Handling *incertae sedis* taxa in propinquity

February 6, 2018

1 Introduction

The phrase "*incertae sedis*", which is a Latin phrase meaning "uncertain seat", is used in classifications to indicate that a group is of uncertain taxonomic placement. Such groups arise in practice when they are classified within a parent group, such as Fungi, but the precise location within the parent group is uncertain. Such a taxon would then be attached as a child of the parent group, but labeled as *incertae sedis*".

Currently *incertae sedis* taxa and their descendants are excluded from the OpenTree synthetic tree. This is problematic, because the OpenTree project seeks to build a comprehensive tree covering all of life. However, including unplaced taxa within the current synthesis framework would be equally problematic, since placement of taxa within groups is currently considered to be a conflict with those groups. This conflict causes both a loss of the taxon name for the conflicting taxon, and a loss of structure in the synthesis tree.

1.1 Some examples in OTT[1]

The problem is becoming more acute because NCBI is putting a larger number of taxa into groups that are marked as "unclassified." For example, when OTT 2.9 was created NCBI's clasification of the bird family Sylvidae included a group five genera that were placed within "unclassified Sylviidae." This includes the genus Regulus. Thus in OTT 2.9 Regulus (ott ID = 3599326) is placed inside the family Silvidae (OTT ID = 259942). The "unclassified Sylviidae" does not appear in OTT; instead Regulus is flagged as "unclassified,sibling higher" and all of the species within Regulus are flagged as "unclassified inherited."

The taxonomy (as of version 2.9) also contains 685 cases of taxa that are flagged as both "unclassified inherited" and "unclassified"[2].

We would like to stop suppressing (pruning) incertae sedis taxa, so that groups such as Regulus can appear appear in the synthetic tree.

1.2 Handling incertae sedis taxa

To solve this problem, we describe an extension of our previous supertree synthesis procedure that can use phylogeny information to place *incertae sedis* taxa into containing taxa without considering the containing taxa to be broken. This extension requires that we first describe an extended semantics for taxa in the presences of *incertae sedis*, which is not a trivial problem. We describe both an updated semantics for conflict with taxonomy edges, and an updated semantics for assigning taxonomy names to nodes in the synthetic tree.

The ability to place taxa within the taxonomy without breaking their containing taxa enables us to include thousands of new taxa that were previously filtered out to avoid broken taxa. It also enables us to use phylogenetic trees to update the taxonomy by extending taxa to include *incertae sedis* taxa placed within them. This makes substantial progress towards our goal of complete representation of taxa. It also enables us to include extinct taxa, since many of these taxa are incertae sedis.

[1]This paragraph pulled from intro to otcetera/doc/handling-incertae-sedis.pdf

[2]based on `grep unclass.*unclassified\_inherited taxonomy.tsv`

1.3 Incertae sedis taxa in taxonomies

Flags. The Open Tree's reference taxonomy is produced by a tool called smasher that notices hints that a taxon is incertae sedis and labels that taxon with one of five flags[3]. While a taxonomist may intend to use incertae sedis to indicate a limited number of possible positions for a taxon to go in the taxonomy, we do not retain any such details.

Container nodes. Some taxonomies have nodes with names like "Incertae sedis (Bacteria)". Such nodes are called containers. They indicate that each child node is an *incertae sedis* child of the container node's parent. The OTT retains container nodes with the *was_ container* flag, but seems to have already moved all of the container's children to the container's parent.

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.*

1.4 Formalism

We consider a ranked collection $\mathcal{T} = \{T_1, \ldots, T_n\}$ of input trees and a single taxonomy tree $\mathbb{T}$. We have a set of labels $\mathcal{L}$ that correspond to taxon names. Therefore taxonomy node has a label, and every label correspond to a unique taxonomy node.

In contrast, input phylogenies only have labels on their leaf nodes. Furthermore, and the set of labels $\mathcal{L}(i)$ on input tree T_i need not include all leaf labels, and in practice always contains only a relatively small part of $\mathcal{L}$.

We extend this framework by adding a set $\mathcal{I} \subseteq \mathcal{L}$ of labels that have the *incertae sedis* property. The taxonomy consists of the labels $\mathcal{L}$, the taxonomy tree $\mathbb{T}$, and the *incertae sedis* property $\mathcal{I}$ of taxon names.

2 Semantics of *incertae sedis* taxa

Incertae sedis taxa affect the synthesis procedure in two ways: they affect what it means for a taxonomy edge to conflict with the synthesis tree, and they affect what it means to place a taxon name on a particular node in the synthesis tree. We examine each of these effects in turn.

2.1 Splits

Each edge of a standard 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

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

with the exclude set always on the right. 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:

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

[3]see `https://github.com/OpenTreeOfLife/reference-taxonomy/wiki/Taxon-flags`.This document is not concerned with the flagging system *per se*, so "*incertae sedis*" will be used here to refer to all of the flags that denote taxa with uncertain placement.

