

Fast Computation of Supertrees for Compatible Phylogenies with Nested Taxa

Vincent Berry¹ and Charles Semple^{2,3}

14 November 2005

¹Département Informatique, L.I.R.M.M. - C.N.R.S., 161 rue Ada, 34392 Montpellier
Cedex 5, France
vberry@lirmm.fr

B₄ D. Fast computation for deciding compatibility is essential if one is to make

Introduction

Supertree methods are a fundamental and practical way of inferring phylogenies. Generally speaking, these methods amalgamate a collection of "source" trees on overlapping subsets of taxa into a single parent tree that contains the taxa of all of the source trees. This parent tree is called a supertree. This approach to constructing evolutionary trees is particularly appealing because it allows the inference of an evolutionary scenario from a combination of analyses differing in the set of taxa they encompass as well as in the primary data from which they were conducted (for example, molecular or mo

basic property that one would always like is that of consistency; that is, if the source trees carry no conflicting information, then the supertree returned by the method displays each of the source trees. Because the property of consistency is such a compelling property, many general supertree methods dealing with leaf-labelled trees (respectively, nested-taxa trees) are likely either to have $B_{\mathcal{D}}$ (respectively, $ANCESTRY_{\mathcal{D}}$) as a subroutine or to be a variant of $B_{\mathcal{D}}$ (respectively, $ANCESTRY_{\mathcal{D}}$). Indeed, this is already the case for some general methods: both $MNC_{\mathcal{D}}^{PE TREE}$ (Semple and Steel, 2000) method and its modified version (Page, 2002) are variants of $B_{\mathcal{D}}$ and, more recently, Daniel and Semple (2005) describe a class of general supertree methods for nested-taxa source trees that is a variant of $ANCESTRY_{\mathcal{D}}$. (This class and more particularly the underlying general supertree method $NESTED_{\mathcal{D}}^{PE TREE}$ is described further in the last section.) Moreover, these all-or-nothing algorithms can be repeatedly used in simple schemes to extract compatible parts out of a collection of incompatible source trees. We highlight two examples of such schemes in the discussion part of this paper.

Reasonable running time. Given the amount of information in current tree databases, it is not unreasonable to try to amalgamate hundreds of trees that collectively contain thousands of taxa and, consequently, fast algorithms are essential. To deal with fully-resolved (i.e., binary) leaf-labelled trees, Henzinger et al. (1999) proposed a fast implementation of $B_{\mathcal{D}}$ that runs in $(\frac{1}{2})$ time, where n is the total sum of the number of nodes in each of the source trees and $(\frac{1}{2})$

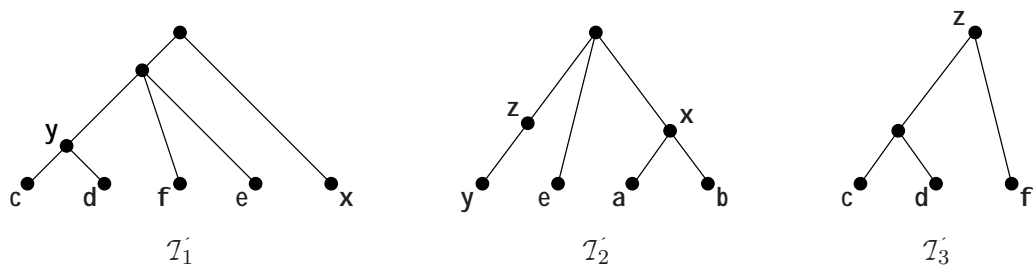


Figure 1: A compatible collection of rooted semi-labelled trees.

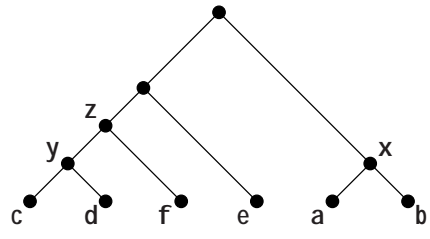
et al. (2005).

Preliminaries

In this section, we describe some concepts that are frequently used in the paper. For further details, we refer the interested reader to Semple and Steel (2003).

