Taxonomic supertree construction with *Incertae sedis* taxa

April 3, 2018

1 Introduction

Supertree methods combine a collection of input trees with different taxon sets into a single tree on the combined taxon set. While such methods are usually applied to leaf-labeled phylogenies that have been estimated from DNA sequence data, we seek to include taxonomic information also. Taxonomies can provide a comprehensive coverage of known taxa that inferred phylogenies cannot. Additionally, taxonomies can provide taxonomic names, which are needed to align input phylogenies to each other, and to allow references to existing groups.

We thus require that one of the input trees is a taxonomy tree. The taxonomy has a number of features not found in the input phylogenies. While the input trees need not cover the entire taxon set, we require the taxonomy to be comprehensive. The taxonomy also has labels for internal nodes, whereas input phylogenies only have labels for leaf nodes. We refer to labels of taxonomy nodes as "taxon names", and require that taxon names and taxonomy nodes have a one-to-one correspondence. The taxonomy may also contain out-degree 1 nodes. These nodes correspond to monotypic taxa which contain a single child of lower taxonomic rank.

In practice, taxonomies may also contain nodes with uncertain placement. These nodes are often labelled "*incertae sedis*", which means "uncertain seat" in Latin. The common interpretation of such taxa is that they may not be moved further towards the root, but may be moved into their sibling taxa. For example, a genus with a sibling that is a family may be annotated as"*incertae familia*", indicating that it is unclear which family the genus should be placed in. *Incertae sedis* taxa frequently occur when a specimen is identified up to a given taxonomic level, but no further. Extinct taxa are often *incertae sedis*.

The Open Tree of Life project seeks to build a comprehensive supertree covering all of life. The approach is to combine information in published phylogenies with a comprehensive taxonomy that supplies taxon names (Redelings and Holder, 2017). We describe an extension to this supertree method so that we can resolve taxonomic uncertainty by using published phylogenies to place *incertae sedis* taxa. We describe an updated semantics for conflict between phylogeny edges and taxonomy edges and an updated semantics for assigning taxonomy names to nodes in the synthetic tree.

Our approach to *incertae sedis* taxa enables us to use published phylogenies to update the taxonomy by extending taxonomic groups to include *incertae sedis* taxa placed within them. The ability to update the taxonomy is important, since the placement of *incertae sedis* taxa creates artificial conflicts with a non-updated taxonomy. Thus, the ability to correctly assess compatibility with the taxonomy when the taxonomy contains *incertae sedis taxa* allows us to include thousands of new taxa that were previously filtered out to avoid conflict. It also enables us to include extinct taxa, since many of these taxa are *incertae sedis*. This makes substantial progress towards our goal of completely representation of taxa.

1.1 Background

Terminology We briefly describe the supertree algorithm proposed in Redelings and Holder (2017). This supertree algorithm takes as input a ranked list $\mathcal{T} = \{T_1, \ldots, T_n\}$ of rooted input phylogenies and a single rooted taxonomy tree $\mathbb{T}$. Each node in the taxonomy tree is associated with a taxon name in the set of taxon names $\mathcal{N}$, and each taxon name is corresponds to only one node of the taxonomy tree. We use $\mathcal{L}$ to indicate taxon names corresponding to leaf nodes of the taxonomy tree. Correspondence between the input trees and the taxonomy is established by annotating leaf nodes of each input tree with taxon names. For purposes of conflict resolution, the taxonomy tree is ranked below all of the input trees.

In this manuscript we extend this framework by adding a set $\mathcal{U} \subseteq \mathcal{N}$ of labels that have the *incertae sedis* property. The extended taxonomy then consists of the taxonomy tree $\mathbb{T}$, the labels $\mathcal{N}$, and set $\mathcal{U}$ of *incertae sedis* taxa.

Exemplification While the taxonomy is comprehensive and contains all taxa, input phylogenies only have labels on their leaf nodes, and generally include a small fraction of the total number of taxa. Furthermore, taxa in input trees may correspond to higher taxa on the taxonomy tree, and multiple tips may be labeled with the same taxon name. Redelings and Holder (2017) described a method called 'exemplification' that reduces this problem to a simpler form where all taxon names of input trees (i) correspond to leaf taxa and (ii) occur only once in an input tree. For the purposes of this paper, we therefore assume that this is the case.

Problem decomposition Given the exemplified input trees and taxonomy, the supertree algorithm first divides the supertree problem into independent pieces, solves the problem on these pieces, and then glues the pieces back together. This is made possible by the assumption that any taxonomy edges that are not contested by any of the input trees can be assumed to occur in the supertree. We may then solve separately for what happens on each side of such an uncontested edge. When the uncontested edge is attached to the solution below the edge and the solution above the edge, the complete solution is then obtained.

Splits-based synthesis method Supertrees for each subproblem are obtained by greedily constructing a set of rooted splits $\mathcal{C}$ that are jointly consistent and correspond to branches of the input trees. We construct an ordered list of splits $S(T_i)$ by walking the branches of each tree T_i in post-order and appending the corresponding splits to the list. These lists are combined to form the list $\Sigma = S(T_1) + S(T_2) + \ldots + S(T_n) + S(\mathbb{T})$, where '+' denotes concatenation. The set C is initialized to the empty set, and we then iteratively consider each split in the list Σ and insert it into C if the resulting set remains jointly compatible. The BUILD algorithm is used both to check compatibility, and to construct a compatible tree from the final set C.

2 Semantics of *incertae sedis* taxa

We seek a semantics for supertrees with *incertae sedis* taxa that satisfy the following properties:

1. An incertae sedis node may intrude into its siblings and their descendants.
2. Two incertae sedis siblings may not be interdigitated.
3. Any placement $T \rightarrow T^*$ of a tree T with incertae sedis taxa is also accessible from a tree T' that is created from T by taking a taxon, attaching it to an ancestor, and marking it *incertae sedis.*

4. The semantics of an *incertae sedis* node does not depend on assigned ranks, but only on the tree.
5. Semantics is based on deriving a split for each branch of the tree.

Here we focus on a split-based semantics instead of a semantics based on apply of a series of edit operations to the tree. The split-based semantics allows us to seek trees that are simultaneously consistent with a set of split-constraints. The use of a split-based semantics does impose some limitations. These come primarily because in the edit-based semantics, after we place an *incertae sedis* taxon, it is no loner *incertae sedis*, whereas with the split-based semantics taxa are either always *incertae sedis* or never *incertae sedis*.

This has an effect, for example, when we consider two incertae sedis siblings A and B. If we can place A within B, and B within A, then a split-based semantics allows us to freely inter-digitate A and B. If we do not want to allow A and B to be inter-digitated, then we cannot both A within B and B within A. In addition, we can imagine that a family A is incertae sedis within a kingdom. In an edit-based semantics, we could place A as a sibling to a genus B, mark A as non-*incertae-sedis*, and then place B within A. In a split-based semantics, we must disallow either B within A or A within B if we want to disallow interdigitation.