For a node n on the tipward side of an edge e, we may also write $\mathcal{I}(n)$ for $\mathcal{I}(e)$, and $\mathcal{E}(n)$ for $\mathcal{E}(e)$. We consider the exclude set of the root node to be empty, and the include set of the root node to contain all tip taxa:

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

In this case the root node stands for an edge that connects the root to the root's parent.

2.1.1 Reduced exclude sets for *incertae sedis* taxa

Marking a taxon as *incertae sedis* changes the meaning of its sibling taxa. Since the *incertae sedis* taxon can be placed inside its sibling subtrees, the exclude sets of the sibling subtrees should not include the *incertae sedis* taxon. Although exclude sets of sibling taxa are shrunk, the include sets of the sibling taxa are unaffected.

An *incertae sedis* taxon can be moved into any of the descendants of its siblings. Therefore, the exclude $\mathcal{E}(n)$ set for a node n contains the children of all the ancestors of n, unless those children are incertae sedis. We can compute exclude sets recursively by writing the exclude set of a node n in terms of the exclude set of its parent. If we use the terminology that the include and exclude sets for a node n are $\mathcal{I}(n)$ and $\mathcal{E}(n)$, then we have

$$\mathcal{E}(n) = \mathcal{E}(parent(n)) \cup \left[\mathcal{I}(m) \middle| m \in siblings(n), m \text{ not } incertae\ sedis\right], \tag{1}$$

This leads to a pre-order recursion that terminates if the exclude set for the root note is set to $\{\}$ as a boundary condition.

2.1.2 Alternative formulation

Consider an I.S. clade A that is more rootward than an I.S. clade B. With the above formulation, we could not place A as sister to B and then place B within A, because A excludes B. To solve this problem, we could modify formula (1) above to avoid excluding `incertae_sedis_inherited` tips of a sister taxon m. Thus, we could modify formula (1) to refer to $\mathcal{I}'(m)$ instead of $\mathcal{I}(m)$, where

$$\mathcal{I}'(m) = \begin{cases} m & \text{if } m \text{ is a leaf} \\ \left[\mathcal{I}'(c) \middle| c \in children(m), c \text{ not } incertae\ sedis\right] & \text{otherwise.} \end{cases}$$

This would not rule out placing B as sister to A and then placing A within B.

2.2 Naming

After solving a supertree (sub) problem, we 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 $S(n) = S(n)_1|S(n)_2$ on the corresponding branch of the taxonomy tree. Without *incertae sedis*, such splits are always of the form $S(n)_1|\mathcal{L} - S(n)_1$, 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 we must raise the question of whether one name could apply to multiple nodes, or whether multiple names could apply to one node.

BDR: *It is well known that monotypic taxa are indistinguishable if you consider only splits on leaf labels, but are distinguishable if you consider splits on all node labels. It seems that some (all?) of the problems with assigning multiple names to the same nodes actually comes from the fact that moving IS taxa can leave the parent as a degree-2 node. The fact that propinquity handles monotypic taxa indicates that we actually implicitly consider all taxa to have labels. It is possible that a trivial extension to the sub-problem solved could thus handle monotypic taxa and issues with mapping 2 names to one node with incertae sedis taxa.*

2.2.1 Multiple nodes that fit one name

Suppose the taxonomy is (((A1,A2)A,B)AB,C,D*) and the solution tree is (((A1,A2)x,D*)y,B)AB,C). In this case the name A leads to the split S(A) = A1 A2 | B C root. This name can apply to both the node x and the node y.

Solution: In this case, we find the most tipward node and attach the name to this node.

2.2.2 Multiple names fit a single node

Example 1 Suppose the taxonomy is (((B1,B2)B,C*)A,Y) and the input tree is (((B1,C),B2)x,Y) then the names A and B both to the node x. In this case the names A and B are ordered.

Case 1: If a taxon contains 2 non-IS taxa, then it cannot be identical with any of its children in the synthesis tree.

Case 2: If a taxon contains 1 non-IS taxon and ≥ 1 IS taxa, then the taxon could be identical with is non-IS child in the synthesis tree, if the IS taxa are placed within the child.

Case 3: If a node contains 0 non-IS taxa and 1 IS taxon, then the IS taxa behaves no differently than a non-IS taxa, since is has no siblings it could be placed into.

Case 4: If a node contains 0 non-IS taxa and ≥ 2 IS taxa, then then taxon *could* be identical with a non-IS child in the synthesis tree, if all but one IS children are placed with in one of the IS children.

Solution: When we assign multiple names to the same node, then we expand the node with multiple names to have monotypic parents, and assign the series of names to the monotypic parents. Another way of saying this is that when a node has ≤ 1 non-IS taxon and ≥ 2 taxa then the node could become monotypic by placement of the IS taxa.

