EXTENDING THE LIMITS OF SUPERTREE METHODS

MAGNUS BORDEWICH, GARETH EVANS, AND CHARLES SEMPLE

Abstract. Recently, two exact polynomial-time supertree methods have been developed in which the traditional input of rooted leaf-labelled trees has been extended in two separate ways. The first method, called RankedTree, allows for the inclusion of relative divergence dates and the second method, called AncestralBuild, allows for the inclusion of rooted trees in which some of the interior vertices as well as the leaves are labelled. The latter is particular useful for when one has information that includes nested taxa. In this paper, we present two supertree methods that unite and generalise RankedTree and AncestralBuild. The first method is polynomial time and combines the allowable inputs of RankedTree and AncestralBuild. It determines if the original input is compatible, in which case it outputs an appropriate 'ranked semi-labelled tree'. The second method lists all 'ranked semi-labelled trees' that are consistent with the original input. While there may be an exponential number of such trees, the second method outputs the next such tree in the list in polynomial time.

1. Introduction

Supertree methods are increasingly being used in evolutionary biology for the construction of evolutionary (phylogenetic) trees. These methods amalgamate a collection of smaller phylogenetic trees on overlapping sets of species into a single parent tree. Despite there increasing use, one of the valid criticisms of such methods is that the input is usually restricted to leaf-labelled trees, thus overlooking other relevant information. However, two new polynomial-time supertree methods, RankedTree [3] and AncestralBuild [7], have extended this usual input in two separate ways. In addition to leaf-labelled trees, the input to RankedTree allows for the inclusion of relative ancestral divergence dates and the input to AncestralBuild allows for the inclusion of trees in which some of the interior as well as all of their leaves are labelled. The former allows for information such as whether one divergence event happened before another divergence event to be included in the input, while the latter allows for nested taxa (for example, genera versus species) to be included in the input. Both algorithms were applied to real data sets in [5]. Both RankedTree and AncestralBuild are exact algorithms in that either they output a certain type of tree that is consistent with all the data in the input (the input is 'compatible') or they return a statement indicating that

Date: 14 June 2004.

The first author was supported by the New Zealand Institute of Mathematics and its Applications funded programme *Phylogenetic Genomics* and the work was conducted while at the University of Canterbury. The third author was supported by the New Zealand Marsden Fund (UOC310).

the input is inconsistent. While such algorithms are not entirely satisfactory (as biological data is often inconsistent), they are necessary steps for constructing more general supertree methods.

The purpose of this paper is twofold. Firstly, to present an exact polynomial-time algorithm (called BuildPlus) that combines the allowable inputs of RankedTree and AncestralBuild. Surprisingly, this is not as straightforward as one might first expect. If BuildPlus returns an appropriate tree, a natural and informative subsequent task is to find all such trees that are consistent with the input, so that one may compare and recognise common features. The second purpose of this paper is to present an algorithm (called AllBuildPlus) that lists all trees that are consistent with the original input with the property that *each* tree is outputted in polynomial time. In terms of complexity, this is the best possible in the sense that, as there may be an exponential number of such trees in the size of the input (see [9]), an algorithm that outputs every such list in polynomial time is not possible. Previously, this type of listing algorithm has only be done in the case the input consists of leaf-labelled trees [4, 8, 9]. The rest of this section provides necessary definitions. The notation and terminology in this paper follows [10].

The name BuildPlus is based on one of the first supertree methods. Originally designed for relational databases, this method, called Build, is an exact algorithm that takes as its input a collection of 'rooted phylogenetic trees' and determines if this collection is 'compatible' (defined below), in which case it outputs a rooted phylogenetic tree that 'displays' each tree in this collection [1]. The algorithm BuildPlus extends Build by allowing collections of 'rooted semi-labelled trees' and 'relative divergence dates' to be included in the input. Moreover, like Build, the algorithm BuildPlus determines if these collections are 'compatible', in which case it outputs a 'ranked semi-labelled tree' that 'ancestrally displays' each of the trees and 'preserves' each of the relative divergence dates in these collections. We describe these notions next.

A *rooted semi-labelled tree $\mathcal{T}$ (on X)* is an ordered pair $(T;\phi)$ consisting of a rooted tree T with vertex set V and root ρ, and a map $\phi : X \to V$ with the following properties:

(i) for all $v \in V - \{\rho\}$ of degree at most two, $v \in \phi(X)$; and
(ii) if ρ has degree zero or one, then $\rho \in \phi(X)$.

Rooted semi-labelled trees on X are also called *rooted X-trees*. The set X is called the *label set* of $\mathcal{T}$ and is denoted by $\mathcal{L}(\mathcal{T})$. The elements of X are the *labels* of $\mathcal{T}$. Ignoring the numerals, Fig. 1 shows a rooted semi-labelled tree. A rooted semi-labelled tree is *fully labelled* if $\phi^{-1}(v)$ is non-empty for all $v \in V$. A *rooted phylogenetic X-tree* is a rooted semi-labelled tree on X in which ρ has degree at least two and ϕ is a bijection from X into the set of leaves of T. Two rooted semi-labelled X-trees $\mathcal{T}_1 = (T_1, \phi_1)$ and $\mathcal{T}_2 = (T_2, \phi_2)$, where $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$, are *isomorphic* if there exists a bijection $\psi : V_1 \to V_2$ which induces a bijection between E_1 and E_2, and satisfies $\phi_2 = \psi \circ \phi_1$. Intuitively, a rooted semi-labelled tree is simply a rooted tree in which the leaves as well as some of the interior vertices are labelled. In evolutionary biology, an interior label represents a taxa at

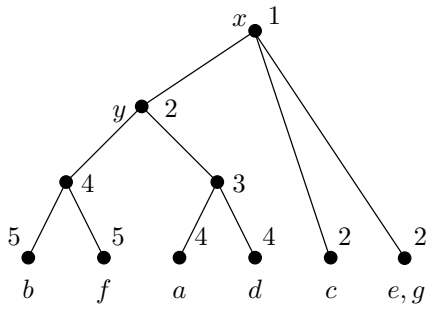


FIGURE 1. A ranked semi-labelled tree on $\{a, b, c, d, e, f, g, x, y\}$.

a taxonomic level higher than that of its descendants. For example, families and genera are higher taxonomic levels than genera and species, respectively. Observe that this type of input in BUILDPLUS allows for the possibility of a leaf of one of the input trees to represent a taxon that is represented by an interior label of another tree.

For taxon $w, x, y, z \in X$, a *relative divergence date* is a statement of the form $\mathrm{div}(w, x) < \mathrm{div}(y, z)$ which is interpreted as "the divergence of w and x predates the divergence of y and z". The *label set* of $\mathrm{div}(w, x) < \mathrm{div}(y, z)$ is $\{w, x, y, z\}$. This type of input could be based, for example, on fossil data or molecular dating techniques.

A useful partial order $\leq_T$ on the vertex set V of T is obtained by setting $u \leq_T v$ if the path from the root of T to v includes u. Observe that the partial order $\leq_T$ has the property that, for every pair of elements, the greatest lower bound exists. For $u, v \in V$, we call this lower bound the *most recent common ancestor* of u and v and denote it by $\mathrm{mrca}_T(u, v)$. Extending this partial order to the label set X of a rooted semi-labelled tree $\mathcal{T} = (T; \phi)$, we write $a \leq_{\mathcal{T}} d$ if $\phi(a) \leq_T \phi(d)$, in which case, d is a *descendant* of a. If, for $x, y \in X$, x is not a descendant of y and y is not a descendant of x, then we say that x and y are *not comparable*. Furthermore, for all $x, y \in X$, we let

$$\mathrm{mrca}_{\mathcal{T}}(x, y) = \phi^{-1}(\mathrm{mrca}_T(\phi(x), \phi(y))),$$

where ϕ^{-1} is taken to be the generalised inverse function of ϕ on sets.

Let $\mathcal{T} = (T; \phi)$ be a rooted semi-labelled tree on X, and let r be a map from the set V of vertices of T into the set of non-negative integers such that if $u, v \in V$ and $u <_T v$, then $r(u) < r(v)$. The pair $(\mathcal{T}, r)$ is called a *ranked semi-labelled tree* and the map r is called a *rank function* for $\mathcal{T}$. A ranked semi-labelled tree is shown in Fig. 1. Now let $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ be two ranked semi-labelled trees on X, and suppose that $\mathcal{T}_1$ is isomorphic to $\mathcal{T}_2$ under ψ. Then $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ are *ranked isomorphic* if, for all vertices $u, v \in \mathcal{T}_1$, the following properties are satisfied:

(i) if $r_1(u) = r_1(v)$, then $r_2(\psi(u)) = r_2(\psi(v))$; and
(ii) if $r_1(u) < r_1(v)$, then $r_2(\psi(u)) < r_2(\psi(v))$.

For nested taxa, the analogue of 'displays' (in the case of BUILD) is 'ancestrally displays'. Let $\mathcal{T} = (T; \phi)$ be a rooted X-tree. For a subset X' of X, the *restriction* of $\mathcal{T}$ to X', denoted $\mathcal{T}|X'$, is the rooted X'-tree that is obtained from the minimal rooted subtree of $\mathcal{T}$ that connects the elements in $\phi(X')$ by suppressing all vertices of degree two that are not in $\phi(X')$. Now let $\mathcal{T}'$ be a rooted semi-labelled tree on X'. Then $\mathcal{T}$ *ancestrally displays* $\mathcal{T}'$ if $\mathcal{T}'$ may be obtained from $\mathcal{T}|X'$ by contracting edges (that is $\mathcal{T}|X'$ *refines* $\mathcal{T}'$, see [10]), and for all $x, y \in X'$, the following hold:

(i) if $x <_{\mathcal{T}'} y$, then $x <_{\mathcal{T}} y$; and
(ii) if x is not comparable to y in $\mathcal{T}'$ under $\leq_{\mathcal{T}'}$, then x is not comparable to y in $\mathcal{T}$ under $\leq_{\mathcal{T}}$.

