Fast Computation of Supertrees for Compatible Phylogenies with Nested Taxa

Vincent Berry[1] and Charles Semple[2,3]

14 November 2005

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

[2]

B[illegible] 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 leaf-labelled trees as input and decides whether or not is compatible, in which case it returns a leaf-labelled supertree that displays . That is, the supertree preserves all of the relative groupings of taxa present in the source trees. For a collection of nested-taxa trees, if a supertree preserves all ancestral relationships as well as all groupings of taxa, then the supertree is said to ancestrally display this collection and the collection is said to be ancestrally compatible. These concepts are formally defined in the next section. However, we make a comment here on the way in which ancestor is used in this paper: by writing that a taxon is an ancestor of a taxon $'$, we mean that is either a hypothetical ancestral taxon of $'$ or that is the name of a taxonomic grouping containing $'$, so that node labelled is ancestral to the node labelled $'$. The algorithm Ance t B d takes a collection of nested-taxa trees as input and outputs a supertree that ancestrally displays if such a supertree exists, otherwise it states that the collection is not ancestrally compatible. Though designed to handle trees containing taxa at both internal nodes and leaves, Ance t B d also accepts collections of source trees that have taxa only at the leaves, because leaf-labelled trees are a special case of nested-taxa trees. In that particular case, Ance t B d decides the compatibility of the source trees in the usual sense. Consequently, it does indeed generalize B d.

Ance t B d has the desirable property to give an exact answer in polynomial-time (Daniel and Semple, 2004). However, there can be three objections to its use. First, for incompatible phylogenies to be combined, an all-or-nothing algorithm, that is just stating the incompatibility when it arises, is not desirable. Second, even for the easiest case of source trees that are all fully-resolved and have taxa only at the leaves, the running time of the version of Ance t B d stated in Daniel and Semple (2004) is (^{2} ^{3}), where is the number of source trees and is the number of taxa. Despite being polynomial, this running time makes t Ance (r)-0.647357(u)1.94718(s)0.806-477.06(a)-2.26269(r)-0.6.972056((r)-0.64735
269(r)-0.6.TJ 310.mbe b-4.85518(l)0.97394718(e)3.5635title b2(m)2.92434-0.648378(r)-0.647357(e)3.5635

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 Build (respectively, AncestralBuild) as a subroutine or to be a variant of Build (respectively, AncestralBuild). Indeed, this is already the case for some general methods: both MinCutSupertree (Semple and Steel, 2000) method and its modified version (Page, 2002) are variants of Build and, more recently, Daniel and Semple (2005) describe a class of general supertree methods for nested-taxa source trees that is a variant of AncestralBuild. (This class and more particularly the underlying general supertree method NestedSupertree 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 Build that runs in ($^{\frac{1}{2}}$) time, where is the total sum of the number of nodes in each of the source trees and (^{1}

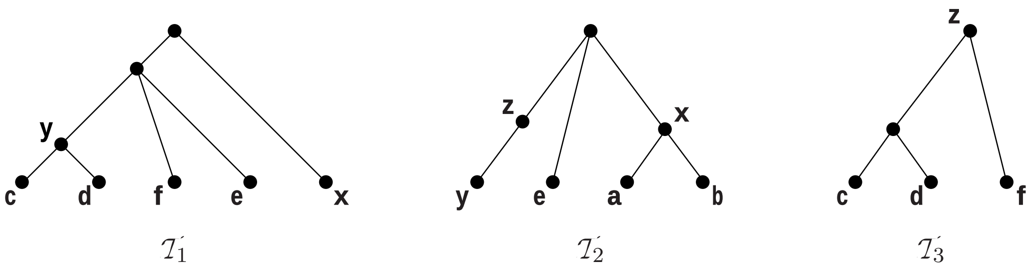


Figure 1: A compatible collection of rooted semi-labelled trees.

et al. (2005).

Preliminaries

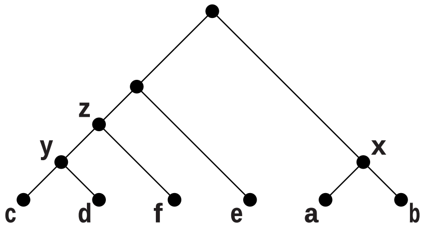
In this section, we describe some concepts that are frequently used in the paper. For further details, we refer the interested reader to Semple and Steel (2003).

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

1. We will often write a rooted semi-labelled -tree for a rooted semi-labelled tree on .

. 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 , 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


z
y
x
c
d
f
e
a
b

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

with the cluster (′) and successively breaking it down into disjoint subclusters. The way in which the clusters are broken up is decided by the descendancy 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 descendancy graph has no nodes which have indegree zero and no incident edges.

A gor ANCE T B D( )
np A collection of rooted semi-labelled trees on **X**.
O p A rooted semi-labelled tree $\mathcal{T}$ that ancestrally displays or the statement is not ancestrally compatible.

Construct a collection ′ of rooted fully-labelled trees from by adding distinct new labels to the unlabelled nodes in the trees of the collection.

Construct the descendancy graph D(′) of ′.

