Therefore, it is more appropriate to discard the triplets mixing the histories of distinct copies of a gene.

$\mathcal{R}_{wd}(M)$ has an $O(n^3)$ size and can be computed in $O(n^3)$ time. Indeed, once the lca of all pairs of nodes in M are computed in $O(n)$ time (see [15, 16]), checking for three leaf nodes x, y, z of M if they satisfy Definition 2 can be done in $O(1)$ time, thus in $O(n^3)$ for all triplets of leaves in M.

3 Auto-coherency of a MUL tree

Let M be a multi-labeled gene tree. Since M contains several copies of the same gene, we can wonder whether the evolutionary signal of each copy is coherent or not.

Definition 3. *A MUL tree M is said to be **auto-coherent** if there exists a single-labeled tree T such that $\mathcal{R}_{wd}(M) \subseteq \mathcal{R}(T)$.*

In the case of an auto-coherent MUL tree, we know that we can find a tree T containing all the information in $\mathcal{R}_{wd}(M)$, *i.e.* the information of M that is considered reliable. To find such a tree, we use the AncestralBuild algorithm of [17]. For a set of triplets $\mathcal{R}_{wd}(M)$, this algorithm indicates in $O(|\mathcal{R}_{wd}(M)| \cdot \log^2 n)$ time whether there exists a tree T *s.t.* $\mathcal{R}_{wd}(M) \subseteq \mathcal{R}(T)$ and returns T in case of a positive answer. At present, it is not clear whether an implicit representation of $\mathcal{R}_{wd}(M)$ using only $O(n)$ triplets, as that reported in [18] to encode an binary tree, could be used here. Indeed, the presence of multiple occurrences of labels might lead to obtain a small compatible sets of triplets, while the whole set $\mathcal{R}_{wd}(M)$ would be incompatible.

4 Computing $\mathcal{D}(M)$ in linear time

The easiest way to compute $\mathcal{D}(M)$ is checking for each node v if the sets $L(s(v))$ and $L(s'(v))$ have at least one label in common; in the case of a positive answer, v is inserted in $\mathcal{D}(M)$. The complexity of this approach is $O(n^2)$, since it requires

[1] $lca(x, y)$ denotes the least common ancestor of nodes x and y, i.e. the lowest node in the tree that has both x and y as descendants.

computing $O(n)$ intersections of two lists of $O(n)$ elements. The algorithm 1 uses the lca to find the set of odn $\mathcal{D}(M)$ and requires only linear time. To demonstrate the correctness of algorithm 1, we need to determine some relationships between the lca and the odn.

Lemma 1. *A node is an odn if and only if it is the lca of at least two repeated leaves m and p.*

Proof. Indeed, from the definition 1, v is an odn iff L(s(v)) $\cap$ L(s'(v)) $\neq \emptyset$. Therefore, $\exists m \in subtree(s(v))$ and $\exists p \in subtree(s'(v))$: $L(m) = L(p)$. Thus v is a common ancestor of the two leaves m and p with the same label. Now, m and p belong to two different subtrees having v as father ($m \in subtree(s(v))$ and $p \in subtree(s'(v))$), hence v is their lowest common ancestor in M. Reciprocally, if v is the lca of two leaves m and p with the same label, this means that L(s(v)) $\cap$ L(s'(v)) $\neq \emptyset$, then v is an odn according to definition 1. □

According to Lemma 1, we can search for the lca of any two leaves m and p with the same label. To determine the lca between multiple pairs of nodes, one can use an algorithm in [15] which preprocesses a data structure in O(n) time, where n is the number of nodes and returns the lca of any two specific nodes from the data structure in O(1). We still have O(n2) of these couples, and even constant time for each gives an O(n2) total complexity. However, since there are only O(n) odn, checking the lca of any pair of leaves computes the same lca several times. A smarter approach is used in algorithm 1: first of all, the subtrees of M are ordered from the left to the right in an arbitrary way. Then, each repeated leaf, starting from the left of the tree and moving to the right, is tagged with the repeated label followed by its occurrence number. Then, for each repeated label e, the lca of any two successive occurrences of e, e_i and e_{i+1} is inserted in $\mathcal{D}(M)$. This leads to a linear time complexity. Indeed, we have $O(n)$ of these couples since each leaf of M is involved in at most two pairs (e_i, e_{i+1}).