As a result of this, we cannot satisfy all five of these criteria. Thus, we describe two alternative split-based semantics. The non-interdigitation semantics gets # (1,2,4,5), and the interdigitation semantics gets (1,3,4,5). This makes the point that each semantics results from different choices about which criteria to satisfy.

2.1 Split-based semantics for *incertae sedis* taxa

In order to apply supertree methods to rooted trees with *incertae sedis* taxa, we first propose a semantics of *incertae sedis* in terms of rooted splits. We define rooted splits by noting that each edge of a tree divides the tip taxa $\mathcal{L}$ into two groups: the include set $\mathcal{I}(e)$ which does not contain the root, and the exclude set $\mathcal{E}(e)$ which does contain the root. Such a split may be written

$$\mathcal{I}(e) | \bullet \mathcal{E}(e).$$

where the $\bullet$ indicates the root. If no taxa are *incertae sedis*, then the exclude set for a node is just the total tip set minus the include set for the node:

$$\mathcal{E}(n) = \mathcal{L} - \mathcal{I}(n).$$

For a node n on the tip-ward side of an edge e, we may also write $\mathcal{I}(n)$ for $\mathcal{I}(e)$, and $\mathcal{E}(n)$ for $\mathcal{E}(e)$.

2.1.1 A semantics that disallows inter-digitating *incertae sedis* siblings

An *incertae sedis* taxon can be moved into any of the descendants of its siblings. We seek to represent this by constructing modified splits for each branch of the taxonomy tree. The include sets of these splits remain unchanged, but we construct reduced exclude sets in order to allow *incertae sedis* taxa to intrude into their sibling taxa.

The reduced exclude set $\mathcal{E}(n)$ for node n should therefore not contain the descendants of its siblings marked *incertae* sedis*,* but should contain the descendants of its other siblings. We additionally stipulate that it is not possible for *incertae sedis* taxa to intrude into their *incertae sedis* siblings. (This is not the only way to construct a split-based semantics for *incertae sedis*, but it avoids certain complications, as noted in the discussion.) We thus define $\mathcal{I}(m,n)$ to indicate descendants of m that are excluded from n. Thus:

$$\mathcal{I}(m,n) = \begin{cases} \emptyset & \text{if } n \text{ is } \textit{incertae sedis} \text{ and } m \text{ is not} \\ \mathcal{I}(m) & \text{otherwise} \end{cases}$$

Additionally, the exclude set $\mathcal{E}(n)$ should contain the children of all the ancestors of n, unless those children are *incertae sedis*. To avoid traversing all ancestors separately for each node n, we note that the excluded children of any ancestors will be in the exclude set of the parent of n already. Therefore, we can write the exclude set of a node n in terms of the exclude set of its parent:

$$\mathcal{E}(n) = \mathcal{E}(parent(n)) \cup \left[\mathcal{I}(m,n)\middle| m \in siblings(n)\right]. \tag{1}$$

This formula allows us to compute exclude sets via a pre-order recursion on the taxonomy tree. This recursion terminates if the exclude set for the root node is set to $\emptyset$ as a boundary condition.

Problem: how about the goal that we should be able to get the same outcome by moving a taxon up one level and marking it as incertae sedis? This might require that moving A next to B, and then moving B into A is allowed.

Problem: should we be able to move incertae sedis taxa down into incertae sedis taxa? If not, then we must change the recursion.

2.1.2 An alternative semantics that allows inter-digitating incertae sedis siblings

If we want to allow placing two incertae sedis siblings within each other, then incertae sedis siblings are never excluded.

Furthermore, consider placing A as a sibling of B, and then placing B within A. In order for this to work, A must not exclude the incertae sedis children of B.

In order to accomplish both of these goals, we define the descendants $\mathcal{I}'(n)$ that are accessible without passing through an incertae sedis node.

$$\mathcal{I}'(n) = \begin{cases} \emptyset & \text{if } n \in \mathcal{U} \\ \{n\} & \text{if } n \text{ is a leaf} \\ [\mathcal{I}'(c) | c \in children(n)] & \text{otherwise.} \end{cases}$$

We may then define the taxa excluded from n as:

$$\mathcal{E}(n) = \mathcal{E}(parent(n)) \cup [\mathcal{I}'(s) | s \in siblings(n)].$$

This semantics satisfies criteria 1,3,4, and 5 above.

2.1.3 A third semantics

OK, the idea here is that we want to avoid inter-digitation. *Incertae sedis* taxa thus exclude *incertae sedis* siblings. However, where the first semantics allows *incertae sedis* taxa to intrude into *incertae* *sedis* descendants of siblings, but not allow *incertae sedis* descendants of siblings to intrude into *incertae sedis* taxa, here we do the reverse.

OK, so the idea here is that an *incertae sedis* group excludes

- *incertae sedis* siblings
- *incertae sedis* descendants of ancestors
- but not *incertae sedis* descendants of non-incertae-sedis siblings

It is also possible to allow placing A as a sibling of B and then placing B within A if we disallow placing A into incertae sedis descendants of its siblings. We may then allow placing those descendants within A.

$$\mathcal{I}'(n) = \begin{cases} \emptyset & \text{if } n \in \mathcal{U} \\ \{n\} & \text{if } n \text{ is a leaf} \\ [\mathcal{I}'(c) | c \in \mathit{children}(n)] & \text{otherwise.} \end{cases}$$

We use $\mathcal{I}(m,n)$ to indicate the descendants of m that are excluded from n.

$$\mathcal{I}(m,n) = \begin{cases} \emptyset & \text{if } n \text{ is } \textit{incertae sedis} \text{ and } m \text{ is not} \\ \mathcal{I}(m) & \text{if } n \text{ and } m \text{ are both } \textit{incertae sedis} \\ \mathcal{I}'(m) & \text{otherwise.} \end{cases}$$

We may then define the taxa excluded from n as:

$$\mathcal{E}(n) = \begin{cases} [\mathcal{L} - \mathcal{I}(\mathit{parent}(n))] \cup [\mathcal{I}(s,n) | s \in \mathit{siblings}(n)] & \text{if } n \in \mathcal{U} \\ \mathcal{E}(\mathit{parent}(n)) \cup [\mathcal{I}(s,n) | s \in \mathit{siblings}(n)] & \text{otherwise.} \end{cases}$$

I think this works... but need to double-check.

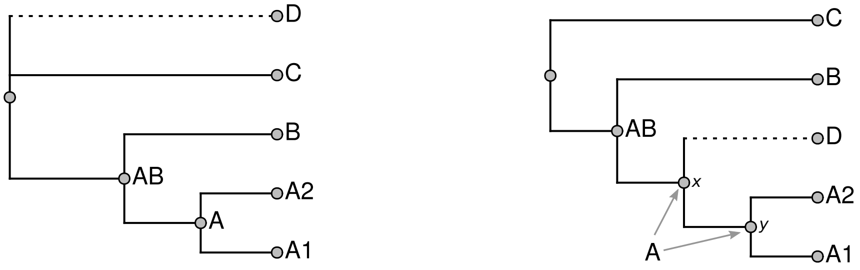
Also: can we genericize $\mathcal{E}(\mathit{parent}(n))$?