Call the subroutine E CEND NT(D(′)).

If E CEND NT **returns** *no possible labelling*, **then return** *is not ancestrally compatible*. **Otherwise, return the semi-labelled tree** $\mathcal{T}'$ **returned by** E CEND NT **with the added labels removed.**

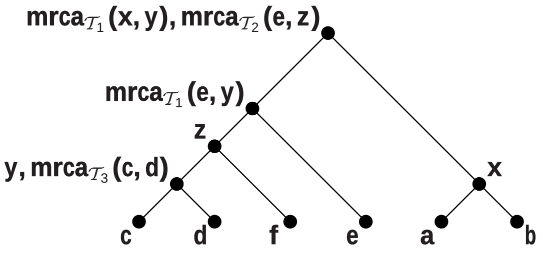


Figure 5: The rooted semi-labelled tree returned by E CEND NT as described in Example 2.

Remark.

1. With respect to descendancy, the added labels act as necessary "place holders" for unlabelled nodes.

. The recursive calls performed at Step 4 in E CEND NT 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 ANCE T B D 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 $'$ of rooted fully-labelled trees shown in Figure 3. Now Step 2 builds the descendancy graph ($'$) as shown in Figure 4 . On the first iteration of E CEND NT, .Tf 254.64(a)-2.26269(p)1.9672Tf 55.55989(e)R1111.95*1itra9.7846269(m)2.92434(p36340.

[illegible]

removes only one node in ( ), in which case the subroutine is executed ( ) times. The computation time is dominated by the cost of Steps 3(b) and 3(c) in the subroutine.

•

For the proof of (ii), suppose that Ance t B d (using the restricted descendancy graph) outputs a rooted semi-labelled tree $\mathcal{T}'$. We show that $\mathcal{T}'$ ancestrally displays . Let $\mathcal{T}_1'$ be an element of . By Lemma 2.1 (Bordewich et al., 2005), it suffices to show, for all ∈ $(\mathcal{T}_1')$ that (I) if is a descendant label of in $\mathcal{T}_1'$, then is a descendant label of in $\mathcal{T}'$, and (II) if and are non-comparable in $\mathcal{T}_1'$, then and 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 and are not comparable in $\mathcal{T}_1'$. Assume that $\mathcal{T}_1' = ({}_1; {}_1)$. Let be the node in ${}_1$ that is the most recent common ancestor of ${}_1( )$ and ${}_1( )$. By the construction of ${}^*( )$, there is a pair of children, and say, of the label labelling in ${}_1$ such that and are joined by an edge, and is an ancestor label of , and is an ancestor label of . Since we eventually output a tree, this edge is eventually deleted, but not until and , and hence and , are in separate arc components of some restriction of ${}^*( )$. It now follows that and are not comparable in $\mathcal{T}'$. □

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

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

edges, where (${}_i$

. Although deleting the elements in $_0$ and their incident edges in $^*(\; ')$ has the potential to create arc components $_1 \; _2 \; _k$, deleting the elements in $_0 \cap$ in $(\; ')$ will not create any blue components. This is because the sibling edges corresponding to the elements in $_0$ are still coloured blue in the resulting subgraph of $(\; ')$. However, Step 3′(a)(iii) colours these sibling edges red and it is this recolouring which reestablishes the correspondence described in Step 3′(b).

3 . The fact that the arc components of $^*(\;)$ correspond to the blue components of $(\; ') \; (\; _0 \cap \;)$ as stated in Step 3′(b) is established in Lemma 12.

. Referring to Step 3′(c) of E CEND NT*, the set of red edges that are deleted is a union

(i)

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

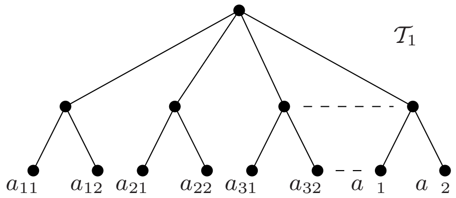
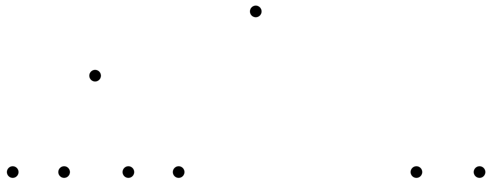
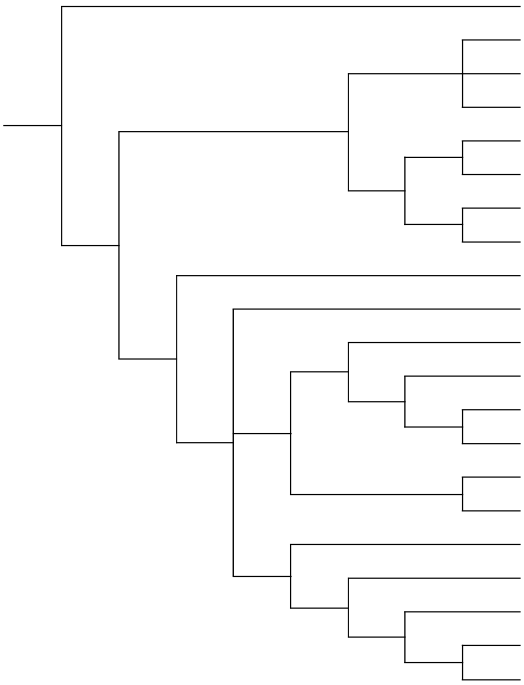
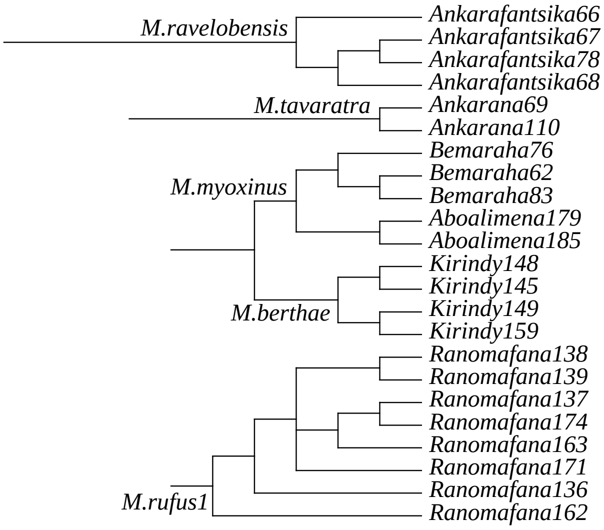