Note that if $\mathcal{T}'$ is a rooted phylogenetic tree, then ancestrally displays is equivalent to the notion of 'displays' mentioned above. A collection $\mathcal{P}$ of rooted semi-labelled trees is *ancestrally displayed* by $\mathcal{T}$ if each tree in $\mathcal{P}$ is ancestrally displayed by $\mathcal{T}$.

Let $\mathrm{div}(w, x) < \mathrm{div}(y, z)$ be a relative divergence date. A ranked semi-labelled tree $(\mathcal{T}, r)$ on X *preserves* $\mathrm{div}(w, x) < \mathrm{div}(y, z)$ if $w, x, y, z \in X$ and $r(\mathrm{mrca}(w, x)) < r(\mathrm{mrca}(y, z))$. A collection $\mathcal{D}$ of relative divergence dates is *preserved* by $(\mathcal{T}, r)$ if each member of $\mathcal{D}$ is preserved by $(\mathcal{T}, r)$. For a collection $\mathcal{P}$ of rooted semi-labelled trees and a collection $\mathcal{D}$ of relative divergence dates, we say that $\mathcal{P}$ and $\mathcal{D}$ are *compatible* if there is a ranked semi-labelled tree $(\mathcal{T}, r)$ that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$.

In Section 2, we present BUILDPLUS and show that it provides a polynomial-time solution to the following problem:

Problem: TREE COMPATIBILITY
Instance: A collection $\mathcal{P}$ of rooted semi-labelled tree and a collection $\mathcal{D}$ of relative divergence dates.
Question: Does there exist a ranked semi-labelled tree that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$ and, if so, can we construct such a ranked semi-labelled tree?

If a collection $\mathcal{P}$ of rooted semi-labelled trees and a collection $\mathcal{D}$ of relative divergence dates are compatible, the next step forward is to list all ranked semi-labelled trees that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. In Section 3, we present ALLBUILDPLUS, a supertree method that takes $\mathcal{P}$ and $\mathcal{D}$ as its input, and returns all such ranked semi-labelled trees. Since the number of such trees may be exponential in the input size, this cannot be achieved in polynomial time; ALLBUILDPLUS runs in time polynomial in the output size. We show, in Section 4, that ALLBUILDPLUS can be easily adjusted to have the desirable property that the time between outputting one tree in the list and the next, is polynomial in the input size. We remark here that counting the number of rank semi-labelled trees that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$ is likely to be computationally hard as it is shown in [2] that, in general, counting the number of rooted 'binary' phylogenetic trees that displays a given collection of rooted binary phylogenetic trees is #P-complete.

The rest of this section contains some additional preliminaries that are needed throughout the paper. Firstly, for collections $\mathcal{P}$ of rooted semi-labelled trees and

$\mathcal{D}$ of relative divergence dates, we let $\mathcal{L}(\mathcal{P})$ and $\mathcal{L}(\mathcal{D})$ denote the union of the label sets of members of $\mathcal{P}$ and $\mathcal{D}$, respectively.

Let $\mathcal{T} = (T; \phi)$ be a rooted X-tree. For a vertex u of T, let

$$C(u) = \bigcup_{v: u \leq_T v} \phi^{-1}(v).$$

The set $C(u)$ is called a *cluster* of $\mathcal{T}$. It follows by the definition of a rooted X-tree that, for all distinct vertices u and v of T, $C(u) \neq C(v)$. The collection of clusters of $\mathcal{T}$ is denoted by $\mathcal{H}(\mathcal{T})$. It is known that $\mathcal{T}$ is completely determined by $\mathcal{H}(\mathcal{T})$ in the sense that if $\mathcal{H}(\mathcal{T}) = \mathcal{H}(\mathcal{T}')$, then $\mathcal{T}$ is isomorphic to $\mathcal{T}'$ (see [10, Theorem 3.5.2]). Furthermore, $\mathcal{T}$ can be efficiently constructed from $\mathcal{H}(\mathcal{T})$. Note that if u is a strict descendant of v then $C(u) \subset C(v)$.

The set of clusters of a rooted X-tree induces a special type of set system called a 'hierarchy'. In this paper we will call a collection $\mathcal{H}$ of non-empty subsets of X is a *hierarchy (on X)* if, for all $A, B \in \mathcal{H}$, we have $A \cap B \in \{\emptyset, A, B\}$. If a hierarchy $\mathcal{H}$ on X contains X, then, up to isomorphism, there is a unique rooted X-tree $\mathcal{T}$ whose set of clusters is $\mathcal{H}$. Furthermore, analogous to the way in which a rank function is associated to the vertices of a rooted semi-labelled tree, we can associate a rank function to a hierarchy on X. Let $\mathcal{H}$ be a hierarchy on X and let r be a map from the elements of $\mathcal{H}$ (under set inclusion) into the set of non-negative integers such that if A and B are two members of $\mathcal{H}$ and B is a proper subset of A, then $r(A) < r(B)$. If $\mathcal{H}$ contains X, we call $(\mathcal{H}, r)$ a *ranked hierarchy* on X and r a *rank function* for $\mathcal{H}$. It follows by the remarks above that ranked hierarchies on X can be viewed as ranked semi-labelled trees on X. This viewpoint will be freely used throughout the remainder of the paper.

Lastly, for a ranked hierarchy $(\mathcal{H}, r)$ on X and a subset X' of X, the restriction of $(\mathcal{H}, r)$ to X', denoted $(\mathcal{H}, r)|X'$, is the ranked hierarchy $(\mathcal{H}', r')$, where

$$\mathcal{H}' = \{A \cap X' : A \in \mathcal{H}, A \cap X' \neq \emptyset\}$$

and, for all $A' \in \mathcal{H}'$, set

$$r'(A') = \max\{r(A) : A' = A \cap X' \text{ and } A \in \mathcal{H}\}.$$

2. BuildPlus

In this section, we present BuildPlus and show that it provides a solution to Tree Compatibility. We begin by defining a construction for fully labelling the input and, secondly, a graph that is central to operation of BuildPlus. This graph contains arcs as well as two types of edges, and extends the analogous graph that is used in AncestralBuild (see [7]).

Let $\mathcal{T} = (T; \phi)$ be a rooted semi-labelled tree on X, where T has vertex set V. A rooted fully-labelled tree $\mathcal{T}_1 = (T; \phi_1)$ on X_1, where $X \subseteq X_1$, is obtained from $\mathcal{T}$ by *adding distinct new labels* if, for all distinct $u, v \in V$, the following properties are satisfied:

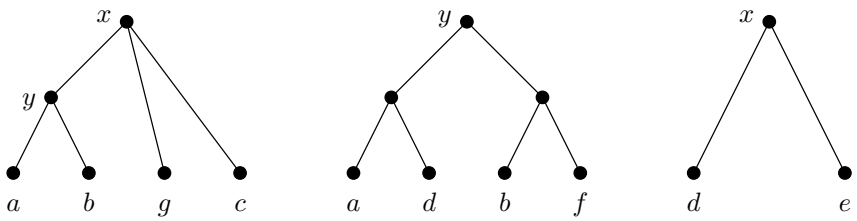


FIGURE 2. A collection $\mathcal{P}$ of rooted semi-labelled trees.

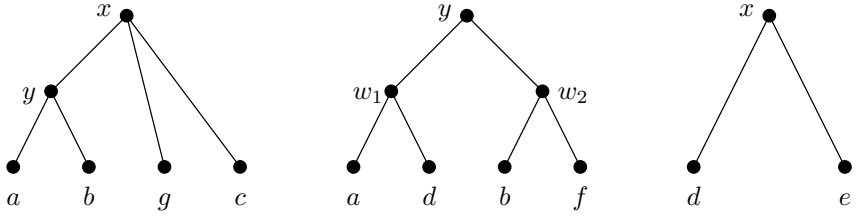


FIGURE 3. A collection $\mathcal{P}'$ of rooted fully-labelled trees.

(i) If $\phi^{-1}(v)$ is non-empty, then $\phi_1^{-1}(v) = \phi^{-1}(v)$.
(ii) If $\phi^{-1}(v)$ is empty, then $|\phi_1^{-1}(v)| = 1$.
(iii) If $\phi^{-1}(v)$ is empty, then $\phi_1^{-1}(v) \notin \phi_1^{-1}(V - \{v\})$.

Loosely speaking, $\mathcal{T}_1$ is obtained from $\mathcal{T}$ by labelling all non-labelled vertices of $\mathcal{T}$ with exactly one label such that all new labels are distinct. Extending this notion to a collection $\mathcal{P}$ of rooted semi-labelled trees, we say that $\mathcal{P}_1$ has been obtained from $\mathcal{P}$ by *adding distinct new labels* if it has been obtained by adding distinct new labels to each tree in $\mathcal{P}$ so that, for any pair of trees, no two new labels are the same. For example, the collection $\mathcal{P}'$ of rooted fully-labelled trees in Fig. 3 has been obtained from the collection $\mathcal{P}$ of rooted semi-labelled tress in Fig. 2 by adding distinct new labels (in particular, w_1 and w_2).