Phylogenies. The degree of a node in a graph (or, in particular, a tree) is the number of edges incident with it. We denote the degree of v by $d(v)$. Essentially, a rooted phylogenetic X -tree is a rooted tree whose leaves are labelled with the elements of a set X of taxa. We assume X is

1. We will often write a rooted semi-labelled \mathcal{S} -tree for a rooted semi-labelled tree on \mathcal{S} .
 - . Observe that rooted phylogenetic trees are special types of rooted semi-labelled trees.
3. To simplify matters and because we see no practical reason for nodes in the source trees to be assigned more than one taxa of \mathcal{S} , we will assume throughout the paper that all rooted semi-labelled trees that are source trees are singularly labelled. However, we note that the upgrade of the results in this paper to non-singular rooted semi-labelled trees is straightforward. Note that this remark about singular labelling does not apply to the output tree, where it is quite possible that some nodes are labelled with more than one taxa. For instance, a node joining human and chimp on a source tree that contains no other mammals could be equally labelled as anything from "hominoid" to "primate" to "mammal". This multiple listing of a node then becomes very important



directed into and the outdegree of is the number of arcs directed out of

with the cluster (\mathcal{C}) and successively breaking it down into disjoint subclusters. The way in which the clusters are broken up is decided by the descendency graph which itself is successively broken into node induced subgraphs. The algorithm either completes the construction of such a tree or returns not ancestrally compatible if at some iteration the associated node induced subgraph of the descendency graph has no nodes which have indegree zero and no incident edges.

Algorithm ANCESTRAL BUILDING

Input: A collection \mathcal{T} of rooted semi-labelled trees on X .

Output: A rooted semi-labelled tree \mathcal{T}' that ancestrally displays \mathcal{T} or the statement is not ancestrally compatible.

- 1. Construct a collection \mathcal{T}' of rooted fully-labelled trees from \mathcal{T} by adding distinct new labels to the unlabelled nodes in the trees of the collection.
- 2. Construct the descendency graph $D(\mathcal{T}')$ of \mathcal{T}' .
- 3. Call the subroutine **ANCESTRAL BUILDING**($D(\mathcal{T}')$).
- 4. If **ANCESTRAL BUILDING** returns *no possible labelling*, then return *is not ancestrally compatible*. Otherwise, return the semi-labelled tree \mathcal{T}' returned by **ANCESTRAL BUILDING** with the added labels removed.

Algorithm ANCESTRAL BUILDING

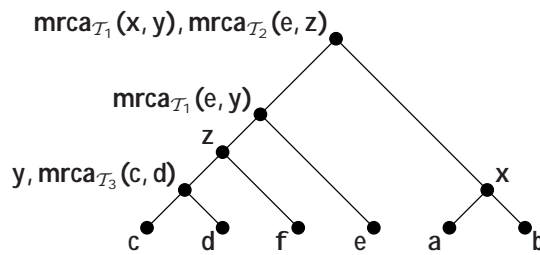


Figure 5: The rooted semi-labelled tree returned by `ANCESTOR-BUILD` as described in Example 2.

Remark.

1. With respect to descendency, the added labels act as necessary “place holders” for unlabelled nodes.
- The recursive calls performed at Step 4 in `ANCESTOR-BUILD` consider disjoint node induced subgraphs so that the processes applied to these subgraphs in subsequent iterations are independent from one subgraph to another.

Example As an example of `ANCESTOR-BUILD` applied to a collection of rooted semi-labelled trees, consider the collection of trees shown in Figure 1. Suppose that Step 1 constructs the collection \mathcal{T}' of rooted fully-labelled trees shown in Figure 3. Now Step 2 builds the descendency graph (\mathcal{D}) as shown in Figure 4. On the first iteration of `ANCESTOR-BUILD`,

`ANCESTOR-BUILD`, \mathcal{T}' 254.64(a)-2.26269(p)1.9672Tf 55.55989(e)R1111.95*1itra9.7846269(m)2.92434(p36340.18(d)-339.309(a)-7.683(a)2.67269(d)5.2067(b)1.74746(c)8935922857(0)8.0251(t)0.4653.56382(4)9329(a)7.96048(5)0.9

removes only one node in (G) , in which case the subroutine is executed (n) times. The computation time is dominated by the cost of Steps 3(b) and 3(c) in the subroutine. Finding the connected arc components of a digraph is linear in the number of its nodes and arcs. Thus, assuming that in the worst case only a constant number of edges are removed with each node, an execution of Step 3(b) can require up to (n^2) time to process the restriction of (G) it is considering. Because of the (n) executions of the subroutine, this leads to an overall running time of (n^3) for Step 3(b). Finding edges across different arc components in Step 3(c) can necessitate at worst to examine the (n^2) edges of the graph. This leads to an overall running time of (n^3) for Step 3(c). Noting that $(G) = (G)$ gives the final result. \square