2.2 Naming

After constructing a supertree, we still need to assign taxon names to the supertree nodes based on the taxonomy tree in the problem. Each taxon name n corresponds to a split $S(n) = S(n)_1 | \bullet S(n)_2$ on the corresponding branch of the taxonomy tree. Without *incertae sedis*, such splits are always of the form $S(n)_1 | \bullet \mathcal{L} - S(n)_1$, but with *incertae sedis* taxa $S_2(n)$ may be smaller than $\mathcal{L} - S(n)_1$.

Without *incertae sedis*, each name applies to at most one node, and each node can take at most one name, with the exception of monotypic taxa. Thus, we may simply search the solution tree for a node that has the same cluster $S(n)_1$ and apply the name n to that node.

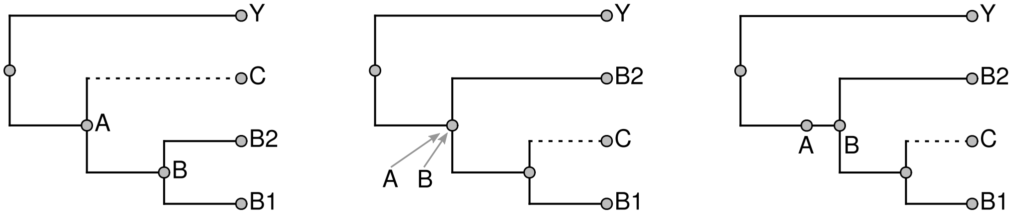
However, in the *incertae sedis* framework, it is possible for one name to apply to multiple nodes. For example, in Figure 1b, the name A can apply to nodes x and y. Here the name A corresponds to the split $A1\,A2 \,|\, \bullet\, B\,C$, leaving out D since it is *incertae sedis*. The two nodes x and y imply the splits $A1\,A2 \,|\, \bullet\, B\,C\,D$ and $A1\,A2\,D | \bullet\, B\,C$ respectively, and both of these splits imply the split $A1\,A2 \,|\, \bullet\, B\,C$,



(a) Taxonomy tree

(b) Solution tree

Figure 1: One name can be consistent with multiple nodes.



(a) Taxonomy tree

(b) Solution tree

(c) Solution tree

Figure 2: Multiple names can apply to a single node.

so the name A can apply to both x and y^1. This cannot happen without *incertae sedis* taxa except at monotypic nodes. When faced with a choice about where to place a name, our solution is to find the most tip-ward node where the name can apply and attach the name to this node. This corresponds to a conservative choice about whether *incertae sedis* taxa should be added to an existing taxon.

It is also possible for multiple names to apply to a single node. For example, in Figure 2a, the taxon A corresponds to the split $B1\,B2\,C \mid \bullet\, Y$, and its child taxon B corresponds to the split $B1\,B2 \mid \bullet\, Y$. The edge leading to A is consistent with the split for B, but the name B is applied to the node with the smallest include group. However, in the solution tree (Fig. 2b), there is is only node for both names A and B to apply to. In this case, we solve this problem by introducing a monotypic parent, and applying A to the newly created parent node (Fig. 2c).

2.3 Placement

After a synthesis tree is constructed, we attach taxon names to the synthesis tree for each non-conflicting taxon. These names attach at the MRCA of the include group. Then, for each taxon A that is found on the synth tree, we associate with the first taxon B found in a rootward walk on the synthesis tree. If A is not a descendant of B on the taxonomy, then we say that A is *placed* within B by the synthesis tree. While this may occur if A is *incertae sedis*, more complex scenarios are possible. We note that if method does not handle *incertae sedis* taxa, then this scenario could not occur because any taxa B that A is placed within would be considered incompatible with the synthesis tree and so their names would not be applied.

In order to fully describe a placement of A within B we consider the sequence of taxa encountered

[1] Define "implies", as in the propinquity paper: $A_1 | \bullet A_2 \implies B_1 | \bullet B_2$ means $B_1 \subseteq A_1$ and $B_2 \subseteq A_2$.

on a walk from A or B up to their MRCA M on the taxonomy tree. Let us denote the sequence of taxa encountered from A as $A_1, \ldots, A_n, M$ where $A_1 = A$. Let $B_1, \ldots, B_m, M$ be the sequence of taxa encountered from B where $B_1 = B$. Then A_n must be *incertae sedis*, and $A_2, \ldots, A_n$ must be incompatible with the synthesis tree (i.e. broken taxa). We then have that A is placed successively within $B_m, B_{m-1}, \ldots, B_2, B_1$, and m is the depth of the placement of A.

3 Synthesis and conflict resolution with incertae sedis taxa

3.1 Placement causes broken taxa

Synthesis with *incertae sedis* taxa has the potential to resolve uncertain taxon placements using information from phylogenies. The propinquity pipeline has always been able to do this kind of resolution. However, without special consideration given to *naming*, placing a taxon A within a taxon B results in conflict with taxon B in the taxonomy. In order to avoid a situation where input phylogenies conflict with a large number of taxa when *incertae sedis* taxa are placed within them, we have filtered *incertae sedis* taxa from the taxonomy when constructing all prior synthesis trees.

When the synthesis tree conflicts with a taxonomy node, we way that the taxon B at that node is a broken taxon. Broken taxa have two main effects, both of which are negative. First, the name B of the broken taxon is removed from the synthesis tree. Second, the conflicting edge for taxon B is not included in the synthesis tree. This means that any children of B that are taxonomy-only will not be placed with the children of B that are mentioned in input phylogenies. Instead the taxonomy-only children of a broken taxon move towards the root of the synthesis tree and attach at the first higher-ranked taxon that is an ancestor of B but is not broken.

3.2 Correctly handling placement

Thus, we seek a synthesis method that can correctly place taxa within containing taxa without breaking the containing taxa. We change the semantics of names to imply, not the exclusion of all non-included taxa, but only some non-included taxa, as described in section 2. As a result of this change in semantics, placing an IS taxon A within a sister taxon B no longer results in conflict with B. This allows us to retain the split for B within the synthesis tree, so that taxonomy-only children of B are correctly grouped with their siblings that are referenced by the input trees. We may then retain the name for the no-longer-broken taxon. Finally, we are then able to stop filtering *incertae sedis* taxa, so that they appear in the synthesis tree. Thus, the synthesis tree is able to represent substantially more species, without suffering the loss of taxa and the loss of structure.

3.3 Conflict with incertae sedis taxa

3.3.1 Conflicting placement among input trees

The addition of *incertae sedis* taxa allows new types of conflict between input trees. For example, different input trees might place an incertae sedis taxon in conflicting locations. This is illustrated in Figure 7, where the IS taxon (ott7,ott6)ott10 is placed as sister to ott1 by phylogeny τ_1 and as sister ott3 by phylogeny τ_2.