Now let $\mathcal{P}$ be a collection of rooted fully-labelled trees and let $\mathcal{D}$ be a collection of relative divergence constraints. The *constraint graph* of $\mathcal{P}$ and $\mathcal{D}$, denoted $G(\mathcal{P}, \mathcal{D})$, is the multi-graph that has vertex set $\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})$, and arc and (partially-labelled) edge sets that are defined as follows. The *blue arc set* of $G(\mathcal{P}, \mathcal{D})$ is

$$\{(x, y) : \text{there exists } \mathcal{T} \in \mathcal{P} \text{ with } x, y \in \mathcal{L}(\mathcal{T}) \text{ and } x <_{\mathcal{T}} y\},$$

the *red edge set* of $G(\mathcal{P}, \mathcal{D})$ is

$$\{\{x, y\} : \text{there exists } \mathcal{T} \in \mathcal{P} \text{ with } x, y \in \mathcal{L}(\mathcal{T}) \text{ and } x \text{ is not comparable to } y \text{ under } \leq_{\mathcal{T}}\},$$

and the *blue edge set* of $G(\mathcal{P}, \mathcal{D})$ is

$$\{\{w, x\} \text{ labelled } yz : \mathrm{div}(y, z) < \mathrm{div}(w, x) \in \mathcal{D}\}.$$

Note that a pair of vertices may be joined by more than one blue edge, each corresponding to a different divergence constraint and therefore having a different label. Now let G be a subgraph of $G(\mathcal{P}, \mathcal{D})$. With regards to G, the *in-degree* of a vertex x is the number of arcs directed into x (edges are ignored). A *blue component* is

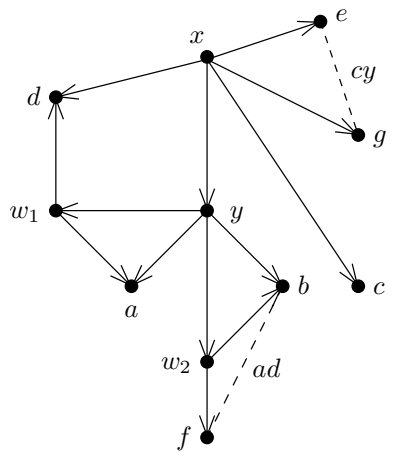


FIGURE 4. An example of a constraint graph. Only blue arcs representing direct descendants are shown, the blue edges are dashed with labels, the red edges are omitted.

a maximal connected subgraph of the graph obtained from G by deleting all the red edges, and considering both the blue edges and blue arcs as undirected edges. Furthermore, $S(G)$ is the set of vertices of G that have in-degree zero and no incident edges, $E_r(G)$ is the set of red edges of G whose end-vertices are in distinct blue components of G, and $E_b(G)$ is the set of blue edges of G whose labels, yz say, have the property that either y and z are in distinct blue components, or y or z is not a vertex of G. To illustrate the constraint graph, let $\mathcal{P}'$ be the collection of rooted fully-labelled trees shown in Fig. 3 and let

$$\mathcal{D} = \{\mathrm{div}(c, y) < \mathrm{div}(e, g), \mathrm{div}(a, d) < \mathrm{div}(b, f)\}. \tag{1}$$

The constraint graph $G(\mathcal{P}', \mathcal{D})$ is shown in Fig. 4, where for simplicity only blue arcs representing direct descendants are shown, the blue edges are dashed with labels, and the red edges are omitted. This example will be extended latter to illustrate the running of BUILDPLUS.

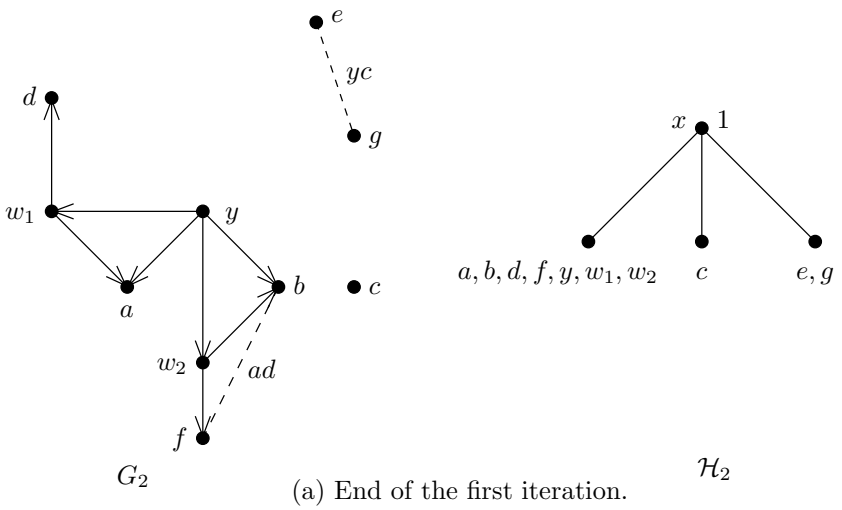
For the reader familiar with BUILD, the algorithm BUILDPLUS works in a similar way. In particular, the algorithm attempts to construct a ranked hierarchy on X (equivalently, a ranked semi-labelled tree on X) that ancestrally displays a collection $\mathcal{P}$ of rooted semi-labelled trees and preserves a collection $\mathcal{D}$ of relative divergence dates beginning with the root and working towards the leaves. Each iteration either refines the current ranked semi-labelled tree on X or determines that $\mathcal{P}$ and $\mathcal{D}$ are not compatible. The basis of this decision is an associated graph which is always a subgraph of $G(\mathcal{P}', \mathcal{D})$, where $\mathcal{P}'$ is a collection of rooted semi-labelled trees that has been obtained from $\mathcal{P}$ by adding distinct new labels. Because the input to BUILDPLUS also includes relative divergence dates, an important difference between BUILD and BUILDPLUS is that, as in RANKEDTREE, at each iteration one must consider all minimal clusters of the currently constructed tree and not just any one of them.

We now present BuildPlus.

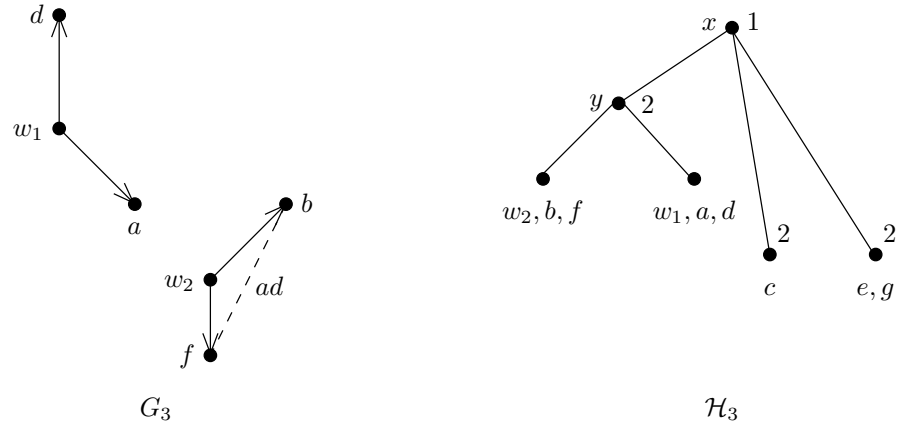
Algorithm: BuildPlus($\mathcal{P}, \mathcal{D}$)
Input: A collection $\mathcal{P}$ of rooted semi-labelled trees and a collection $\mathcal{D}$ of relative divergence constraints.
Output: A ranked hierarchy $(\mathcal{H}, r)$ on $\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})$ that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$, or the statement $\mathcal{P}$ *and* $\mathcal{D}$ *are not compatible.*

begin
Construct a collection $\mathcal{P}'$ of rooted fully-labelled trees from $\mathcal{P}$ by adding distinct new labels.
$k \leftarrow 1$
$G_1 \leftarrow G(\mathcal{P}', \mathcal{D})$
Let π_1 be the partition of the vertex set of G_1 induced by the blue components of G_1.
if $|\pi_1| = 1$, **then** // add root vertex (cluster)
$\mathcal{H}_1 \leftarrow \{\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})\}$
else // add root vertex plus its children
$\mathcal{H}_1 \leftarrow \{\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})\} \cup \pi_1$
$r'(\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})) \leftarrow 0$
end (if-else)
repeat while G_k non-empty
Let G'_k be the graph obtained from G_k by deleting the edges in $E_r(G_k)$ and $E_b(G_k)$.
if $S(G'_k)$ is empty, **then**
return $\mathcal{P}$ *and* $\mathcal{D}$ *are not compatible* and **halt**
else
Let G_{k+1} be the graph obtained from G'_k by deleting the vertices in $S(G'_k)$.
Let π_{k+1} be the partition of the vertex set of G_{k+1} induced by the blue components of G_{k+1}.
$\mathcal{H}_{k+1} \leftarrow \mathcal{H}_k \cup \pi_{k+1}$
for each subset B in π_k which is not in π_{k+1} **do**
$r'(B) \leftarrow k$ // assign rank to finished vertices
end (for)
$k \leftarrow k + 1$
end (if-else)
end (repeat)
$\mathcal{H}' \leftarrow \mathcal{H}_k$
$(\mathcal{H}, r) \leftarrow (\mathcal{H}', r')|\,(\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D}))$
return $(\mathcal{H}, r)$
end.

As an example of the running of BuildPlus, recall the collection $\mathcal{P}$ of rooted semi-labelled trees shown in Fig. 2 and the collection of relative divergence dates (1). Using the collection $\mathcal{P}'$ of fully-labelled trees shown in Fig. 3, G_1 is shown in Fig. 4 with the red edges missing. Since G_1 is a single blue component, $\pi_1 = \mathcal{H}_1 = \big\{\{a, b, c, d, e, f, g, x, y, w_1, w_2\}\big\}$. In the first iteration, $E_r(G_1) = \emptyset$, $E_b(G_1) = \emptyset$, and $S(G'_1) = x$. Once x has been deleted, G_2 is as shown in Fig. 5, where again only blue arcs representing direct descendants are shown, the blue edges are dashed with labels, and the red edges are omitted. Hence $\pi_2 = \{\{a, b, d, f, y, w_1, w_2\}, \{c\}, \{e, g\}\}$,



(a) End of the first iteration.