Algorithm 1: CompDuplicationNodes(r)

```
Data: A MUL tree $M$.
Result: A set of odn $\mathcal{D}(M)$.
Order $M$ in an arbitrary way. In this order, tag each duplicated leaf with the
repeated label followed by its occurrence number. Compute the lca for each
couple of leaves.
$\mathcal{D}(M) \leftarrow \emptyset$;
foreach (repeated label $e$) do
    foreach ($\{e_j, e_{j+1}\}$) do $\mathcal{D}(M) \leftarrow lca(e_j, e_{j+1})$;
return $\mathcal{D}(M)$;
```

The correctness of algorithm 1 is justified by Lemma 2 showing that algorithm 1 retrieves all the odn of M.

Lemma 2. *Let M be a MUL tree. For each odn v, $\exists$ two successive occurrences of a label e denoted by e_i and e_{i+1} s.t. $v = lca(e_i, e_{i+1})$.*

Proof. Given an odn v, there exists at least one label e present in both subtrees $s(v)$ and $s'(v)$. We denote by A the set of leaves a_i *s.t.* $a_i \in subtree(s(v))$ and

$L(a_i) = e$ and by B the set of leaves b_j *s.t.* $b_j \in subtree(s'(v))$ and $L(b_j) = e$. If we take the last element of B ($b_{|B|}$) and the first one of A (a_1), we know that v is their lca. Additionally, due to the way we tagged M, we know that there is no other occurrence of the label e between $b_{|B|}$ and a_1. Indeed, if there was another leaf x labeled with e, it would be either in $s(v)$ (and then $x = a_1$) or in $s'(v)$ (and then $x = b_{|B|}$). Then $b_{|B|}$ and a_1 are two successive occurrences of the same label and their lca is the node v. □

5 Isomorphic subtrees

Definition 4. *Two rooted trees T_1 , T_2 are isomorphic (denoted by T_1=T_2) iff there exists a one-to-one mapping from the nodes of T_1 onto the nodes of T_2 preserving leaf labels and descendancy.*

We are interested in testing if, for each odn v, the two subtrees $s(v)$ and $s'(v)$ are isomorphic or not. In the positive, we can prune one of the two isomorphic subtrees without losing any information contained in $\mathcal{R}_{wd}$ and eliminate the odn v, as in the example of Fig. 2. For detecting isomorphism of *multi-labelled* trees, we propose Algorithm 2, an extension of the CHECK-ISOMORPHISM-OR-FIND-CONFLICT algorithm [19]. Indeed, the latter does not deal with MUL trees. Alternatively, we could have proposed an appropriate variant of the tree isomorphism algorithm detailed in [20]. However, such an algorithm would likely have been less space efficient than the one we present here due to a number of string sorting steps using a number of queues and lists to ensure linear running time.

A node that has only two leaves as children is called **cherry**. In the case of single-labeled trees we have the following lemma:

Lemma 3. *[13] Let T_1, T_2 be two isomorphic trees and let c_1 be a cherry in T_1. Then, there is a cherry $c_2 \in T_2$ s.t. $L(c_1) = L(c_2)$.*

In the case of MUL trees, we can have several copies of the same cherry. We call a **multiple cherry** the list of cherries on the same two labels. For a multiple cherry mc, we note $|mc|$ the number of occurences of the cherry in its tree.

Lemma 4. *Let M_1, M_2 be two isomorphic MUL trees and let mc_1 be a multiple cherry in M_1. Then, there is a multiple cherry $mc_2 \in M_2$ s.t. $L(mc_1) = L(mc_2)$ and $|mc_1| = |mc_2|$.*

The proof is inspired from that of Lemma 3 in [13] and left to the reader.

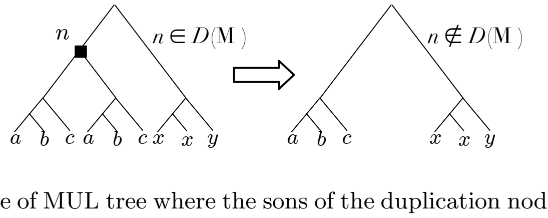

Fig. 2. An example of MUL tree where the sons of the duplication node are isomorphic.