When this happens, the placement of the IS taxon is not influenced by its being marked IS on the taxonomy. Thus, in Example 1, the higher ranked tree τ_1 will be reflected in the synthesis tree, ott10 will be placed as sister to ott1. In contrast, the conflicting placement in τ_2 will not be reflected in the synth tree.

All this would occur in the previous version of propinquity. Where the updated version differs that (a) the names ott9 and ott8 are retained instead of being dropped. (b) as a result of not breaking ott8 and ott9, we do not move ott3 and ott2 up to ott11. [BDR: Extend more figures!]

3.3.2 Example B

However, incertae sedis taxa are not always tip nodes, but may themselves contain other taxa. In such cases, it is possible for input trees to conflict with the incertae sedis taxon itself. For example, consider the following example

- $T_1 = ((w_1, w_2, z_1), y_1)$
- $T_2 = ((y_1, y_2, z_2), w_1)$
- Taxonomy tree $((w_1, w_2)w, (y_1, y_2)y, (z_1, z_2, z_3)z)$ with z marked *incertae sedis.*
  - splits $w_1 w_2 | \bullet\, y_1 y_2$, $y_1 y_2 | \bullet\, w_1 w_2$ and $z_1 z_2 | \bullet\, w_1 w_2 y_1 y_2$

In this case, tree T_1 places z_1 within w, while T_2 places z_2 within y. Since different members of z are placed, neither placement for z is rejected. Instead the taxon z is broken, the name z disappears, and z_3 floats to the top level.

Furthermore, since taxon z is a broken *incertae sedis* taxon, all of its children are effectively incertae sedis independently, with the difference that they cannot be placed within each other. Therefore, the names w and y are not lost, since the taxa placed within them are *incertae sedis* names. Note that z_3 maybe therefore be placed in a third location, since it is also *incertae sedis.*

The situation here would be different if the monophyly of z was supported by an input phylogeny.

Note that the synthesis of all input trees before the taxonomy is unaffected by incertae sedis information.

3.3.3 Example C - Nested *incertae sedis* taxa

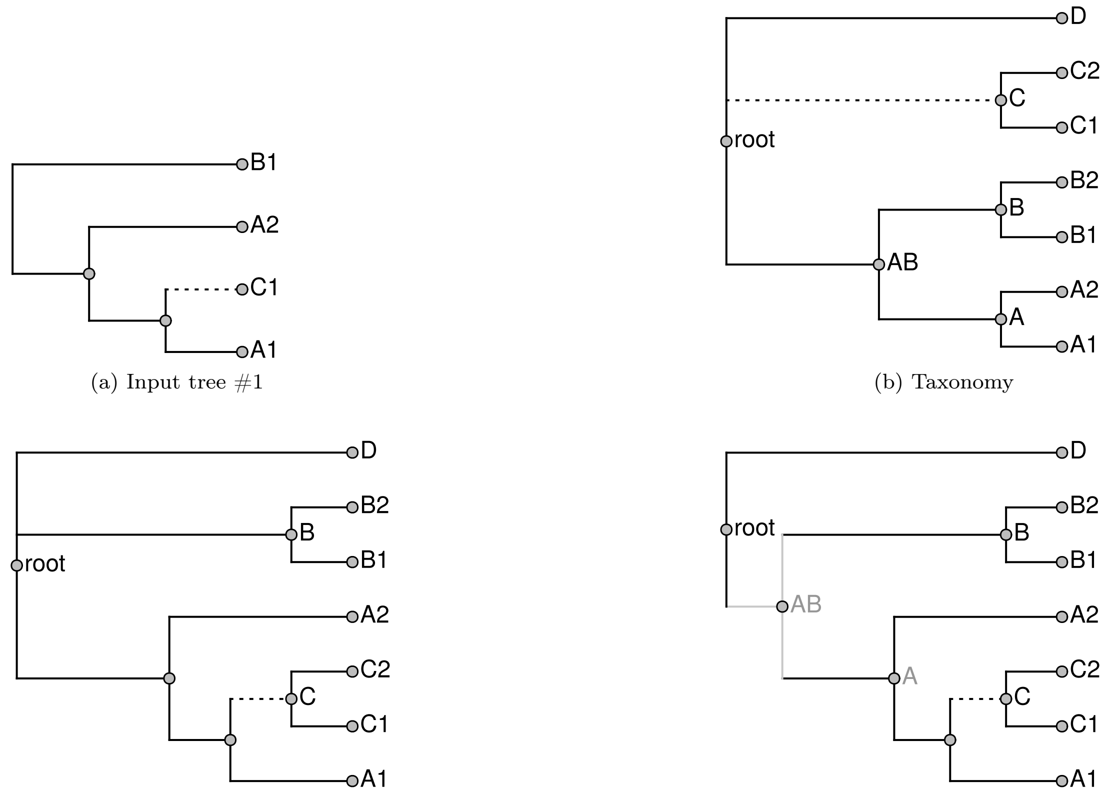
In addition to incertae sedis taxa containing other taxa, it is also possible for incertae sedis taxa to contain nested *incertae sedis* taxa. **What issues might this raise?**

4 Cases

When only tip nodes on the taxonomy are *incertae sedis*, we have a very simple form of the *incertae sedis* supertree problem. However, in practice entire clades may be *incertae sedis*, and so more complex issue arise. In this section we consider a number of cases that must be handled when attempting to place *incertae sedis* taxa.

4.1 Case 1: An *incertae sedis* clade

When a clade is marked as *incertae sedis*, we need to consider two cases. In the first case, the clade may be placed within a sister clade intact (Figure 3). However, if the clade is not monophyletic in the synthesis tree, then we allow the members of the clade to be placed separately (Figure 4).