Example 2

If ((A1,A2)A,(B1,B2)B*,(C1,C2)C*); is the taxonomy with asterisks denoting incertae sedis taxa, then the solution ((A1,A2)A,((B1,C1)mrcaB1C1,(B2,C2)mrcaB2C2)x); has a node x that could be called B* or C*.

Case: If a taxa B* and C* are IS, then their tips can be intermingled in a new clade x. Both names would then apply.

Solution: ??

In summary, a taxon with split $A_1 | \bullet B_1$ attaches to the most tipward node n where $A_1 \subseteq S_1(n)$ and $B_1 \subseteq S_2(n)$. If multiple names end up on the same node, we try to resolve the problem by creating a monotypic node for a name that is more rootward than all other names at that node. If this fails to resolve the problem, we arbitrarily choose one of the names.

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 1, 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 $w_1w_2|\bullet y_1y_2$, $y_1y_2|\bullet w_1w_2$ and $z_1z_2|\bullet w_1w_2y_1y_2$

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?**

3.4 Placement

Placement of incertae sedis taxa by input trees is unfortunately not quite as simple as finding a single location where an I.S. taxon should attach. For example, when an incertae sedis taxon is broken, its children need to be "placed" separately.

Each input tree can relate to an *incertae sedis* taxon $(A, B, C)D$ in a number of ways

- it could resolve A, B, C, or D.
- it could place D on a degree-2 (=out-degree-1) node that bisects a branch
- it could place a descendant taxon of D
- it could place a descendant taxon of D in a *different place* than another input tree.
- it could place children of D in multiple places, thus conflicting with the branch. If the *incertae sedis* taxon (A, B, C) is broken, then A, B and C become *incertae sedis* clades in their own right, that may attach separately, except that they . This is because none of A, B, or C is in the exclude set of the siblings of D.

4 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.

4.1 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.

4.2 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

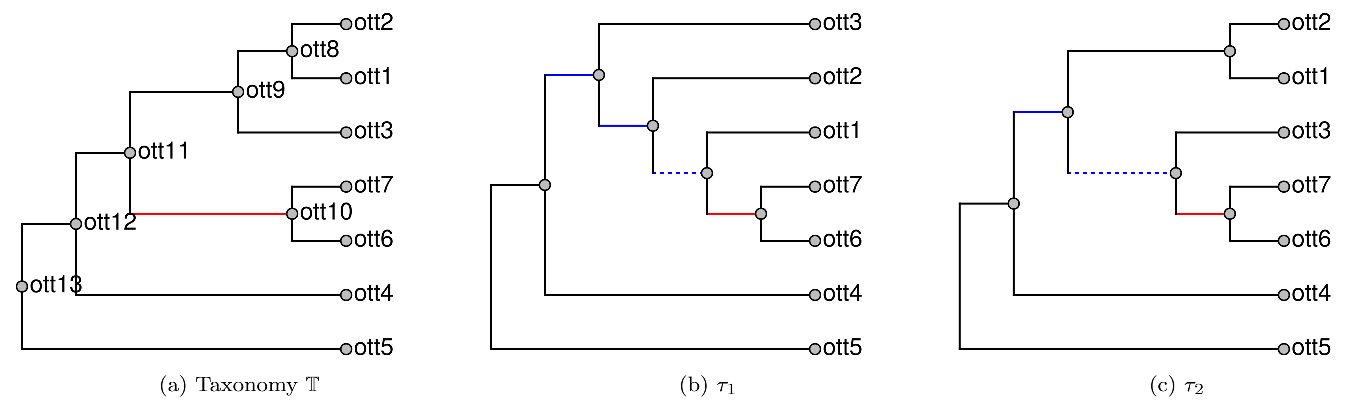

Figure 1: Example. An incertae sedis clade (ott6,ott7) is placed in different subtrees by input trees τ_1 and τ_2. In τ_1, two nodes that correspond to the taxonomy their ingroup extended to include (ott6,ott7), and the branches leading to these nodes have been colored blue. The dashed blue edge leads 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,

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 1, 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.

4.3 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.

4.3.1 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.

4.3.2 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*.

4.3.3 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.

4.3.4 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.*

4.3.5 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?*

4.4 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.

4.5 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.

4.6 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.

4.7 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.

4.7.1 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.
```

4.7.2 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...

5 Results

Should we do this? We could say:

- we placed x incertae sedis taxa.
- we avoiding breaking y 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 w were previously excluded *entirely* because they are nested within in an incertae sedis taxon.
- v_2 input trees w were previously excluded *partially* because they are nested within in an incertae sedis taxon.

Currently the numbers z_1 and z_2.

6 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.

References

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