Algorithm 2: CheckIsomorphismMULTree(M_1,M_2)

```
Data: Two MUL tree $M_1$ and $M_2$.
Result: TRUE if $M_1$ and $M_2$ are isomorphic, FALSE otherwise.
Let $L_{mc}$ be the list of multiple cherries in $M_1$ and $M_2$. Let $H$ be the hashtable
where each $mc \in L_{mc}$ is a key. To each $H(mc)$, we associates two lists $O_2(mc)$
and $O_1(mc)$, resp. of the occurrences of $mc$ in $M_1$ and $M_2$;
while ($L_{mc} \neq \emptyset$) do
    $mc \leftarrow$ the first multiple cherry in $L_{mc}$; delete $mc$ form $L_{mc}$;
    if ($O_2(mc) = O_1(mc)$ ) then
        Turn all cherries in $O_2(mc)$ and $O_1(mc)$ into leaves to which the same
        label chosen in $L(mc)$ is assigned;
        add the new multiple cherries in $L_{mc}$ and $H$;
    else return FALSE;
return TRUE;
```

5.1 Outline of the algorithm

First of all, we find all the multiple cherries for the MUL trees M_1 and M_2. We store them in the list L_{mc} using a simple linked list. Additionally, we use a hashtable H where each $mc \in L_{mc}$ is a key. H associates to each multiple cherry mc two linked lists, $O_1(mc)$ and $O_2(mc)$, storing pointers to nodes of M_1 and M_2 respectively that correspond to the occurrences of mc. The multiple cherries of a MUL tree are then examined in a bottom-up process. Given a multiple cherry mc in L_{mc} we check if the size of $O_1(mc)$ is the same as that of $O_2(mc)$. If this is not the case, we have found a multiple cherry for which we do not have the same number of occurrences in M_1 and M_2. In this instance, M_1 and M_2 are not isomorphic (Lemma 4) and the algorithm returns FALSE. Otherwise we turn all the cherries in $O_1(mc)$ and $O_2(mc)$ into leaves to which the same label, chosen arbitrarily in $L(mc)$, is assigned. This modification of M_1 and M_2 can turn the fathers of some cherries in $O_1(mc)$ and $O_2(mc)$ into new cherries. Then L_{mc} is updated and the processing of cherries in M_1 is iterated until both MUL trees are reduced to a single leaf with the same label if M_1 and M_2 are isomorphic, or a FALSE statement is returned.

Theorem 1. *Let M_1 and M_2 be two rooted MUL trees with $L(M_1) = L(M_2)$ of cardinality n. In time $O(n)$, algorithm 3 returns TRUE if M_1 and M_2 are isomorphic, FALSE otherwise.*

Proof. This algorithm is an extension of the CHECK-ISOMORPHISM-OR-FIND-CONFLICT algorithm [19] applicable to MUL trees. We show here that we can keep a linear time execution, using supplementary data structures.
A simple depth-first search of trees M_1 and M_2 initializes L_{mc} and H in $O(n)$ time. At each iteration of the algorithm, choosing a multiple cherry mc to process is done in $O(1)$ by removing the first element mc of L_{mc}. H then provides in $O(1)$ the lists $O_1(mc)$ and $O_2(mc)$ of its occurrences in the trees. Checking that these lists have the same number of elements is proportional to the number

of nodes they contain, hence costs $O(n)$ amortized time, as each node is only once in such a list, and the list is processed once during the whole algorithm. Replacing all occurrences of mc by a previously chosen label $l \in L(mc)$ is done in $O(n)$ amortized time, since each replacement is a local operation replacing three nodes by one in a tree and at most $O(n)$ such replacements can take place in a tree to reduce it down to a single node (the algorithm stops when this situation is reached). Reducing a cherry can create a new occurrence $o_{mc'}$ of a cherry mc'. Checking in $O(1)$ time if mc' is a key in H allows to know whether occurrences of mc' have already been encountered or not. In the positive, we simply add $o_{mc'}$ to the beginning of the list $O_1(mc)$ (if $o_{mc'} \in M_1$) or $O_2(mc)$ (if $o_{mc'} \in M_2$), requiring $O(1)$ time. In the negative, we add mc' to the beginning of L_{mc}, create a new entry in H for mc', and initialize the associated lists $O_1(mc)$ and $O_2(mc)$ so that one contains $o_{mc'}$ and the other is the empty list. Again, this requires only $O(1)$ time. Thus, performing all operations required by the algorithm globally costs $O(n)$ time.