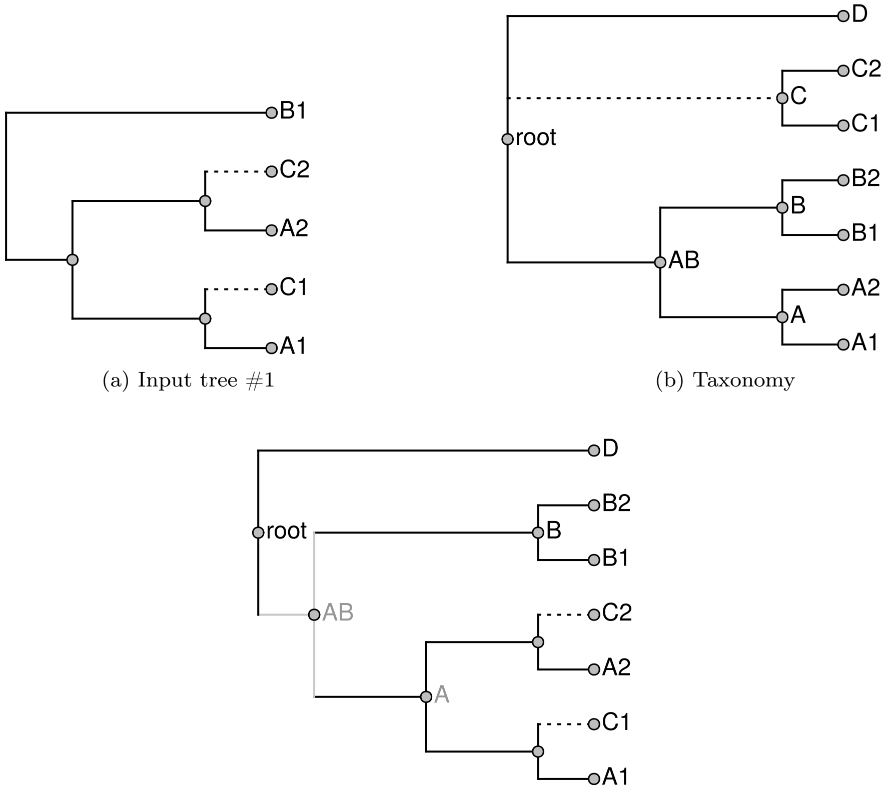
(a) Input tree #1

(b) Taxonomy

(c) Synthesis Tree #1

(d) Synthesis Tree #1

Figure 3: Handling *incertae sedis* taxa recovers additional edges and taxon names. Supertree construction on input tree (a) and taxonomy tree (b) with *incertae sedis* clade C leads to synthesis tree (c). However, supertree construction with *incertae sedis* handling constructs tree (d), which recovers taxon name AB and A, as well as the edge to clade AB. Taxon names and edges that are conditional on handling *incertae sedis* are shaded grey.



(a) Input tree #1

(b) Taxonomy

(c) Synthesis Tree #1

Figure 4: Broken incertae sedis clade. Taxonomy tree (b) contains *incertae sedis* taxon C that is broken by input tree (a). Supertree construction results in a synthesis tree (c) that places $C1$ and $C2$ separately within A.

4.2 Case 2: A nested *incertae sedis* clade

When *incertae sedis* clades are allowed, it is possible for an *incertae sedis* taxon to be nested within another *incertae sedis* taxon. In such a case we must allow the higher-ranked taxon to be placed within a sibling, while preserving the right of the lower ranked taxon to be placed within a lower-level sibling (Figure 5).

4.3 Case 3: Placement of one incertae sedis taxon within another.

In our semantics of incertae sedis taxa, it is possible for to place one *incertae sedis* sibling inside another. A special case of this is when a higher-rank incertae sedis taxon A is placed as sister to an incertae sedis taxon B, and then B is placed within A (Figure #).

This case makes placement more complicated. Use a sequence of ordered placements, preorder?

5 Handling *incertae sedis* taxa in the propinquity pipeline

In order to handle *incertae sedis* taxa within propinquity, we must modify some of the stages of the propinquity pipeline. Subproblem decomposition must place *incertae sedis* taxa in the correct subproblem. Subproblem files must indicate which taxa are incertae sedis. The subproblem solver must read this information, account for *incertae sedis* taxa when solving subproblems, and correctly name taxa that have been modified by having *incertae sedis* taxa place inside them. The unpruner must be aware of *incertae sedis* taxa. Annotations of the tree must be aware of *incertae sedis* taxa so that it does not consider taxa broken when they have an incertae sedis taxon placed inside them.

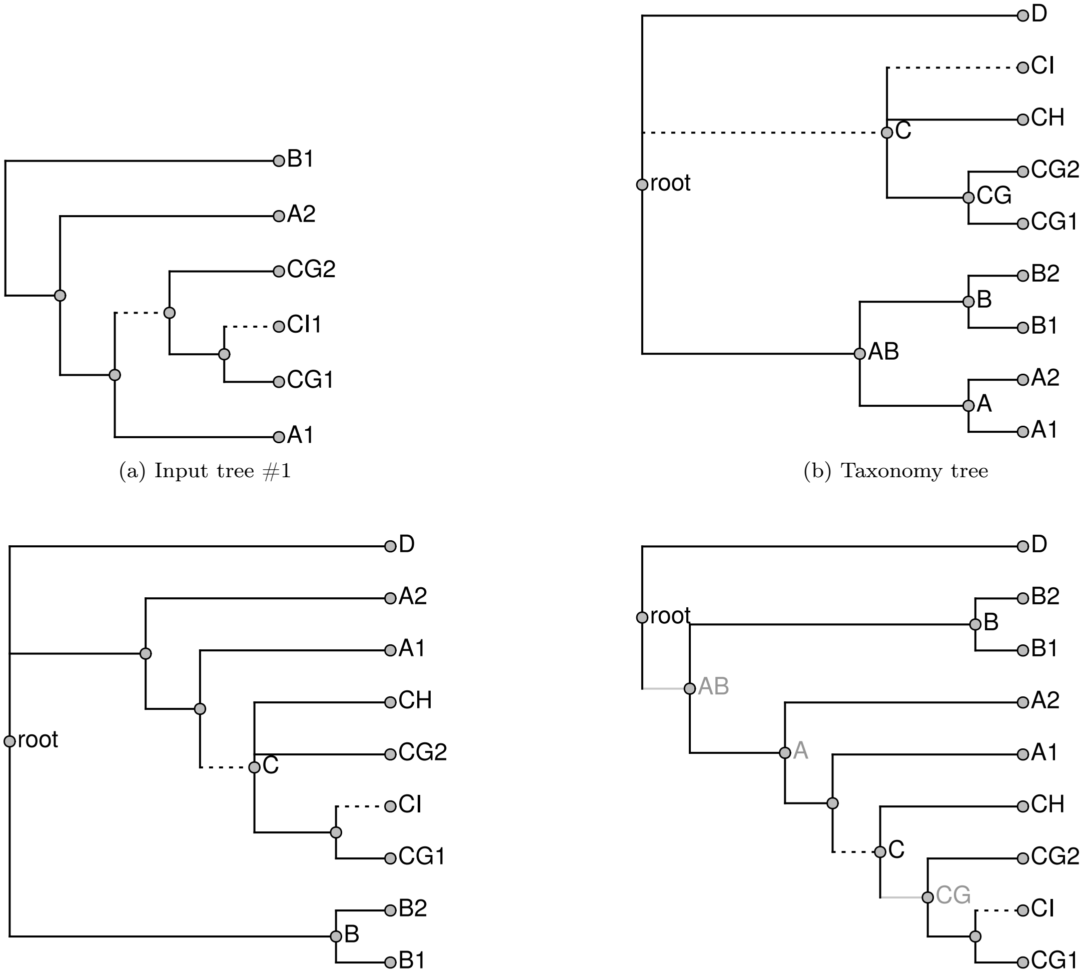
5.1 Exemplifying taxa

One current problem is that well-known taxa like Fungi or Mammalia tend to have a very large number of incertae sedis children, making browsing in the tree viewer difficult. This can happen when, for example, fossils or other hard-to-place taxa get classified only to the level of these well-known nodes and no further. This leads to a situation where well-known taxa serve as a dumping ground for unplaced taxa.