(b) End of the second iteration.

FIGURE 5. Two iterations of BUILDPLUS. Only blue arcs representing direct descendants are shown, the blue edges are dashed with labels, the red edges are omitted.

and $\mathcal{H}_2$ is as shown in Fig. 5. Writing an edge $\{p, q\}$ as pq, we have that

$$E_r(G_2) = \{ca, cb, cg, cy, ed, ag, yg, bg\}$$

and

$$E_b(G_2) = \{eg\},$$

so $S(G_2') = \{y, c, e, g\}$ and $\pi_3 = \big\{\{w_2, b, f\}, \{w_1, a, d\}\big\}$. After deleting the vertices in $S(G_2')$, we get G_3 and $\mathcal{H}_3$ as shown in Fig. 5.

Now

$$E_r(G_3) = \{w_1w_2, w_1b, w_1f, w_2a, w_2d, ab, af, bd, df\}$$

and

$$E_b(G_3) = \emptyset.$$

In this iteration, $S(G_3') = \{w_1, w_2\}$ and G_4 has exactly three edges; a blue edge joining b and f labelled ad, a red edge joining a and d, and a red edge joining b and f. Thus $\pi_4 = \big\{\{a\}, \{d\}, \{b, f\}\big\}$, and the clusters $\{w_1, a, d\}$ and $\{w_2, b, f\}$ are ranked in this iteration. Cluster $\{b, f\}$ is ranked in the next iteration, since a and d are now in separate blue components. This process continues for one further iteration. It follows that $\mathcal{P}$ and $\mathcal{D}$ are compatible, and the eventual ranked semi-labelled tree that is returned by BuildPlus is shown in Fig. 1.

We now show that BuildPlus provides a polynomial-time solution to Tree Compatibility. We begin with two lemmas. For a rooted semi-labelled tree $\mathcal{T}$, let

$$D(\mathcal{T}) = \{(a, d) : a, d \in \mathcal{L}(\mathcal{T}) \text{ and } a <_{\mathcal{T}} d\}$$

and

$$N(\mathcal{T}) = \{\{x, y\} : x, y \in \mathcal{L}(\mathcal{T}) \text{ and } x \text{ is not comparable to } y \text{ under } \leq_{\mathcal{T}}\}.$$

Lemma 2.1. *Let $\mathcal{T}'$ be a rooted fully-labelled tree on X' and let $\mathcal{T}$ be a rooted semi-labelled tree on X, where $X' \subseteq X$. Then $\mathcal{T}$ ancestrally displays $\mathcal{T}'$ if and only if $D(\mathcal{T}') \subseteq D(\mathcal{T})$ and $N(\mathcal{T}') \subseteq N(\mathcal{T})$.*

Proof. By definition, it immediately follows that if $\mathcal{T}$ ancestrally displays $\mathcal{T}'$, then $D(\mathcal{T}') \subseteq D(\mathcal{T})$ and $N(\mathcal{T}') \subseteq N(\mathcal{T})$. Therefore, suppose that $D(\mathcal{T}') \subseteq D(\mathcal{T})$ and $N(\mathcal{T}') \subseteq N(\mathcal{T})$. Consider the two rooted semi-labelled trees shown in Fig. 6. For convenience, we call these trees Type (I) and (II) as shown. To prove the converse, it suffices to show by [7, Proposition 4.3] that if $a, b, c \in \mathcal{L}(\mathcal{T}')$ and $\mathcal{T}'|\{a, b, c\}$ is isomorphic to a Type (I) or Type (II) tree, then $\mathcal{T}|\{a, b, c\}$ is isomorphic to a Type (I) or Type (II) tree, respectively.

Suppose that $\mathcal{T}'|\{a, b, c\}$ is isomorphic to Type (I) tree. Since $\mathcal{T}'$ is fully-labelled, there exists labels $y, z \in \mathcal{L}(\mathcal{T}')$ such that $y \in \mathrm{mrca}_{\mathcal{T}'}(a, b)$ and $z \in \mathrm{mrca}_{\mathcal{T}'}(a, c)$. Then $(z, y), (y, a), (y, b), (z, c) \in D(\mathcal{T}')$, which implies that $(z, y), (y, a), (y, b), (z, c) \in D(\mathcal{T})$. Furthermore, $\{a, b\}, \{y, c\} \in N(\mathcal{T}')$ and so $\{a, b\}, \{y, c\} \in N(\mathcal{T})$. It is now easily seen that these four 2-tuples in $D(\mathcal{T})$ and two 2-element sets in $N(\mathcal{T})$ means that $\mathcal{T}|\{a, b, c, y, z\}$ ancestrally displays $\mathcal{T}'|\{a, b, c, y, z\}$. It now follows that $\mathcal{T}|\{a, b, c\}$ is isomorphic to a Type (I) tree. A similar argument shows that if $\mathcal{T}'|\{a, b, c\}$ is isomorphic to Type (II) tree as shown in Fig. 6, then $\mathcal{T}|\{a, b, c\}$ is isomorphic to a Type (II) tree as well. This completes the proof of the lemma. □

The next lemma is an immediate consequence of [6, Lemma 4.2], which is essentially the same statement, but without reference to $\mathcal{D}$.

Lemma 2.2. *Let $\mathcal{P}$ be a collection of rooted semi-labelled trees and let $\mathcal{D}$ be a collection of relative divergence constraints. Let $\mathcal{P}'$ be a collection of rooted fully-labelled trees obtained from $\mathcal{P}$ by adding distinct new labels. Then $\mathcal{P}$ and $\mathcal{D}$ are compatible if and only if $\mathcal{P}'$ and $\mathcal{D}$ are compatible. Moreover, if $(\mathcal{T}, r)$ is a ranked rooted semi-labelled tree that ancestrally displays $\mathcal{P}'$ and preserves $\mathcal{D}$, then $(\mathcal{T}, r)|(\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D}))$ ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$.*

FIGURE 6. Two rooted semi-labelled trees.

Theorem 2.3. *Let $\mathcal{P}$ be a collection of rooted semi-labelled trees and let $\mathcal{D}$ be a collection of relative divergence constraints.*

(i) *If* BUILDPLUS *applied to $\mathcal{P}$ and $\mathcal{D}$ returns a ranked hierarchy, then this ranked hierarchy ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$.*
(ii) *If* BUILDPLUS *applied to $\mathcal{P}$ and $\mathcal{D}$ returns the statement* $\mathcal{P}$ and $\mathcal{D}$ are not compatible*, then there is no ranked hierarchy that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$.*

Proof. By Lemma 2.2, it suffices to prove the theorem for when $\mathcal{P}$ is a collection of rooted fully-labelled trees. To establish (i), suppose that BUILDPLUS applied to $\mathcal{P}$ and $\mathcal{D}$ returns a ranked hierarchy $(\mathcal{H}, r)$ on $\mathcal{L}(P) \cup \mathcal{L}(D)$. Let $(\mathcal{T}, r)$ be the ranked semi-labelled tree whose ranked hierarchy is $(\mathcal{H}, r)$. We first show that $\mathcal{T}$ ancestrally displays $\mathcal{P}$ using Lemma 2.1. Let $\mathcal{T}_1$ be an element of $\mathcal{P}$, and let a and b be elements of $\mathcal{L}(\mathcal{T}_1)$. Assume that $b <_{\mathcal{T}_1} a$. Since BUILDPLUS returns $(\mathcal{H}, r)$ and (b, a) is an arc of $G(\mathcal{P}, \mathcal{D})$, there must be some iteration k in which b and a are in the same blue component of G_k, and $b \in S(G'_k)$. This implies that $b <_{\mathcal{T}} a$. It now follows that $D(\mathcal{T}_1) \subseteq D(\mathcal{T})$.

Now assume that a and b are not comparable in $\mathcal{T}_1$. Then a and b are connected by a red edge in $G(\mathcal{P}, \mathcal{D})$. Since BUILDPLUS returns $(\mathcal{H}, r)$, this edge is eventually deleted, but not until a and b are in separate blue components of some subgraph of $G(\mathcal{P}, \mathcal{D})$. This implies that in $\mathcal{T}$ there is a cluster in which a is an element but b is not, and there is a cluster in which b is an element but a is not. In particular, a and b are not comparable in $\mathcal{T}$, and so $N(\mathcal{T}_1) \subseteq N(\mathcal{T})$. It now follows by Lemma 2.1 that $\mathcal{T}$ ancestrally displays $\mathcal{T}_1$ and, more particularly, $\mathcal{T}$ ancestrally displays $\mathcal{P}$.

To complete the proof of (i), we now show that $(\mathcal{T}, r)$ preserves $\mathcal{D}$. Assume that $\mathrm{div}(c, d) < \mathrm{div}(a, b)$ is an element of $\mathcal{D}$. Then, in $G(\mathcal{P}, \mathcal{D})$, there is a blue edge connecting a and b which is labelled cd. Because BUILDPLUS returns $(\mathcal{H}, r)$, this edge is eventually deleted. Consider the iteration, k say, in which this blue edge is deleted. Then, in the $(k-1)$-th iteration, a and b are in the same blue component of G_{k-1}, and either c and d are in separate blue components of G_{k-1}, or c or d is not a vertex of G_{k-1}. In either case, it follows from the description of BUILDPLUS that, in $(\mathcal{T}, r)$, the most recent common ancestor of c and d has rank $k-1$, while the

most recent common ancestor of a and b has rank at least k. Thus $(\mathcal{T}, r)$ preserves $\mathrm{div}(c, d) < \mathrm{div}(a, b)$ and, more particularly, preserves $\mathcal{D}$.