Remark. It is worth noting that the running time of A_N

•

For the proof of (ii), suppose that ANCESTOR_BUILD (using the restricted descendant graph) outputs a rooted semi-labelled tree \mathcal{T}' . We show that \mathcal{T}' ancestrally displays \mathcal{G} . Let \mathcal{T}'_1 be an element of \mathcal{P} . By Lemma 2.1 (Bordewich et al., 2005), it suffices to show, for all $\ell \in \mathcal{L}(\mathcal{T}'_1)$ that (I) if ℓ is a descendant label of ℓ' in \mathcal{T}'_1 , then ℓ is a descendant label of ℓ' in \mathcal{T}' , and (II) if ℓ and ℓ'' are non-comparable in \mathcal{T}'_1 , then ℓ and ℓ'' are non-comparable in \mathcal{T}' .

The argument for (I) is very similar to the corresponding argument in the proof of Theorem 4.1(ii) (Daniel and Semple, 2004), and so we omit it. To show (II), suppose that ℓ and ℓ'' are not comparable in \mathcal{T}'_1 . Assume that $\mathcal{T}'_1 = (\mathcal{T}_1; \mathcal{T}_2)$. Let ℓ be the node in \mathcal{T}_1 that is the most recent common ancestor of $\ell_1(\ell)$ and $\ell_2(\ell)$. By the construction of ANCESTOR_BUILD , there is a pair of children, ℓ_1 and ℓ_2 , say, of the label labelling in \mathcal{T}_1 such that ℓ and ℓ'' are joined by an edge, and ℓ is an ancestor label of ℓ_1 , and ℓ is an ancestor label of ℓ_2 . Since we eventually output a tree, this edge is eventually deleted, but not until ℓ_1 and ℓ_2 , and hence ℓ and ℓ'' , are in separate arc components of some restriction of ANCESTOR_BUILD . It now follows that ℓ and ℓ'' are not comparable in \mathcal{T}' . \square

Remark. Let \mathcal{T} be a collection of rooted semi-labelled trees with $\mathcal{L}(\mathcal{T}) = \mathcal{L}$ and $\mathcal{I}(\mathcal{T}) = \mathcal{I}$, and let \mathcal{T}' be a collection of fully-labelled trees that is obtained from \mathcal{T} by adding distinct new labels. Let $\mathcal{G} = \sum_{\mathcal{T} \in \mathcal{P}} \mathcal{T}$. Then the mixed graph $\text{ANCESTOR_BUILD}(\mathcal{G})$ contains \mathcal{L} nodes and arcs. However, the number of edges in $\text{ANCESTOR_BUILD}(\mathcal{G})$ is a function of the degree of the nodes in the source trees. In particular, $\text{ANCESTOR_BUILD}(\mathcal{G})$ contains

$$\left(\sum_{\mathcal{T} \in \mathcal{P}} \sum_{u \in \mathcal{I}(\mathcal{T})} \deg(u) \right)^2$$

edges, where $\deg(u)$

- Although deleting the elements in V_0 and their incident edges in $G^*(V)$ has the potential to create arc components C_1, C_2, \dots, C_k , deleting the elements in $V_0 \cap V$ in $G(V)$ will not create any blue components. This is because the sibling edges corresponding to the elements in V_0 are still coloured blue in the resulting subgraph of $G(V)$. However, Step 3'(a)(iii) colours these sibling edges red and it is this recolouring which reestablishes the correspondence described in Step 3'(b).
- The fact that the arc components of $G^*(V)$ correspond to the blue components of $G(V) \setminus (V_0 \cap V)$ as stated in Step 3'(b) is established in Lemma 12.
- Referring to Step 3'(c) of $E \text{ CEND } NT^*$, the set of red edges that are deleted is a union

(i)