Apply algorithm 2 to each $subtree(s(v))$ and $subtree(s'(v))$ of each odn of a MUL tree M in a bottom-up approach requires $O(dn)$ time, where d is the number of duplication nodes in M.

6 Computing a largest duplication-free subtree of a MUL tree

If a MUL tree is not auto-coherent, identifying duplication nodes allows for the discrimination of leaves representing orthologous and paralogous sequences. Since only orthologous sequence history reflects the species history, a natural question is to determine the most informative sequence set for a given gene. As long as the gene tree contains odn, it will also contain leaves representing paralogous sequences. Yet, if for each node $v \in \mathcal{D}(M)$ of M we choose to keep either $s(v)$ or $s'(v)$, we obtain a pruned single-labeled tree containing only apparent orthologous sequences (observed paralogous have been removed by pruning nodes). Note that the so obtained single-labeled tree is auto-coherent by definition.

Definition 5. *Let M be a MUL tree. We say that T is obtained by (duplication)* pruning *M iff T is obtained from M choosing for each odn v either $s(v)$ or $s'(v)$. We denote this operation by the symbol $\lesssim$.*

One can wonder, for a non auto-coherent MUL tree M, what is the most informative single-labeled tree T *s.t.* $T \lesssim M$. We define this problem as the $MIPT$ (Most Informative Pruned Tree) problem.
To evaluate the informativeness of a tree we can use either the number of triplets of T (see [21, 22, 11]) that, for binary trees, depends only on the number of leaves, or the CIC criterion (see [23, 12]). The CIC of a not fully resolved and incomplete[2] tree T with $|L(T)|$ leaves among the n possible is a function of both the

[2] A tree is called incomplete when it misses some taxa.

Algorithm 3: pruning(v,M,$\mathcal{D}(M)$)

```
Data: A node $v$, a MUL tree $M$, and a set of odn $\mathcal{D}(M)$.
Result: The most informative MUL tree $M'$ s.t. $subtree(v)_{M'} \lesssim subtree(v)_M$.
foreach ($m \in sons(v)$) do pruning($m$,$M$,$\mathcal{D}(M)$);
if ($v \in \mathcal{D}(M)$) then
    if ($|L(s(v))| > |L(s'(v))|$) then $v \leftarrow s(v)$;
    else $v \leftarrow s'(v)$;
    $\mathcal{D}(M) \leftarrow \mathcal{D}(M) - \{v\}$;
return $M$;
```

number $n_R(T, n)$ of fully resolved trees T' on $L(T)$ such that $\mathcal{R}(T) \subseteq \mathcal{R}(T')$ and the number $n_R(n)$ of fully resolved trees on n leaves. More precisely,

$$CIC(T, n) = -\log\Big(n_R(T, n)/n_R(n)\Big)$$

In the case of binary trees, $n_R(T, n)$ depends only on the number of source taxa missing in T since T does not contain multifurcations. Thus, dealing with binary trees, maximizing the information of a tree (*i.e.* maximizing the number of triplets or minimizing the CIC value) consists in finding the tree with the largest number of leaves. A natural approach for the MIPT problem for binary MUL trees is an algorithm that, after having computed $\mathcal{D}(M)$, uses the bottom-up algorithm 3, with v =root(M), to keep the most informative subtree between $subtree(s(m))$ and $subtree(s'(m))$, for each odn m.

Theorem 2. *Let M a MUL tree on a set of leaves of cardinality n. In time $O(n)$, pruning(M,root(M),$\mathcal{D}(M)$) returns the most informative tree T* s.t. *$T \lesssim M$.*

Proof. First of all, it's obvious that pruning(M,root(M),$\mathcal{D}(M)$) returns a tree. Indeed, if for each odn v only one node between $s(v)$ and $s'(v)$ is kept, at the end of the bottom-up procedure one copy of each duplicated leaf is present in the modified M. Now, we have to show that the resulting tree is the most informative tree *s.t.* $T \lesssim M$, *i.e.* the tree with as many leaves as possible. For an odn v that is the ancestor of other duplication nodes, the choices made for $s(v)$ do not influence the choices for $s'(v)$ since for each duplication node we can keep only one of the two subtrees, the most populous one. Thus we can search for the best set of choices left/right for $s(v)$ and $s'(v)$ independently and then choose the most populous pruned subtree between $s(v)$ and $s'(v)$. Iterating recursively this reasoning, we demonstrate that the tree obtained by Algorithm 3 is the most informative tree T *s.t.* $T \lesssim M$.