Our current approach to this problem is to perform a second round of pruning, or "cleaning", during the exemplification step. Incertae sedis taxa are pruned at this stage if they do not occur in any input trees. We thus generate a second "cleaned taxonomy" that has undergone this further round of cleaning. This approach improves on the previous approach in that *incertae sedis* taxa in input trees are no longer pruned. This approach also removes tons of *incertae sedis* children from nodes like "Fungi", where a lot of unplaced fossils with few observable characters have been dumped.

However, this approach has the negative effect of pruning some incertae sedis taxa that need not be pruned. For example, suppose *incertae sedis* taxon A contains 5 children, of which only 1 child A_1 occurs in an input tree. If the taxon A is not broken, then it should be possible to attach the other 4 members of A next to A_1, without cluttering up the synthesis tree. Such taxa have been successfully placed even though they are not in any input tree. This can only be discovered after synthesis is complete, though.

Additionally, it should also be possible to filter unplaced taxa in the tree viewer instead of in the synthesis pipeline.



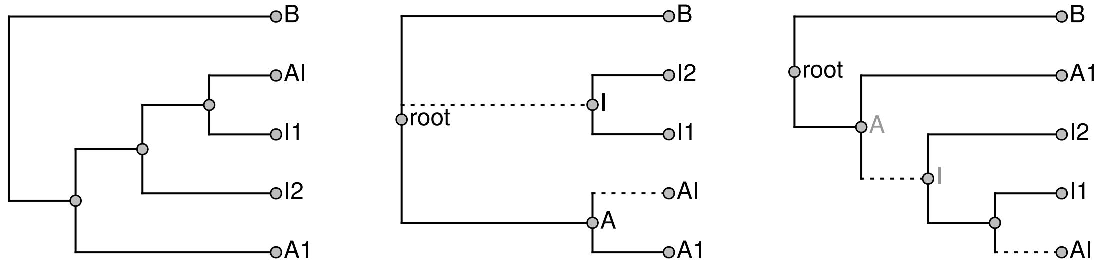
(a) Input tree #1

(b) Taxonomy tree

(c) Synthesis tree (*incertae sedis* ignored)

(d) Synthesis tree (*incertae sedis* considered)

Figure 5: Nested incertae sedis taxa. (a) Input tree #1 places CI1 within CG, and C within *A*. (b) In the taxonomy, CI is *incertae sedis* within C, while C is *incertae sedis* under the root. (c) The synthesis tree when *incertae sedis* taxa are not considered. The names AB, A, and CG are lost. (d) The synthesis tree when *incertae sedis* taxa are considered. The names AB, A, and CG are regained, as well as the edges leading to AB and CG.



(a) Input tree #1

(b) Taxonomy

(c) Synthesis tree

Figure 6: Placing one incertae sedis group within another. Here I is placed within A, and then AI is placed within A.

5.2 Sub-problem decomposition

The presence of *incertae sedis* taxa poses a problem to sub-problem decomposition, since taxonomy edges no longer completely separate subproblems. Instead, *incertae sedis* taxa may attach on either side of a taxonomy edge. We seek to place *incertae sedis* taxa into subproblems in such a way that the subproblem solver can perform the placement inside the subproblem. This approach postpones handling of conflict in *incertae sedis* taxa to the subproblem solver, where the problem is well formulated in terms of splits. However, it does have the effect of creating larger subproblems.

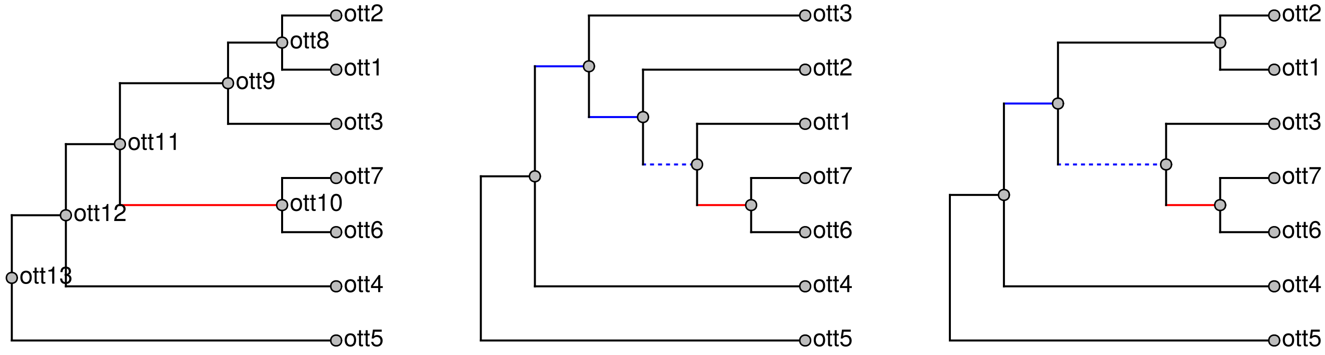
We must also handle conflicting placements of *incertae sedis* taxa by different input trees. Thus, if one input tree places the *incertae sedis* taxon X in $((X)B)A$ and another places X in $((X)C)A$ then we must mark both edges B and C as contested edges, even if these edges would *not* be contested were taxon X to be removed. This results in a new way to contest edges that involves the interaction of two input trees, and not just the interaction of each input tree with the taxonomy.

We choose to solve these problems by merging any subproblems that an *incertae sedis* taxon might be placed in. The simplest way to achieve this is simply to regard any taxon that has an *incertae sedis* taxon placed within it as contested. This results in marking both B and C as contested edges in the example above. In fact, this is the current behaviour of the non-*incertae-sedis* aware subproblem decomposer. One downside of this approach is that, if we have $((X)B)A)$ in one input tree, and X is mentioned nowhere else, then by marking B as contested, we are merging subproblems unnecessarily. We could instead placed X in B and avoid contesting the edge B. However, this approach is more complex and does not seem necessary in practice.

For example, in Figure 7, input tree τ_1 contests ott9 and input tree τ_2 contests ott9 and ott8. Thus ott1, ott2, ott3, ott6, and ott7 end up in the same sub-problem.

5.3 Subproblem solution

Our sub-problem solver naturally handles *incertae sedis* taxa. This is because we define the semantics of *incertae sedis* taxa in terms of partial splits, and our solver natively supports building trees from partial splits through its use of the BUILD algorithm. Handling *incertae sedis* taxa thus requires loading incertae sedis information and computing partial splits for *incertae sedis* taxa before solving a sub-problem. After solving a sub-problem, we must apply taxon names from the taxonomy tree to the sub-problem solution tree. The solution tree is considered to a fixed tree and not to have any *incertae sedis* nodes, or any other forms of uncertainty.