Proof. Let e be the initial number of edges in (G) . As stated above, computing the blue components of (G) over all executions of Step 3'(b) necessitates (e)

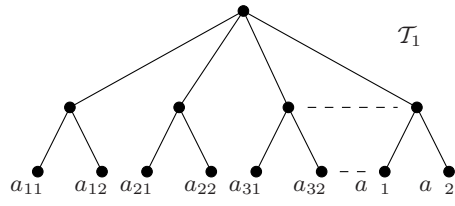
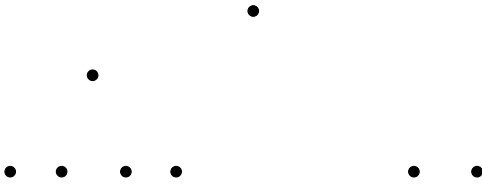
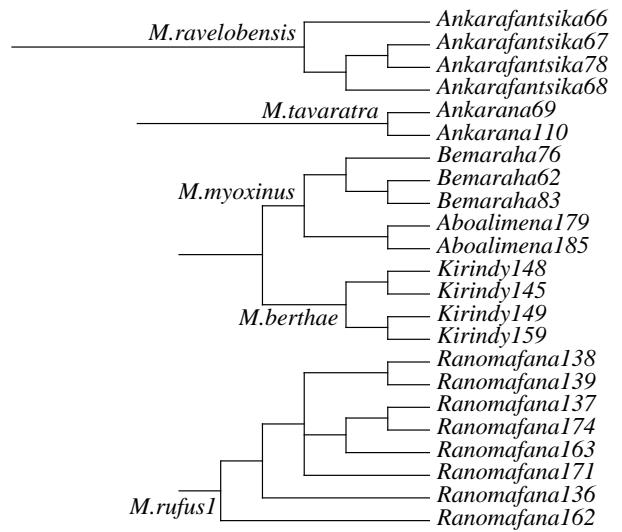
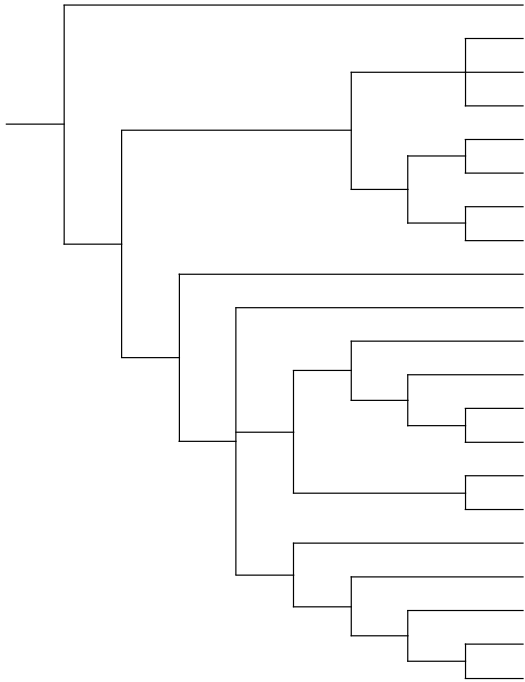


Figure 8: The rooted phylogenetic tree \mathcal{T}_1 .





inferred from craniodental morphological data (Masters and Brothers, 2002, Fig. 6.a). This tree resolves the first trifurcation mentioned above. As it is the strict consensus of the two most parsimonious trees, this tree also contains multifurcations. More precisely, the two observed trifurcations respectively concern the placement of two *Galagoides* species and of two *Galago* species.

The third and fourth source trees (c) and (d) have been inferred from mtDNA sequences combined from the control region homologous with the hypervariable region 1 in humans, COII and cytochrome b (Yoder et al., 2000, Fig. 2 and Fig. 3). The third tree (c) contains 18 species and subspecies of *Microcebus* and *Eulemur*. The fourth tree (d) contains 40 individuals of the *Microcebus* genus arranged in 9 identified species.

Internal labels of trees (b), (c), and (d) correspond to those displayed in the original figures, while those in tree (a) were added manually to demonstrate the ability of ANCESTRAL BINDER* to deal accurately with many labels, located at the same or different levels in the trees. The four detailed source trees are ancestrally compatible and Figure 11 shows the supertree resulting from the application of ANCESTRAL BINDER* to this collection. The obtained phylogeny is one of the largest produced for the strepsirrhines, spanning approximately 100 taxa on a number of taxonomic levels, from order to individuals. The source trees used in this example as well as the final supertree are available online from <http://www.systematicbiology.org>.