The computation of the set of odn $\mathcal{D}(M)$ takes linear time. The subroutine $pruning(M, root(\mathrm{M}), \mathcal{D}(M))$ requires a tree walk, thus the time complexity of Algorithm 3 is $O(n)$. □

7 The compatibility issue of single-labeled subtrees obtained from MUL trees

We can also ask if it is possible, given a collection of MUL tree $\mathcal{M}$, to discriminate leaves representing orthologous and paralogous sequences in a gene tree using the information contained in the other gene trees to obtain a compatible forest $\mathcal{T}$, *i.e.* a forest for which there exists a tree T *s.t.* $\cup_{T_i \in \mathcal{T}} \mathcal{R}(T_i) \subseteq \mathcal{R}(T)$. We denote this problem by EPCF, Existence of a Pruned and Compatible Forest. Unfortunately, the EPCF problem is NP-complete.

EPCF	**Instance** : A set of leaves X and a collection $\mathcal{M} = \{M_1, \cdots M_k\}$ of MUL trees on X. **Question** : $\exists$ a set S of choices left/right, $S : \mathcal{M} \rightarrow \mathcal{T}$, with $\mathcal{T} = \{T_1, \cdots T_k\}$ *s.t.* $T_i \lesssim M_i$ and $\mathcal{T}$ is compatible?

Theorem 3. *The EPCF problem is NP-complete.*

Proof. We start by proving that EPCF is in NP, *i.e.* checking if a set S of choices left/right is a solution for the instance $\mathcal{I} = (\mathcal{M}, X)$ can be done in polynomial time. First of all, for each MUL tree $M_j \in \mathcal{M}$, we place the choices left/right on M_j, *i.e.* we discard the subtrees not chosen, obtaining a forest of trees $\mathcal{T}$. We check then the compatibility of $\mathcal{T}$ with the Aho graph[14]. Constructing this graph can be done in polynomial time.

Given that EPCF is in NP, we use a reduction of 3-SAT to EPCF to demonstrate that it is NP-complete.

3-SAT	**Instance** : A boolean expression $\mathcal{C} = (C_1 \wedge C_2 \wedge \cdots \wedge C_n)$ on a finite set $L = \{l_1, l_2, \cdots, l_m\}$ of variables with $C_j = (a \vee b \vee c)$ where $\{a, b, c\} \in \{l_1, l_2, \cdots, l_m, \overline{l_1}, \overline{l_2} \cdots, \overline{l_m}\}$ **Question** : $\exists$ a truth assignment for L that satisfies all C_j in C ?

We need to show that every instance of 3-SAT can be transformed into an instance of EPCF; then we will show that given an instance $\mathcal{I} = (\mathcal{C}, L)$ of 3-SAT, $\mathcal{I}$ is a positive instance, *i.e.* an instance for which a solution exists, iff the corresponding instance for EPCF is positive.

Given an instance $\mathcal{I} = (\mathcal{C}, L)$ of 3-SAT, we build an instance $\mathcal{I}' = (\mathcal{M}, X)$ of EPCF associating to each l_i in L the binary tree[3] $T(l_i) = (((x_i, y_i), z_i), d)$ and to $\overline{l_i}$ the binary tree $T(\overline{l_i}) = (((z_i, y_i), x_i), d)$ (see Fig. 3 for an example).

[3] $T(l_i)$ is expressed in the Newick format.

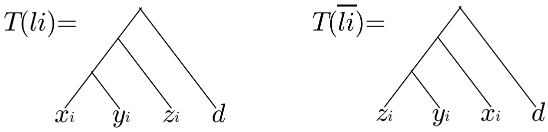

Fig. 3. Binary trees on four leaves associated to l_i and to $\overline{l_i}$.