Figure 8: The rooted phylogenetic tree $\mathcal{T}_1'$.






M.ravelobensis
Ankarafantsika66
Ankarafantsika67
Ankarafantsika78
Ankarafantsika68
M.tavaratra
Ankarana69
Ankarana110
M.myoxinus
Bemaraha76
Bemaraha62
Bemaraha83
Aboalimena179
Aboalimena185
M.berthae
Kirindy148
Kirindy145
Kirindy149
Kirindy159
Ranomafana138
Ranomafana139
Ranomafana137
Ranomafana174
Ranomafana163
Ranomafana171
Ranomafana136
M.rufus1
Ranomafana162

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 AncestralBuild* 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 AncestralBuild* 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

AncestralBuild* 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(r0r0r0r0r0r0.64

Integration of AncestralBuild 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 Semple (2005). In this paper, the authors describe a general supertree method that allows for the possibility of variants. This method, called NestedSupertree, extends AncestralBuild, and thus AncestralBuild*. 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, NestedSupertree, 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 NestedSupertree runs in polynomial time and allows for the source trees to be weighted. Such weights, irrelevant for deciding compatiblity (and thus ignored by AncestralBuild), can really help to arbitrate the conflicts between incompatible source trees.

The progress made in this paper on the running time of AncestralBuild improves the practicality of general supertree methods for nested taxa such as NestedSupertree.

Repeated use of AncestralBuild in the production of a supertree

Despite the exactness AncestralBuild*, 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 $'$, which is checked by Ance t B d*. At the end of the process, $'$ is a subset of compatible source trees, a supertree of which is provided by the final call to Ance t B d*.

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

Proof. Without loss of generality, we may assume that labels the root of $\mathcal{T}_1'$. Furthermore, since for any element $\in$ $(\mathcal{T}_1') \cap$, there is a path in ($'$) from 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 and in $(\mathcal{T}_1') \cap$, there is a path joining this pair in ($'$) with the property that all nodes on this path are in $_1$)28236]TJ /R1111.959552Tf 8./R177.97011T

- **(I)** After Step 3′(a) is completed, if $_1$ $_2$ $_k$ are the node sets of the arc components of $_i$ $_0$, then $_1 \cap$ $_2 \cap$ $_k \cap$ are the node sets of the blue components of $_i$ ($_0 \cap$).

- **(II)** Before Step 3′(c) is performed, an edge $=$ $'$, of $_i$ $_0$ joins two arc components if and only if, for each sibling edge set of , each edge in this set is coloured red and joins two blue components in $_i$ ($_0 \cap$) with the labels in in one blue component and the labels in $'$ in the other blue component.

- **(III)** After Step 3′(c) is completed, for` that and are non-comparable in $\mathcal{T}_1'$. By considering the directions of the arcs in the path of directed edges between and , it is easily seen that one node on this path, say, is an ancestor label of both and in $\mathcal{T}_1'$. Since the node appears in $_i$ $_0$, it follows by the fact that ($_i$) $\cap$ = ($_i$) that each of its descendant labels in ($_1$) $\cap$ are in $_i$ ($_0$ $\cap$). From Lemma 11, we now deduce that there is a path of blue edges in $_i$ ($_0$ $\cap$) from to , and so and are in the same blue component of $_i$ ($_0$ $\cap$). This contradiction completes the proof of (I).

Appendix [illegible] Implementation Details

We give here a description of Ance t B[illegible] d* **that is more tuned towards implemen-**

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

```
foreach edge , $\in L_{red}$ do
    perform a deletion update in for the edge , ;
    ifb
```

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

Note that, in the case where $'$ is compatible, Step 4 issues a recursive call for each new component created by the on-going execution of E 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.