We now prove (ii). In the proof that follows, we shall view the clusters of $\mathcal{T}$ as the vertices of $\mathcal{T}$. Suppose that there is a ranked semi-labelled tree $(\mathcal{T}, r)$ that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$, but suppose that BuildPlus applied to $\mathcal{P}$ and $\mathcal{D}$ returns *$\mathcal{P}$ and $\mathcal{D}$ are not compatible*. Let k be the iteration at which this statement is returned by BuildPlus. Let $\pi_k = \{B_1, B_2, \ldots, B_m\}$ and, for all $i \in \{1, 2, \ldots, m\}$, let $C(B_i)$ be the minimal cluster of $\mathcal{T}$ that contains B_i. Without loss of generality, we may assume that $r(C(B_i)) \leq r(C(B_j))$ whenever $i \leq j$. Furthermore, suppose that $C_1, C_2, \ldots, C_n$ are the immediate descendants of $C(B_1)$ in $\mathcal{T}$, and let $S = B_1 - \bigcup_{j=1}^{n} C_j$ and $C'_j = B_1 \cap C_j$.

Now consider G'_k and, in particular, the blue component whose vertex set is B_1. Observe that the sets $S, C'_1, C'_2 \ldots, C'_n$ partition B_1. We next show that S is non-empty, and that each vertex in S has in-degree zero and no incident edges. To establish this, first observe that, as $(\mathcal{T}, r)$ ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$, the following hold in G'_k:

(i) for all $i \neq j$, there are no blue arcs joining a vertex in C'_i with a vertex in C'_j;
(ii) for all i, there are no blue arcs from a vertex in C'_i to a vertex in S;
(iii) for all i, there are no red edges joining a vertex in C'_i to a vertex in S;
(iv) for all $i \neq j$, there are no blue edges joining a vertex in C'_i with a vertex in C'_j; and
(v) for all i, there are no blue edges joining a vertex in C'_i with a vertex in S.

The fact that (i)—(iii) hold is easily seen. To see that (iv) holds, suppose that there is a blue edge in G'_k that joins a vertex a in C'_i to a vertex b in C'_j, and it is labelled cd. Because of the existence of this blue edge, c and d are in some common block B_i of π_k. Since $(\mathcal{T}, r)$ preserves $\mathcal{D}$ and, by choice of B_1, we have $r(C(B_1)) \leq r(C(B_i))$ for all i, it follows that $i = j$, and so (iv) holds. A similar argument shows that (v) also holds. Now, either exactly one of $S, C'_1, C'_2 \ldots, C'_n$ is non-empty in which case, by choice of $C(B_1)$, this must be S, or at least two of $S, C'_1, C'_2 \ldots, C'_n$ are non-empty. For the latter case, as B_1 is a connected blue component of G'_k, it follows by (i), (ii), (iv), and (v) that the only way this can happen is if S is non-empty and, for each C'_i of $C'_1, C'_2 \ldots, C'_n$ that is non-empty, there is a blue arc from a vertex in S to a vertex in C'_i. In both cases, S is non-empty. Moreover, if x is an element in S, then, by (ii), (iii), and (v), x has in-degree zero and no incident edges. But this means that $S(G'_k)$ is non-empty, contradicting the assumption that, at iteration k, BuildPlus returns *$\mathcal{P}$ and $\mathcal{D}$ are not compatible*. This completes the proof of (ii), and therefore the proof of the theorem. □

Proposition 2.4. *Let $\mathcal{P}$ be a collection of rooted semi-labelled trees and let $\mathcal{D}$ be a collection of relative divergence dates. Then the running time of* BuildPlus *applied to $\mathcal{P}$ and $\mathcal{D}$ is polynomial in n and m, where $n = |\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})|$ is the number of labels in the input, and $m = |\mathcal{P}| + |\mathcal{D}|$ is the number of constraints in the input. In particular, the running time is at most $O((nm)^3)$.*

Proof. Let $\mathcal{P}'$ be a collection of fully-labelled trees that is constructed from $\mathcal{P}$ by adding distinct new labels. In any tree in $\mathcal{P}$, since the only possible vertices that are labelled with a new label are either the root vertex or a vertex of degree at least three, the number of such interior vertices is at most one less than the number of leaves. The number of new labels is at most nm, and constructing $\mathcal{P}'$ can be done in time $O(nm)$. Thus to establish the proposition, it suffices to show the running time of BuildPlus is $O(n'^2nm)$, where $n' = |\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})|$.

In terms of running time, the most expensive parts of BuildPlus are the construction of $G(\mathcal{P}', \mathcal{D})$ and, at each iteration k, finding $E_r(G_k)$, $E_b(G_k)$, $S(G'_k)$, and π_{k+1}. Since $G(\mathcal{P}', \mathcal{D})$ has n' vertices, and we can easily determine whether a given pair of vertices are connected by each type of edge from the input, $G(\mathcal{P}', \mathcal{D})$ can be constructed in time $O(n'^2nm)$. Since G_k is a subgraph of $G(\mathcal{P}', \mathcal{D})$, we can determine the blue components in time $O(n'^2)$ and thus also compute $E_r(G_k)$, $E_b(G_k)$, $S(G'_k)$, and π_{k+1} in time $O(n'^2)$. Furthermore, provided we do not return $\mathcal{P}$ *and* $\mathcal{D}$ *are not compatible*, each iteration of BuildPlus, successively considers proper subgraphs of the graph considered in the previous iteration and these subgraphs always have at least one less vertex. Thus the number of iterations is at most n'. The proposition now follows. □

3. Listing All Consistent Ranked Hierarchies

The supertree method BuildPlus is useful to determine whether a collection $\mathcal{P}$ of rooted semi-labelled trees and a collection $\mathcal{D}$ of relative divergence dates are compatible. However, if this is the case, then care should be exercised in the way in which the ranked hierarchy returned by BuildPlus is used. This ranked hierarchy is simply one example of a ranked hierarchy that is consistent with the constraints inferred by $\mathcal{P}$ and $\mathcal{D}$. It is possible for there to be others, some of which may be quite structurally different. Such a situation gives rise to an interest in algorithms that return all ranked hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. In this section we present such an algorithm. This 'listing' algorithm is called AllBuildPlus.

Since the number of ranked hierarchies that are consistent with $\mathcal{P}$ and $\mathcal{D}$ may be exponential in the size of the input, we cannot hope to always list such hierarchies in polynomial time. However, the running time of AllBuildPlus is polynomial *in the size of the output*. More precisely, the running time of AllBuildPlus is bounded by $p(n, m)N$ for some polynomial p, where $n = |\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})|$, $m = |\mathcal{P}| + |\mathcal{D}|$, and N is the number of ranked hierarchies in the output. While such an algorithm has been exhibited for instances where the input is just a collection of phylogenetic trees [4, 8, 9], no algorithm handles input of semi-labelled trees and relative divergence dates. The algorithm AllBuildPlus permits both types of input.

To describe AllBuildPlus, we need some further definitions. Throughout these preliminaries, we will always assume that $\mathcal{P}$ and $\mathcal{D}$ are compatible collections of rooted semi-labelled trees and relative divergence dates. Strictly speaking, the

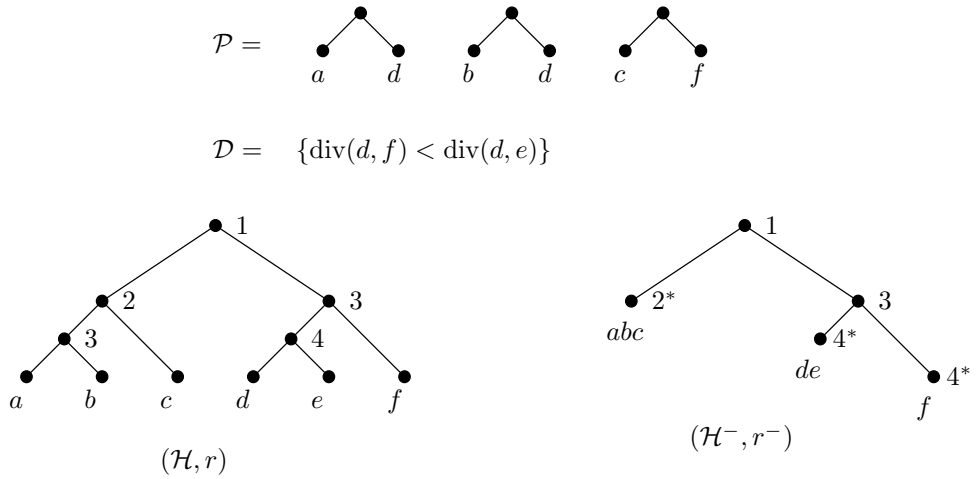


FIGURE 7. A ranked hierarchy $(\mathcal{H}, r)$ consistent with constraints $\mathcal{P}$ and $\mathcal{D}$, and its reduced form $(\mathcal{H}^-, r^-)$.

output returned by AllBuildPlus applied to $\mathcal{P}$ and $\mathcal{D}$ is, in fact, a set of so-called 'reduced hierarchies'. This is not a significant change, since the set of all consistent ranked hierarchies can be easily obtained from this set and, moreover, it has the possibility of significantly speeding up the running time of the algorithm. We begin these preliminaries by defining a reduced hierarchy.