Discussion

ANCESTRAL BINDER* does not take primary data as input, but rather source trees inferred from this data with some level of confidence and through an adequate method. Thus, it is likely that the source trees considered for building a supertree will be more often compatible than, say, a set of primary character data. Nonetheless, it is likely that in many cases the source trees turn out to be incompati6(l)0.967972(e)3.56557(l)-n2647357(r0r0r0r0r0.6

Integration of ANCESTRAL BIPARTITION in a general supertree method

As stated in the introduction, consistency is an attractive property for any supertree method. Thus, in constructing a general supertree method, deciding compatibility is an integral part of the method. Currently, it seems that the only general supertree methods for rooted semi-labelled trees is given in Daniel and Sempel (2005). In this paper, the authors describe a general supertree method that allows for the possibility of variants. This method, called $NE_{TED}^{*}_{PE} T_{EE}$, extends ANCESTRAL BIPARTITION, and thus ANCESTRAL BIPARTITION*. If the source trees are compatible, then it outputs a supertree that ancestrally displays each of these trees. On the other hand, if the source trees are not compatible, then at some iteration there are no nodes that have indegree zero and no incident edges. By making an appropriate choice of nodes to delete, $NE_{TED}^{*}_{PE} T_{EE}$, or more particularly one of its variants, resolves this and continues on, eventually returning a supertree with several desirable features including the following:

- (i) ancestrally displaying every rooted binary semi-labelled trees that is ancestrally displayed by each of the source trees;
- (ii) independent of the order in which the source trees are listed.

We also remark that $NE_{TED}^{*}_{PE} T_{EE}$ runs in polynomial time and allows for the source trees to be weighted. Such weights, irrelevant for deciding compatibility (and thus ignored by ANCESTRAL BIPARTITION), can really help to arbitrate the conflicts between incompatible source trees.

The progress made in this paper on the running time of ANCESTRAL BIPARTITION improves the practicality of general supertree methods for nested taxa such as $NE_{TED}^{*}_{PE} T_{EE}$.

Repeated use of ANCESTRAL BIPARTITION in the production of a supertree

Despite the exactness ANCESTRAL BIPARTITION*, it can still be used to build a supertree from incompatible source trees. Two ways are highlighted below.

Finding a subset of the source trees that are compatible. Given an incompatible collection of source trees, finding a maximum-sized subset of trees in that are compatible is an NP-hard task (Bryant, 1997). However, heuristic methods can be easily implemented: (i) rank all trees in according to their size, or to some confidence value on the trees (es

already in \mathcal{T}' , which is checked by `ANCESTRAL_COMPATIBILITY*`. At the end of the process, \mathcal{T}' is a subset of compatible source trees, a supertree of which is provided by the final call to `ANCESTRAL_COMPATIBILITY*`.

Finding parts of the source trees that are compatible. Usually, source trees result from an extensive analysis of primary data and their clades are provided with associated confidence values, such as bootstrap values or bayesian posterior probabilities. As a first approximation, we may assume that these confidence values are representative in some sense of the correctness of the corresponding clades (see e.g. Berry and Gascuel (1996) for a discussion). Thus, when source trees are incompatible, a reasonable option is to first put into question the clades of the source trees that display the least support from the data. This suggests an intuitive and simple scheme to remove conflicts from the source trees by collapsing some of the clades from consideration: Let \mathcal{C} be the list of support values for clades of the source trees, sorted by increasing order of confidence. Note that a clade appearing in different trees with different confidence values can be accounted for by resorting to suitable weighting schemes. Collapse clades of the source trees whose support value is equal to the first value of \mathcal{C} and remove that value from the list. Then iterate until the

SIAM J. Comput. 10:405–421.

Masters, J. C. and D. J. Brothers. 2002. Lack of congruence between morphological and molecular data in reconstructing the phylogeny of the galagonoidae. *American Journal of Physical Anthropology* 117:79–93.

Page, R. D. M. 2002. Modified mincut supertrees. Pages 537–552 in *Second International Workshop on Algorithms in Bioinformatics* (R. Guig and D. Gusfield, eds.) Springer.