The set of subtrees $\Big\{T(a) \mid a \in \{l_1, l_2, \cdots, l_m, \overline{l_1}, \overline{l_2}, \cdots, \overline{l_m}\}\Big\}$ is denoted by $\mathcal{T}_L$. Then, for each clause $C_j = (a \vee b \vee c)$ in C, a binary MUL tree M_j is built, formed by three subtrees $((T(a), T(b)), T(c))$. Note that M_j has exactly two duplication nodes due to the presence of d in $T(a)$, $T(b)$ and $T(c)$, so that any left/right choice of M_j will reduce it to either $T(a)$, $T(b)$ or $T(c)$. In Fig. 4 an example of a MUL tree built from a clause. In this way we obtain a forest of MUL trees $\mathcal{M}$ on the leaf set $X = \Big\{\big\{\bigcup_{i=1}^{m}\{x_i, y_i, z_i\}\big\} \cup \{d\}\Big\}$, *i.e.* an instance of the EPCF problem. Clearly $\mathcal{M}$ can be built in polynomial time.

We now need to show that a positive instance of 3-SAT gives a positive instance of EPCF through the previous transformation. Having a positive instance for 3-SAT implies that for each $C_j \in \mathcal{C}$ with $C_j = (a \vee b \vee c)$, at least one of the three literals is TRUE. Without loss of generality, let us suppose that a is TRUE. Then in the MUL tree M_j corresponding to C_j we set the choice left/right so that only the subtree $T(a)$ is kept. We then obtain a forest $\mathcal{T}$ that is a subset of $\mathcal{T}_L$. We need to prove that $\mathcal{T}$ is compatible. Let $\tilde{T}(a)$ denote the tree $T(a)|(L(T(a)) - \{d\})$[4] and $\tilde{\mathcal{T}}$ the forest composed by all trees $\{\tilde{T}(a)|T(a) \in \mathcal{T}\}$. Then, we can build a tree $T_s = (\tilde{T}_1, \tilde{T}_2, \cdots, \tilde{T}_{|\tilde{\mathcal{T}}|}, d)$. Since l_i cannot have the value TRUE and FALSE at the same time, we have either $T(l_i)$ or $T(\overline{l_i})$ in $\mathcal{T}$. The tree T_s is therefore a

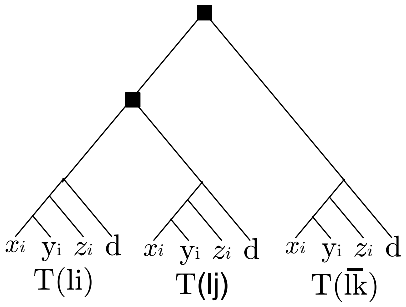

Fig. 4. MUL tree built from the clause $\{l_i \vee l_j \vee \overline{l_k}\}$. Odn are indicated by black squares.

[4] Given a tree T and a label set S, we denote by $T|S$ the restriction of T to the set S.

single-labeled tree. Moreover, by construction $T_s|(L(T(a))$ is identical to $T(a)$, for all $T(a)$ in $\mathcal{T}$ ensuring that $\bigcup_{T_i \in \mathcal{T}} \mathcal{R}(T_i) \subseteq \mathcal{R}(T_s)$. Thus $\mathcal{T}$ is compatible.

Now, the only thing left to prove is that a positive instance of EPCF gives a positive instance of 3-SAT.
The repetition of the taxon d in each subtree makes the two nodes connecting the subtrees in each M_j be odn. Thus a left/right choice set S reduces each M_j in $\mathcal{M}$ into a tree $T(a) \in \mathcal{T}_L$, providing the forest $\mathcal{T}$. Setting the value of a to TRUE ensures that the clause C_j corresponding to M_j is TRUE. This can be done simultaneously for all clauses $\in \mathcal{C}$ since the forest compatibility implies that there is no contradiction among the trees in $\mathcal{T}$, all the more so direct contradictions. Then, either $T(l_i)$ or $T(\overline{l_i})$ is in $\mathcal{T}$. This ensures us that either l_i or $\overline{l_i}$ is assigned to TRUE, but not both. □

Note that the problem to find the most informative forest $\mathcal{T} = \{T_1, \cdots T_k\}$ *s.t.* $T_i \lesssim M_i$ and $\mathcal{T}$ is compatible, denoted by MIPCF (Most Informative Pruned and Compatible Forest) is FPT. Indeed, analyzing all possible scenarios of choices left/right is FPT on the total number of duplication nodes in $\mathcal{M}$.