(a) Taxonomy $\mathbb{T}$ (b) τ_1 (c) τ_2

Figure 7: Example. An incertae sedis clade (ott6,ott7) is placed in different subtrees by input trees τ_1 and τ_2. In τ_1, two nodes that correspond to the taxonomy their ingroup extended to include (ott6,ott7), and the branches leading to these nodes have been colored blue. The dashed blue edge leads to a node that is a newly-introduced degree-2 node which does not correspond to any taxonomy node. In τ_2, only one node that corresponds to a taxonomy node needs to have its ingroup extended. The placement of (ott6,ott7) into ott8 toward ott1 by τ_1 conflicts with the placement of (ott6,ott7) into ott9 toward ott3 by τ_2,

5.3.1 Reading incertae sedis information

Currently, we read the *incertae sedis* information as a list of OTT ids for *incertae sedis* taxa. This does not require adding further annotations to the node names. Only taxonomy nodes can be *incertae sedis* at the moment, and only the taxonomy tree for the subproblem contains OTT ids for internal nodes. Therefore we handle *incertae sedis* information by constructing modified split sets for the lowest-ranked tree when the list of *incertae sedis* nodes is not empty.

5.3.2 Exclude sets modified by *incertae sedis* marks

Equation (1) leads to the following algorithm to compute the exclude set for all nodes in a tree.

1. Set the exclude set of the root node to be empty
2. For each *node* (except the root) in preorder
   - combine the *exclude* set of the parent node with the *include* set of non-*incertae-sedis* siblings.
   - store this set in a hash, with key *node*

This algorithm is currently implemented in *otc-solve-subproblem*. We store the sets as *std::set*.

5.3.3 Implementation: finding the node for a name

To find the node for a name n, we find the MRCA of the cluster $S_1(n)$. If the MRCA excludes the entire exclude group $S_2(n)$ then the name applies to the MRCA; otherwise the taxon does not exist on the tree.

5.3.4 Implementation: handling name clashes

When multiple names $N = \{n_1, \ldots n_N\}$ map to the same solution node x, then these names must satisfy some tree structure on the taxonomy, such that $n_1 < n_2$ if n_1 is a descendant of n_2 in the taxonomy. If it is possible to find a name n_{max} that is the unique maximal element of N, then it is permissible to

1. create a monotypic parent $p(x)$ of x, and assign n_{max} to $p(x)$
2. continue handling name clashes at x with the set of possible names reduced to $N - n_{max}$.

However, its certainly possible that there might not be any such N_{max}, in which case we could just choose a name for x from N (perhaps not an *incertae sedis* name) and then record all the other names as equivalents somewhere.

BDR: *we might get this behavior in a nice an automatic way if we create a single fake leaf for each monotypic taxonomy node that holds the node's leaf label.*

5.3.5 Caveats

When multiple I.S. taxa have been moved to the root node of a subproblem, they may be I.S. over the entire subproblem, and some may be I.S. over others in an asymmetric manner. Therefore, we might need to specify additional information about the original attachment location of the I.S. taxa, such as their depth. This only affects problems that have been decomposed.

BDR: *currently we don't actually move taxa to get them into a subproblem. So, is this even an issue?*

5.4 Grafted supertree

Question: Does the synthesis tree contain any *incertae sedis* groups?
Answer: The grafted supertree will not contain any *incertae sedis* groups. However, when we attach pruned nodes to a parent in the grafted supertree, we could mark such nodes *incertae sedis* if we want.

5.5 Unpruning

Currently the unpruner *does not* require that the OTT ids are named in the grafted solution before unpruning starts. According to Mark's document, he wasn't sure if such names were generated for nodes that had an IS taxon placed inside of them, so otc-unprune-solution-and-name-unnamed nodes throws away all the names and generates them itself.

BDR: See document `otcetera/doc/unprune-solution-and-name-unnamed-nodes.pdf`

The unpruner should record when unpruned nodes are *incertae sedis*. Such nodes are unaffected by phylogenies, and so *incertae sedis* annotations for them make good sense.

5.6 Annotation

Annotation primarily involves running a conflict analysis between the synthesis tree and each input tree. Since neither tree has any *incertae sedis* taxa, the conflict algorithm does not need to change. Furthermore, if we allow *incertae sedis* taxa that are taxonomy-only to be annotated as *incertae sedis* on the synth tree, then such groups will not affect conflict with the input trees. We would also like to allow running a conflict analysis between the synthesis tree and the taxonomy tree. However, naming the nodes *is* a (almost) run of conflict analysis on the taxonomy tree, and this has already been done in a prior step. So, the current annotation procedure actually works as-is.

It would be nice to allow running conflict against the cleaned taxonomy, though. One way to do this would be to generated a "placed taxonomy", with groups extended to include *incertae sedis* taxa that have been placed within them. This would not require any updating to the conflict-analysis code in the annotation step.

5.7 Conflict service

The current conflict service considers a group A to conflict with the taxonomy if group A has an incertae sedis group B placed within it. This doesn't affect the annotations, since taxon names are added by the unpruner. But it could make perfectly fine input trees incorrectly look like they are the cause of broken taxa, if they contain IS taxa. Thus, it would be nice to have a modified conflict algorithm.

5.7.1 Current conflict algorithm

The current conflict algorithm is pretty fast, but it works by classifying tips into either (i) the include group or (ii) the exclude group. To avoid counting the exclude group for every split, we instead count the total number of children for each node, and assume that any children not in the include set are in the exclude set. This is no longer true when we have incertae sedis taxa. I suspect that if we want to handle incertae sedies, we'd need a third category (iii) for "neither include group nor exclude group".

1. Get induced trees on intersection of leaf sets
2. Compute depth for each node (nd->depth)
3. Compute number of tips at or below each node (nd->n_tips)
4. for each input tree node -> *nd*
    (a) skip the root
    (b) skip monotypic
    (c) if its a tip then find corresponding ("terminal") edges in synth tree and continue
    (d) leaves1 <- get the list of leaves in the include group of *nd* (in input)
    (e) L2 <- find the total number of tips (L2 = sum [nd->n_tips| nd <- leaves1])
    (f) leaves2 <- get list of corresponding synth leaf nodes (in synth)
    (g) nodes <- find all nodes between leaves2 and the MRCA (in synth)
    (h) MRCA <- mrca of leaves2 (in synth. this uses the nd->depth annotation)
    (i) Compute number of tips in the include set (nd->include_tips) below each node in *nodes* (in synth)
    (j) if n_include_tips(MRCA) == n_tips(MRCA) then the MRCA displays *nd*
    (k) if n_include_tips(MRCA) < n_tips(MRCA) then
        - foreach node in nodes
            - if (n_include_tipes(nd) < n_tips(nd) and n_include_tips(nd) < l2)
                - this is a conflict!
        - if there are no conflicts, then this is a resolved_by.

5.7.2 Modified conflict algorithm?

This probably is outside the scope of the paper, but if we could come up with a modified conflict algorithm, that would be nice/useful, and probably novel. It would also probably be slower...