Let $(\mathcal{H}, r)$ be a ranked hierarchy that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$. Informally, the reduced hierarchy $(\mathcal{H}^-, r^-)$ is the hierarchy formed by truncating each branch of $(\mathcal{H}, r)$ at the highest point below-which the constraints inferred by $\mathcal{P}$ and $\mathcal{D}$ have no further influence (see Fig. 7). Formally, the *reduced hierarchy* $(\mathcal{H}^-, r^-)$ of $(\mathcal{H}, r)$ with respect to constraints $\mathcal{P}$ and $\mathcal{D}$ is defined as follows. Let $\mathcal{U}$ be the set of maximal elements U in $\mathcal{H}$ such that, for all $a, b \in U$, a and b satisfy the following conditions:

(i) for all trees $\mathcal{T}$ in $\mathcal{P}$, $\{a, b\}$ is not a subset of $\mathcal{L}(\mathcal{T})$;
(ii) for all constraints $\mathrm{div}(c, d) < \mathrm{div}(e, f)$ in $\mathcal{D}$, $\{c, d\} \neq \{a, b\}$; and
(iii) for all constraints $\mathrm{div}(c, d) < \mathrm{div}(a, b)$ in $\mathcal{D}$, the rank of the minimal element of $\mathcal{H}$ containing c and d is strictly less than $r(U)$.

For each $U \in \mathcal{U}$, let $\mathcal{H}_U$ be the set of elements $H \in \mathcal{H}$ for which there exists some $\mathrm{div}(c, d) < \mathrm{div}(a, b) \in \mathcal{D}$ with $a, b \in U$ and $c, d \in H$. Set k_U to be

$$k_U = \max\{\{r(H) : H \in \mathcal{H}_U\} \cup \{r(H) : H \in \mathcal{H} \text{ and } U \text{ is a proper subset of } H\}\}+1.$$

Now set

$$\mathcal{H}^- = \mathcal{H} - \{H' \in \mathcal{H} : H' \text{ is a proper subset of } U \text{ for some } U \in \mathcal{U}\},$$

and, for all $H \in \mathcal{H}^-$, set

$$r^-(H) = \begin{cases} r(H), & \text{if } H \notin \mathcal{U}; \\ k^*_U, & \text{if } H \in \mathcal{U}. \end{cases}$$

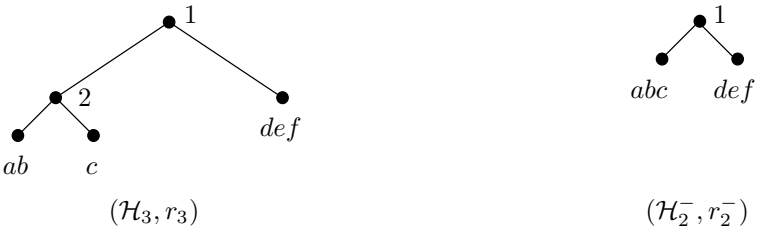


FIGURE 8. The 3-truncation $(\mathcal{H}_3, r_3)$ of $(\mathcal{H}, r)$, and the 2-truncation of $(\mathcal{H}^-, r^-)$.

Remarks.

(i) The value k_U is the minimal value that could be assigned to U such that $(\mathcal{H}^-, r^-)$ is a ranked hierarchy preserving $\mathcal{D}$.
(ii) The star denotes that any possible sub-hierarchy of U, with any ranking starting at at least k_U, will also ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$.
(iii) When dealing with reduced hierarchies, statements such as "all elements of rank at most k" will be taken to include all elements, starred or not; the star is simply a flag to highlight the cluster.
(iv) Every minimal element of a reduced hierarchy (leaf) is ranked and starred.
(v) As mentioned above, AllBuildPlus applied to $\mathcal{P}$ and $\mathcal{D}$ returns the set of all reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. Furthermore, the advantage of returning the set of reduced hierarchies over the appropriate set of ranked hierarchies is that there may be many fewer, and no information is lost in doing this. For example, if the original data forced 10 mammals to be in one cluster, but then offered no further information as to the substructure, there would be more than 2^{31} possible ranked sub-hierarchies of mammals (see [10], Proposition 2.3.4). However, AllBuildPlus simply produces a reduced hierarchy in which the set of mammals is flagged to indicate there is no further restriction on the structure or ranking of these mammals.

In addition to reduced hierarchies, AllBuildPlus makes use of 'truncated hierarchies'. For a ranked hierarchy $(\mathcal{H}, r)$, the *truncated hierarchy* $(\mathcal{H}_k, r_k)$ is the ranked hierarchy with

$$\mathcal{H}_k = \{H \in \mathcal{H} : r(H) \leq k-1\} \cup \{H \in \mathcal{H} : H \text{ is maximal such that } r(H) \geq k\}$$

and, for all $H \in \mathcal{H}_k$, $r_k(H) = r(H)$ if $r(H) \leq k-1$. To illustrate, two truncated hierarchies are shown in Fig. 8, where $(\mathcal{H}, r)$ and (H^-, r^-) are as shown in Fig. 7.

Before describing AllBuildPlus formally, we first give a brief and informative description of the algorithm. Let $\mathcal{R}$ be the set of reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. As in BuildPlus, we first construct a set $\mathcal{P}'$ of fully-labelled trees from $\mathcal{P}$ by adding distinct new labels. Intuitively, AllBuildPlus works by essentially computing a rooted tree, $\mathcal{T}_{\mathcal{R}}$ say, whose vertices consist of truncated hierarchies. This computation starts with the root, which is the ranked hierarchy consisting of the set $\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})$ and the null rank function, and works

its way towards the leaves of $\mathcal{T}_{\mathcal{R}}$. The set of leaves of $\mathcal{T}_{\mathcal{R}}$ is precisely the set of reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. Suppose P is a path $v_1v_2\cdots v_kv_{k+1}$ from the root of $\mathcal{T}_{\mathcal{R}}$ to a leaf $(\mathcal{H}, r)$. Then k is the maximum rank of a set in $(\mathcal{H}, r)$ and, for all $i \in \{1, 2, \ldots, k\}$, v_i corresponds to $(\mathcal{H}_i, r_i)$. The construction of $\mathcal{T}_R$ is breadth first construction, as opposed to a depth first construction. The advantages of the latter are discussed in the next section. Lastly, AllBuildPlus uses a subroutine called Augment. For each interior vertex of $\mathcal{T}_{\mathcal{R}}$, this subroutine is used to identify each of its immediate descendants. A further discussion of Augment is given after the description of AllBuildPlus.

Algorithm: AllBuildPlus$(\mathcal{P}, \mathcal{D})$
Input: A collection $\mathcal{P}$ of rooted semi-labelled trees and a collection $\mathcal{D}$ of relative divergence dates.
Output: The set of reduced hierarchies $\mathcal{R}$ on $\mathcal{L} = \mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})$ that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$.

begin
  Construct a set $\mathcal{P}'$ of fully-labelled trees from $\mathcal{P}$ by adding distinct new labels.
  Construct $G(\mathcal{P}', \mathcal{D})$.
  Let $\mathcal{L}' = \mathcal{L}(\mathcal{P}') - \mathcal{L}(\mathcal{P})$.
  $k \leftarrow 1$
  $\mathcal{R} \leftarrow \emptyset$
  $\mathcal{R}_1 \leftarrow \{(\{\mathcal{L} \cup \mathcal{L}'\}, r_1)\}$, where r_1 is the null function
  repeat while $\mathcal{R}_k$ is non-empty
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathcal{H}, r) \in \mathcal{R}_k$: for all $H \in \mathcal{H}$, $r(H)$ is defined$\}$
    $\mathcal{R}'_k \leftarrow \{(\mathcal{H}, r) \in \mathcal{R}_k$: there exists $H \in \mathcal{H}$, $r(H)$ is undefined$\}$
    $\mathcal{R}_{k+1} \leftarrow \bigcup_{(\mathcal{H},r)\in\mathcal{R}'_k}$ Augment$((\mathcal{H}, r), k)$
    $k \leftarrow k + 1$
  end (repeat).
  return $\{(\mathcal{H}, r)$: there exists $(\mathcal{H}', r') \in \mathcal{R}, (\mathcal{H}, r) = (\mathcal{H}', r')|\mathcal{L}\}$.
end.

We next describe Augment. Given a truncated hierarchy $(\mathcal{H}, r)$ in which the maximum rank of a set is $k-1$, Augment determines the possible choices for extending this ranked hierarchy so that the maximum rank of a set is k and each resulting hierarchy still leads to a reduced hierarchy that is consistent with $\mathcal{P}$ and $\mathcal{D}$. To do this, it makes use of the following graph. Let $G(\mathcal{H}, r)$ be the subgraph of $G(\mathcal{P}', \mathcal{D})$ that has vertex set

$$V = \{x : \text{there exists } H \in \mathcal{H} \text{ such that } x \in H,\ r(H) \text{ is undefined}\},$$

and edge and arc sets defined as follows. Let π be the partition of V induced by the minimal elements of $\mathcal{H}$. The red edge set of $G(\mathcal{H}, r)$ consists of those red edges $\{a, b\}$ in $G(\mathcal{P}', \mathcal{D})$ such that a and b are in the same block of π. Likewise, the blue edge set of $G(\mathcal{H}, r)$ consists of those blue edges $\{a, b\}$ labelled cd in $G(\mathcal{P}', \mathcal{D})$ such that c and d are in the same block of π. Lastly, the blue arc set of $(\mathcal{H}, r)$ consists of those blue arcs in $G(\mathcal{P}', \mathcal{D})$ both of whose endpoints are in V. The set of *unconstrained vertices* $W(\mathcal{H}, r) \subseteq V$ is the union of those minimal elements in $\mathcal{H}$ in which every member is an isolated vertex in $G(\mathcal{H}, r)$ and no member labels a blue edge in $G(\mathcal{H}, r)$. Furthermore, $S(G(\mathcal{H}, r))$ is the set of vertices in $V - W(\mathcal{H}, r)$

with in-degree zero and no incident edges. Intuitively, the edge set of $G(\mathcal{H}, r)$ is the subset of edges and arcs of $G(\mathcal{P}', \mathcal{D})$ corresponding to constraints not yet satisfied. This subgraph is used for assigning the rank k^* to all unconstrained elements of $\mathcal{H}$; computing the set $S(G(\mathcal{H}, r))$ of labels that could be of rank k; and, for each subset S of $S(G(\mathcal{H}, r))$, computing the possible maximal subsets of rank at least $k+1$ which do not fail any constraints.