Page, R. D. M. 2004. Taxonomy, supertrees, and the tree of life. Pages 247–265 in *Phylogenetic supertrees: combining information to reveal the Tree of Life* (O. R. P. Bininda-

ancestor labels. Let $\mathcal{T}_1 \in \mathcal{L}$, and suppose that $\alpha, \beta \in (\mathcal{T}_1)_{\cap}$ such that α and β are not comparable in \mathcal{T}_1 . If $\gamma \in (\mathcal{T}_1)$ is an ancestor label of both α and β in \mathcal{T}_1 , then there is a path in (\mathcal{T}_1) from γ to α in which all nodes on this path are descendant labels of γ in \mathcal{T}_1 .

Proof. Without loss of generality, we may assume that γ labels the root of \mathcal{T}_1 . Furthermore, since for any element $\delta \in (\mathcal{T}_1)_{\cap}$, there is a path in (\mathcal{T}_1) from γ to any of its descendant labels in $(\mathcal{T}_1)_{\cap}$, we may also assume that $(\mathcal{T}_1)_{\cap}$ bijectively labels the leaves of \mathcal{T}_1 . This implies that \mathcal{T}_1 has no degree-two nodes.

We prove the lemma by showing that, for any pair of elements α and β in $(\mathcal{T}_1)_{\cap}$, there is a path joining this pair in (\mathcal{T}_1) with the property that all nodes on this path are in $(\mathcal{T}_1)_{\cap}$.

in 1)28236]TJ /R1111.959552Tf 8./R177.97011T

(I) After Step 3'(a) is completed, if V_1, V_2, \dots, V_k are the node sets of the arc components of G_i , then $V_1 \cap V_0, V_2 \cap V_0, \dots, V_k \cap V_0$ are the node sets of the blue components of G_i ($G_i \cap V_0$).

(II) Before Step 3'(c) is performed, an edge $e = (u, v)$ of G_i joins two arc components if and only if, for each sibling edge set S , each edge in this set is coloured red and joins two blue components in $G_i \cap V_0$ with the labels in S in one blue component and the labels in S' in the other blue component.

(III) After Step 3'(c) is completed, for

that α and β are non-comparable in \mathcal{T}'_1 . By considering the directions of the arcs in the path of directed edges between α and β , it is easily seen that one node on this path, say, γ , is an ancestor label of both α and β in \mathcal{T}'_1 . Since the node γ appears in $\mathcal{L}_i(\mathcal{O})$, it follows by the fact that $\mathcal{L}_i(\mathcal{O}) = \mathcal{L}_i(\mathcal{O}_1)$ that each of its descendant labels in $\mathcal{L}_i(\mathcal{O}_1)$ are in $\mathcal{L}_i(\mathcal{O} \cap \mathcal{O}_1)$. From Lemma 11, we now deduce that there is a path of blue edges in $\mathcal{L}_i(\mathcal{O} \cap \mathcal{O}_1)$ from γ to β , and so α and β are in the same blue component of $\mathcal{L}_i(\mathcal{O} \cap \mathcal{O}_1)$. This contradiction completes the proof of (I).

Appendix A Implementation Details

We give here a description of ANCE T B₀ D* that is more tuned towards implemen-

is the dynamic connectivity algorithm supporting deletion updates and connectivity queries.

```
foreach edge  $e = (i, j) \in L_{red}$  do
  perform a deletion update in  $\mathcal{L}$  for the edge  $e = (i, j)$ ;
  ifb
```

```

foreach component  $C_i$  in  $L_C^{next}$  do
   $E \text{ CEND NT}^*(C_i)$ ;
  if  $C_i$  returns no possible labelling then return no possible labelling
return the tree obtained by grafting all  $C_i$  as child subtrees of a root node labelled
by nodes in  $n_0 \cap C_i$ .

```

Note that, in the case where C_i is compatible, Step 4 issues a recursive call for each new component created by the on-going execution of $E \text{ CEND NT}^*$, as well as for the component n_0 . Indeed, this latter component still contains nodes. Note that some of these nodes that were not in n_0 at the beginning of the on-going call, can now be in this set because initial nodes of this set and some edges have been removed from this arc component.