6 Results

6.1 Case 1

taxonomy = (((a1,a2)A,(b1,b2)B)AB,(c1,c2)?C,D)root;

6.1.1 If we place *c* within *A*

tree1 = (((a1,c1),b1),d1);

synth-with-is: ((((a1,(c1,c2)C),a2)A,(b1,b2)B)AB),(d1)D)

placement: C within A <- AB

synth-no-is: ((a1,(c1,c2)C),a1,(b1,b2)B,(d1)D)

6.1.2 If we place *c* within *A* but break *C*

tree1 = ((a1,c1),(a2,c1),b1)

synth-with-is: ((((a1,c1),(a2,c2))A,(b1,b2)B)AB, (d1)D)

synth-no-is: ??

placement:

- c1 <- (broken) C within A <- AB.
- c2<- (broken) C within A <- AB

result: we lose *C*, but keep *A* and *AB*.

6.2 Case 2: nested is

taxonomy = (((a1,a2)A,(b1,b2)B),(((c11,c12)C1,(c21,c22)C2,(ci1,ci2)?CI)?C) ,(d1)D)

tree1 = ((a1,(c11,(ci1,c12))),b1)

synth-with-is: places CI wihin C1, and C within A.

6.3 Case 3: interleaving taxa

taxonomy = ((a1,a2)A,(b1,b2)?B,(c1,c2)?c)

input tree: ((b1,c1),(b2,c2))

synth-with-is: ((a1,a2)A,((b1,c1),(b2,c2))B) or ((a1,a2)A,((b1,c1),(b2,c2))C)

Result: we name the interleaved taxon B or C, but cannot name it both names.

6.4 Case 4: creation of monotypic taxa

see test cases.

Should we do this? We could say:

- there are 331 incertae sedis taxa that are mentioned in input trees.
  - what is the kingom / phylum / class / order / genus / species
- we placed 14 incertae sedis taxa inside a sister taxon. (“incertae sedis”, “versus “unplaced”?)
- we placed o incertae sedis taxa *outside* all sister taxa.
- we confirmed n incertae sedis taxa as being separate from all sister (sampled) taxa??
- we avoiding breaking 8 taxa that had IS taxa placed inside them.
- we allowed z_1 new taxa into the synthesis tree that were incertae sedis.
- we allowed z_2 new taxa into the synthesis tree that are marked as extinct.
- some nodes have as many as w *incertae sedis* children, making them unbrowseable when incertae sedis children are not excluded.
- v_1 input trees were previously excluded *entirely* because they are nested within in an incertae sedis taxon.
- v_2 input trees were previously excluded *partially* because they are nested within in an incertae sedis taxon.

Currently the numbers z_1 and z_2.

7 Discussion

One thing we could do (perhaps) that we are not currently doing, is to have nodes marked as incertae sedis on the synth tree. This would be easy enough if such nodes are not affected in any way by the input trees. Thus, when unpruning nodes we could mark any nodes *incertae sedis* if they were marked *incertae sedis* on the taxonomy.

Secondly, I think we need to distinguish *incertae sedis* taxa that are “unplaced” from *incertae sedis* taxa that do not occur in any input tree. I think that if A contains child A_1 that is an input tree, and the taxon A is not broken, then A will be placed, and thus any other children $A_2, A_3, \ldots, A_n$ will also be placed, since they will be added as children of the (placed) node A by the unpruner. This could be considered when deciding which nodes to suppress in the tree viewer.

Taxonomy merging Not all *incertae sedis* taxa in our taxonomy are directly labeled *incertae sedis* by taxonomists. *Incertae sedis* taxa can also result from automatic merging of taxonomies to create the OpenTree taxonomy. For example, in Figure 3 of Rees and Cranston (2017), cases #4 and #6 illustrate examples where merging of two taxonomies leads to a taxonomy with a taxon of uncertain placement. The reason is primarily that if taxonomy $\mathbb{T}_1$ contains more levels of hierarchy than taxonomy $\mathbb{T}_2$, then we must add internal nodes to $\mathbb{T}_2$ to align it to $\mathbb{T}_1$. However, if taxonomy $\mathbb{T}_2$ contains more leaves than $\mathbb{T}_1$, then it is unclear if these extra leaves should be nested inside the additional internal nodes, or not. Thus, the extra leaves are marked *incertae sedis.*

For example, if $\mathbb{T}_1 = ((a, b)x, (c, d)y)z$ and $\mathbb{T}_2 = (a, b, c, d, e)z$ then a and b in $\mathbb{T}_2$ should be nested within x, but we do not know if e should be nested within x or not. Thus we obtain $((a, b)x, (c, d)y, ?e)z$, where ? indicates that taxon e is marked *incertae sedis.*

If incertae sedis taxa are cannot be placed within other incertae sedis taxa, then such an approach does not work if y is already incertae sedis. However, we take the approach that *incertae sedis* notations are not a general solution to expression of uncertain placement.

7.0.1 Comparison to operational definition

The splits-based semantics for incertae sedis taxa has many desirable properties. In allows the use of the BUILD algorithm to assess compatibility of split sets. However, it also has some properties that may not be expected. For example, two incertae sedis siblings can be freely interdigitated, which might not be the expected outcome. The current definition, also ignore ranks. Thus if we had two incertae sedis siblings that were genera, but other siblings were families, then one might expect that the genus-level incertae sedis taxa could be placed within families, but not intermixed. Since the current approach ignores ranks, it cannot do that.

The above definition of incertae sedis taxa does allow two incertae sedis sister taxa to be interdigitated. It also

Also, the splits-based semantics might not completely recover the edit-operation semantics. The corner cases are ... .

7.0.2 Intrusion into incertae sedis taxa

In theory it could be possible for an *incertae sedis* taxon A to be placed within an *incertae sedis* sibling B. We initially implemented this semantics. It requires updating the recursion to define $I(m, n)$ as the descendants of m that can be access from n without traversing an *incertae sedis* node.

With a splits-based semantics, we cannot allow both (i) placing A in B and (ii) placing B in A and also forbid interdigitating A and B.

However, if we allow this behavior by reducing the exclude sets for A and B, then this has the unfortunate consequence that the members of A and B may be inter-digitated. In many cases, *incertae sedis* genera have siblings that are families, and the expected semantics is that genera may be placed within their sibling families, or may remain outside existing families. In such cases, inter-digitating the genera would not be expected behavior. Thus, we choose the semantics that *incertae sedis* taxa exclude their *incertae sedis* siblings. This is not because the other semantics is impossible. However, we note that use of the interdigitating semantics allows a situation where the names for A and B both apply to the same node.

This semantics also complicates placement, since it is possible to place A as a sibling of B, and then place B within A. This violates the current invariant that taxa are only ever placed within more tipward taxa. This makes the description of placement more complicated, since we can no longer simply record all cases where a parent node is not ancestor on the taxonomy.

7.0.3 Usage in the literature

The term *incertae sedis* can be used in a number of different ways. For example, one author might describe *incertae sedis* genera that are asserted to be monophyletic, whereas another author might describe *incertae sedis* genera that may be interdigitated.

References

Benjamin D Redelings and Mark T Holder. *A supertree pipeline for summarizing phylogenetic and taxonomic information for millions of species*. PeerJ, 5:e3058, 2017.

Jonathan A Rees and Karen Cranston. *Automated assembly of a reference taxonomy for phylogenetic data synthesis*. Biodiversity data journal, (5), 2017.