In the description of AUGMENT, we write $\pi \leq \pi'$ if π and π' are two partitions on the same set and π' refines π.

Algorithm: AUGMENT$((\mathcal{H}, r), k)$
Input: A ranked hierarchy $(\mathcal{H}, r)$ and an integer k.
Output: A set $\mathcal{Q}$ of ranked hierarchies, each of which is a rank one extension of $(\mathcal{H}, r)$.

begin
  $\mathcal{Q} \leftarrow \emptyset$
  $G \leftarrow G(\mathcal{H}, r)$
  $W \leftarrow W(\mathcal{H}, r)$
  for each subset S of $S(G)$ that contains $S(G) \cap \mathcal{L}'$ **do**
    $G_S \leftarrow G - (S \cup W)$
    Let π be the partition of $V(G)$ induced by the minimal elements of $\mathcal{H}$.
    Let π_S be the partition of $V(G_S)$ induced by the blue components of G_S.
    for each partition π' of $V(G_S)$ with $\pi|V(G_S) \leq \pi' \leq \pi_S$ and $\pi' \neq \pi$ **do**
      $\mathcal{H}_{S,\pi'} \leftarrow \mathcal{H} \cup \{A : A \text{ is a block of } \pi'\}$
      $r_{S,\pi'}(A) \leftarrow r(A)$ if $A \in \mathcal{H}$
      for each subset B in π which is not in π' **do**
        $r_{S,\pi'}(B) \leftarrow k$
      end (for)
      for each minimal element H in $\mathcal{H}$ which is a subset of W **do**
        $r_{S,\pi'}(H) \leftarrow k^*$
      end (for)
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup (\mathcal{H}_{S,\pi'}, r_{S,\pi'})$
    end (for)
  end (for)
  return $\mathcal{Q}$
end.

Theorem 3.1 shows that ALLBUILDPLUS does indeed return the desired set of reduced hierarchies. The running time of ALLBUILDPLUS is considered after the proof of this theorem.

Theorem 3.1. *Let $\mathcal{P}$ and $\mathcal{D}$ be compatible collections of rooted semi-labelled trees and relative divergence dates, respectively. Then the set returned by* ALLBUILDPLUS *when applied to $\mathcal{P}$ and $\mathcal{D}$ is exactly the set of reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$.*

Proof. In order to establish Theorem 3.1, we show that

(i) every reduced hierarchy that is consistent with $\mathcal{P}$ and $\mathcal{D}$ is in the set returned by AllBuildPlus; and
(ii) every hierarchy in the set returned by AllBuildPlus is consistent with $\mathcal{P}$ and $\mathcal{D}$, and is in reduced form.

To prove (i), let $(\mathcal{H}, r)$ be a reduced hierarchy that ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$, and suppose that the maximum rank of a set in $\mathcal{H}$ is t. Let $(\mathcal{H}', r')$ be the ranked hierarchy on labels $\mathcal{L} \cup \mathcal{L}'$ obtained from $(\mathcal{H}, r)$ as follows. For each $x \in \mathcal{L}'$, x appears in a unique tree $\mathcal{T}_x$ in $\mathcal{P}'$. Let D_x be the set of labels in $\mathcal{L}$ that are descendants of x in $\mathcal{T}_x$, and let N_x be the set of labels in $\mathcal{L}$ that are not comparable to x in $\mathcal{T}_x$. Then H_x is the maximal element of $\mathcal{H}$ which contains D_x, but contains no member of N_x. Set $\mathcal{H}' = \{H \cup \{x : H_x \subseteq H\} : H \in \mathcal{H}\}$, and $r'(H') = r(H'|\mathcal{L})$ for $H' \in \mathcal{H}'$. Since $(\mathcal{H}, r)$ displays $\mathcal{P}$, $(\mathcal{H}', r')$ is well defined and displays $\mathcal{P}'$ (and preserves $\mathcal{D}$). Also $(\mathcal{H}', r')|\mathcal{L} = (\mathcal{H}, r)$.

We will show that, for each $i \leq t$, $(\mathcal{H}'_i, r'_i)$ is in the set $\mathcal{R}_i$ constructed by AllBuildPlus at iteration $i-1$. Furthermore, at iteration $t+1$ of AllBuildPlus, $(\mathcal{H}', r')$ is added to the set $\mathcal{R}$. We prove the first part by induction on i. If $i = 1$, then, as $(\mathcal{H}'_1, r'_1)$ is the ranked hierarchy consisting of the set $\mathcal{L} \cup \mathcal{L}'$ and the null rank function, it is clear that the first part holds.

Now suppose that k is an integer with $k \leq t$ and that the first part holds for all i such that $i < k$. Then $(\mathcal{H}'_{k-1}, r'_{k-1})$ in $\mathcal{R}_{k-1}$. By exhibiting a suitable choice of S and π', we will show that $(\mathcal{H}'_k, r'_k)$ is in the set returned by Augment$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$. For the rest of this part of the proof we will denote $G(\mathcal{H}'_{k-1}, r'_{k-1})$ by G and $W(\mathcal{H}'_{k-1}, r'_{k-1})$ by W, as we are only interested in the computation of Augment$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$,

Let $\mathcal{L}_{k-1}$ be the set of elements a in $\mathcal{L} \cup \mathcal{L}'$ such that the minimal element of $\mathcal{H}'$ containing a has rank $k-1$, and consider G. Since $(\mathcal{H}', r')$ is consistent with $\mathcal{P}'$ and $\mathcal{D}$, each element in $\mathcal{L}_{k-1}$ is a vertex of G with in-degree zero and no incident edges. Furthermore, those elements in $(\mathcal{H}', r')$ ranked $(k-1)^*$ are subsets of W, and so every such element is ranked $(k-1)^*$ in every member of the set returned by Augment$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$. Let $S = \mathcal{L}_{k-1} - W$. Then, by the construction of $(\mathcal{H}', r')$, S satisfies $S(G) \cap \mathcal{L}' \subseteq S \subseteq S(G)$.

Let π' be the set of maximal elements in $(\mathcal{H}', r')$ with rank at least k. Since π in the computation of Augment$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$ is the partition of $V(G)$ induced by the set of minimal elements in $\mathcal{H}'_{k-1}$, it follows that π is the set of maximal elements in $\mathcal{H}'$ with rank at least $k-1$. Therefore π' is a refinement of $\pi|V(G_S)$.

To see that π_S is a refinement of π', suppose that there are elements x and y such that x and y are in separate blocks of π', but are in the same block of π_S in the computation of Augment$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$. Since x and y are in the same block of π_S, they must be in the same blue component of G_S. It is easily seen that without loss of generality, we may assume that x and y are either joined by a blue edge or a blue arc in G_S. Since x and y are in separate blocks of π', we now deduce that if x and y are joined by a blue edge in G_S, then $(\mathcal{H}', r')$ does not preserve an

element in $\mathcal{D}$, while if x and y are joined by a blue arc, then it does not display one of the ancestor relationships in $D(\mathcal{P}')$. This gives a contradiction, hence π_S is a refinement of π'. Furthermore, $\pi' \neq \pi$. For otherwise, if $\pi' = \pi$, then we must have $\mathcal{L}_{k-1} = \emptyset$. This implies that the ranked semi-labelled tree associated with $(\mathcal{H}', r')$ must contain an unlabelled vertex v which has rank $k-1$, and this vertex must have at least two immediate descendant vertices. The cluster associated with v is in π, but the clusters associated with its immediate descendant vertices are in π'; a contradiction.

We now deduce that with the above choice of S and π', the ranked hierarchy $(\mathcal{H}_{S,\pi'}, r_{S,\pi'})$ is a member of AUGMENT$((\mathcal{H}'_{k-1}, r'_{k-1}), k-1)$, and $\mathcal{H}_{S,\pi'} = \mathcal{H}'_k$ and $r_{S,\pi'} = r'_k$. By induction, it now follows that at iteration t, the ranked hierarchy $(\mathcal{H}'_{t+1}, r'_{t+1})$ is in the set $\mathcal{R}_{t+1}$ computed by ALLBUILDPLUS. Since the maximum rank of any element in $(\mathcal{H}', r')$ is t, $(\mathcal{H}'_{t+1}, r'_{t+1}) = (\mathcal{H}', r')$. Finally, since every element of $\mathcal{H}'$ is ranked, $(\mathcal{H}', r')$ is added to $\mathcal{R}$ in iteration $t+1$. As $(\mathcal{H}', r')|\mathcal{L} = (\mathcal{H}, r)$, this completes the proof of (i).