8 Conclusions

In this paper we presented several algorithms to transform multi-labeled evolutionary trees into single-labeled ones so that they can be used by all existent supertree methods. In the future we plan to go further and extend the algorithm FIND-REFINEMENT-OR-CONFICT in [19] applicable to MUL trees. We are also working on a more sophisticated FPT algorithm for the MIPCF problem.

References

1. J.A. Cotton and R.D.M. Page. Rates and patterns of gene duplication and loss in the human genome. In Royal Society of London. Biological science, editor, *Proceedings*, volume 272, pages 277–283, 2005.
2. B. Ma, M. Li, and L. Zhang. From gene trees to species trees. *SIAM J. Comput*, 30:729—752, 2000.
3. M. T. Hallett and J. Lagergren. New algorithms for the duplication-loss model. In *RECOMB2000, Fourth Annual International Conference on Computational Molecular Biology*, pages 138–146, 2000.
4. K. Chen, D. Durand, and M. Farach-Colton. Notung: a program for dating gene duplications and optimizing gene family trees. *J Comput Biol*, 7:429—444, 2000.
5. B. Vernot, M. Stolzer, A. Goldman, and D. Durand. Reconciliation with non-binary species trees. *J Comput Biol*, 15(8):981—1006, 2008.
6. C. Chauve, J. P. Doyon, and N. El-Mabrouk. Gene family evolution by duplication, speciation, and loss. *J Comput Biol*, 15(8):1043—1062, 2008.
7. M. Goodman, J. Czelusniak, G.W. Moore, Romero-Herrera A.E., and G. Matsuda. Fitting the Gene Lineage into its Species Lineage, a Parsimony Strategy Illustrated by Cladograms Constructed from Globin Sequences. *Systematic Zoology*, 28(2):132—163, 1979.

8. L. Arvestad, A.C. Berglund, J. Lagergren, and B. Sennblad. Bayesian gene/species tree reconciliation and orthology analysis using MCMC. *Bioinformatics*, 19 Suppl 1:7—15, 2003.
9. D. Durand, B.V. Halldrsson, and B. Vernot. A hybrid micro-macroevolutionary approach to gene tree reconstruction. *J. Comput. Biol.*, 13:320—335, Mar 2006.
10. B. R. Baum and M. A. Ragan. The MRP method. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees: combining information to reveal the Tree of Life*, pages 17–34. Kluwer, 2004.
11. R. D. M. Page. Modified mincut supertrees. In R. Guigó and D. Gusfield, editors, *Proceedings of the 2nd International Workshop on Algorithms in Bioinformatics (WABI'02)*, pages 537–552, 2002.
12. C. Scornavacca, V. Berry, V. Lefort, E. J. P. Douzery, and V. Ranwez. Physic_ist: cleaning source trees to infer more informative supertrees. *BMC Bioinformatics*, 9(8):413, 2008.
13. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:12–28, 1991.
14. A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comp.*, 10(3):405–421, 1981.
15. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
16. M. A. Bender and M. Farach-Colton. The lca problem revisited. In Springer-Verlag, editor, *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, 2000.
17. V. Berry and C. Semple. Fast computation of supertrees for compatible phylogenies with nested taxa. *Syst. Biol.*, 55:270–288, 2006.
18. M.R Henzinger, V. King, and T.J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24:1–13, 1999.
19. V. Berry and F. Nicolas. Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(3):289–302, 2006.
20. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1974.
21. V. Ranwez, V. Berry, A. Criscuolo, P.H. Fabre, S. Guillemot, C. Scornavacca, and E.J. Douzery. PhySIC: a veto supertree method with desirable properties. *Syst. Biol.*, 56:798–817, Oct 2007.
22. C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Appl. Math.*, 105:147–158, 2000.
23. J. Thorley, M. Wilkinson, and M. Charleston. The information content of consensus trees. In A. Rizzi, M. Vichi, and H.-H.. Bock, editors, *Advances in Data Science and Classification. Studies in Classification, Data Analysis, and Knowledge Organization*, pages 91–98. 1998.