To prove (ii), first observe that the proof of Theorem 2.3 may be easily adapted to show that any ranked hierarchy produced by ALLBUILDPLUS ancestrally displays $\mathcal{P}$ and preserves $\mathcal{D}$. Thus to establish (ii), it remains to show that ALLBUILDPLUS returns only reduced hierarchies. Suppose that $(\mathcal{H}, r)$ is a ranked hierarchy that is returned by ALLBUILDPLUS, but it is not reduced; that is $(\mathcal{H}, r) \neq (\mathcal{H}^-, r^-)$. Let k be the least integer such that there is an element $H^- \in \mathcal{H}^-$ with $r^-(H^-) = k^*$, but $r(H^-) \neq k^*$. Now $(\mathcal{H}_k, r_k) = (\mathcal{H}_k^-, r_k^-)$. By (i)-(iii), in the definition of reduced hierarchy, $H^- \subseteq W$ at the appropriate stage of the computation of ALLBUILDPLUS, and so $r(H^-)$ is set to k^*. This contradiction completes the proof of (ii). □

We now turn our attention to the running time of ALLBUILDPLUS. First, we bound the running time of AUGMENT in terms of the size of its output.

Lemma 3.2. *Let $\mathcal{P}'$ be a collection of rooted fully-labelled trees and let $\mathcal{D}$ be a collection of relative divergence dates. Let $n' = |\mathcal{L}(\mathcal{P}') \cup \mathcal{L}(\mathcal{D})|$ and $m = |\mathcal{P}'| + |\mathcal{D}|$. Let $(\mathcal{H}, r)$ be a truncated hierarchy in the set $\mathcal{R}_k$ that is obtained in the computation of* ALLBUILDPLUS$(\mathcal{P}', \mathcal{D})$. *Then the running time of* AUGMENT$((\mathcal{H}, r), k)$ *is bounded by $p(n', m)|\mathcal{Q}|$, where $p(n', m)$ is a polynomial in n' and m, and $\mathcal{Q}$ is the set returned by* AUGMENT$((\mathcal{H}, r), k)$.

Proof. As $G(\mathcal{P}', \mathcal{D})$ has n' vertices, it is clear that $G(\mathcal{H}, r)$ can be constructed in polynomial time in n' and m. Furthermore, determining $W(\mathcal{H}, r)$ and $S(G(\mathcal{H}, r))$ can also be done in polynomial time in n' and m. Now, for each choice of S and π', the time taken to construct $\mathcal{H}_{S,\pi'}$ is polynomial in n' and m, since we only consider each element of π' of which there are at most n'. Also, the time taken to construct $r_{S,\pi'}$ is polynomial in n' and m, since we only consider each element of $\mathcal{H}_{S,\pi'}$, of which there are at most $2n'$. Finally, each distinct choice of S and π', clearly yields a distinct element of $\mathcal{Q}$. Hence the running time of AUGMENT$((\mathcal{H}, r), k)$ is bounded by $p(n', m)|\mathcal{Q}|$ for some polynomial p in n' and m. □

Proposition 3.3. *Let $\mathcal{P}$ be a collection of rooted semi-labelled trees, let $\mathcal{D}$ be a collection of relative divergence dates. Then the running time of* ALLBUILDPLUS

applied to $\mathcal{P}$ and $\mathcal{D}$ is bounded by $p'(n,m)|\mathcal{R}|$, where p' is a polynomial, $n = |\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})|$, $m = |\mathcal{P}|+|\mathcal{D}|$, and $\mathcal{R}$ is the set of reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$.

Proof. As for BuildPlus, since $n' = |\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})|$ is at most $nm+n$, it is sufficient to show that the running time is bounded by $p''(n',m)|\mathcal{R}|$. It is easily checked that each of the preliminaries in AllBuildPlus can be done in time polynomial in n' and m. The repeat loop in AllBuildPlus is used to compute $\mathcal{R}_k$ from $\mathcal{R}_{k-1}$ and, as the maximum rank of any set in a reduced hierarchy in $\mathcal{R}$ is $2n'$, this repeat is done at most $2n'$ times. In each repeat, the algorithm scans every member of $\mathcal{R}_k$ to determine if it is complete, in which case it is assigned to $\mathcal{R}$, or incomplete, in which case it is assigned to $\mathcal{R}'_k$ and Augment is called. This takes time at most $n'm|\mathcal{R}_k|$ plus the time for calls to Augment. Since $|\mathcal{R}_k| \leq |\mathcal{R}|$, the total time for the repeat loop is at most $2n'^2m|\mathcal{R}|$ plus the time for calls to Augment.

It is clear that the subroutine Augment is called only once for each non-leaf vertex $(\mathcal{H}_k, r_k)$ in the tree $\mathcal{T}_\mathcal{R}$, since the corresponding truncated hierarchy appears in only one of the sets $\mathcal{R}_k$ during the computation of AllBuildPlus. Also, there is a unique element $(\mathcal{H}_{k-1}, r_{k-1}) \in \mathcal{R}_{k-1}$ such that $(\mathcal{H}_k, r_k)$ is in the set returned by Augment$((\mathcal{H}_{k-1}, r_{k-1}), k)$. Now, by Lemma 3.2, the running time of Augment for a vertex of $\mathcal{T}_\mathcal{R}$ with l descendants is bounded by $p(n',m)l$ for some polynomial in n' and m. Hence the total running time for all calls to Augment is bounded by $p(n',m)|\mathcal{T}_\mathcal{R}|$, where $|\mathcal{T}_\mathcal{R}|$ is the number of vertices in $\mathcal{T}_\mathcal{R}$. Since $\mathcal{T}_\mathcal{R}$ has $|\mathcal{R}|$ leaves and has depth at most $2n'$, this is bounded by $2p(n',m)n'|\mathcal{R}|$. We deduce that the total running time of AllBuildPlus applied to $\mathcal{P}$ and $\mathcal{D}$ is bounded by $(2p(n',m)n' + 2n'^2m)|\mathcal{R}| = p'(n,m)|\mathcal{R}|$ for some polynomial p' in n and m. □

4. Listing Hierarchies with Polynomial Delay

In the this section we describe briefly how AllBuildPlus can be easily modified to output the reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$ one at a time (rather than all together as in AllBuildPlus), so that the time to produce the first reduced hierarchy, and the time delay between producing one reduced hierarchy and the next, is bounded by a polynomial in $|\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{D})|$ and $|\mathcal{P}| + |\mathcal{D}|$ only. Note that it may still take an exponential amount of time to produce the entire list of reduced hierarchies.

Recall that AllBuildPlus is effectively a breadth first search of the tree $\mathcal{T}_\mathcal{R}$, where $\mathcal{R}$ is the set of reduced hierarchies that ancestrally display $\mathcal{P}$ and preserve $\mathcal{D}$. In order to generate the elements of $\mathcal{R}$ with only a polynomial time delay between each output, it is necessary to convert the algorithm to a 'depth first search'. In other words, rather than computing the entire set $\mathcal{R}_k$ at each stage of AllBuildPlus, the algorithm would pick one element from $\mathcal{R}_k$ and then compute an appropriate descendant of that in $\mathcal{T}_\mathcal{R}$. Once a leaf of $\mathcal{T}_\mathcal{R}$ had been reached, the algorithm would backtrack as far as necessary up the tree to find a vertex with a descendant not yet considered. By following $\mathcal{T}_\mathcal{R}$ down this new branch until a leaf is reached, and then repeating the process, the algorithm eventually visits every

leaf of $\mathcal{T}_{\mathcal{R}}$. The time taken to find the first leaf is the same as the time BuildPlus takes to run, and hence polynomial in the input size. It requires at most $2n$ times as much time and memory to keep track of the path taken through $\mathcal{T}_{\mathcal{R}}$, and which vertices had alternative descendants. Hence the backtracking can be achieved in polynomial time, and computing the path down a new branch has the same time bound as computing the original path, which was polynomial. Therefore the delay between returning one member of $\mathcal{R}$ and the next is polynomial in the size of $\mathcal{P}$ and $\mathcal{D}$.

References

[1] A. V. Aho, S. Yehoshua, 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. Comput. **10** (1981) 405–421.

[2] M. Bordewich and C. Semple, Counting consistent phylogenetic trees is #P-complete, Advances in Applied Mathematics **33** (2004) 416–430.

[3] D. Bryant, C. Semple and M. Steel, Supertree methods for ancestral divergence dates and other applications, In: Phylogenetic Supertrees: combining information to reveal the Tree of Life, O. Bininda-Emonds, Ed., Computational Biology Series, Kluwer, (2004), pp. 129–150.

[4] M. Constantinescu and D. Sankoff, An efficient algorithm for supertrees, J. Classif. **12** (1995) 101–112.

[5] P. Daniel, W. Hordijk, R. D. M. Page, C. Semple, and M. Steel, Supertree algorithms for ancestral divergence dates and nested taxa, Bioinformatics, **20** (2004) 2355–2360.

[6] P. Daniel and C. Semple, Supertree algorithms for nested taxa, In: Phylogenetic supertrees: combining information to reveal the Tree of Life, O. Bininda-Emonds, Ed., Computational Biology Series, Kluwer, (2004), pp. 151–171.

[7] P. Daniel and C. Semple, A class of general supertree methods for nested taxa, SIAM J. Discrete Math., in press.

[8] M. P. Ng and N. C. Wormald, Reconstruction of rooted trees from subtrees, Discrete Appl. Math. **69** (1996) 19–31.

[9] C. Semple, Reconstructing minimal rooted trees, Discrete Appl. Math., **127** (2003) 489–503.

[10] C. Semple and M. Steel, Phylogenetics, Oxford University Press, 2003.

School of Computing, University of Leeds, Leeds, United Kingdom

E-mail address: magnusb@comp.leeds.ac.uk

Biomathematics Research Centre, Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand

E-mail address: c.semple@math.canterbury.ac.nz, gee16@student.canterbury.ac.nz