Improvements to `propinquity`

November 30, 2016

1 Interpretation of *incertae sedis* taxa

1.1 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

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

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:

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

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.

1.2 Reduced exclude sets for *incertae sedis* taxa

A taxon that is marked *incertae sedis* is not excluded from its sibling subtrees. This changes the splits induced by each edge only by shrinking their exclude sets. The exclude set for a node is is the union of the exclude set of the parent node and the include sets of non-*incertae sedis* siblings. 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 valid pre-order recursion if $\mathcal{E}(root)$ is set to $\{\}$ as a boundary condition.

1.3 Containers

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.

2 Handling *incertae sedis* taxa

In order to handle *incertae sedis* taxa we must modify some of the stages (and maybe concepts) of the propinquity pipeline.

- **Subproblem decomposition** must be modified to place *incertae sedis* taxa in the correct subproblem, and to handle conflict in placement information between input trees.
- The **subproblem solver** has been modified to read *incertae sedis* information, solve problems which include that information, and name OTT nodes that have *incertae sedis* taxa newly placed within them.
- Q: does **unpruning** need to be modified, either in theory, or in terms of the code we currently have?

We will also generate another product, the *placed taxonomy.* The placed taxonomy will probably not have any *incertae sedis* taxa. Subproblem decomposition will place *incertae sedis* taxa up to sub-problem resolution, but will not resolve conflicting placements within subproblems. Conflicting placements withing subproblems will be resolved by the subproblem solver.

2.1 Subproblem solution

Subproblem solution does not need to be modified very much, since the BUILD algorithm already natively supports building trees from partial splits. Thus, we need to modify the subproblem solver to create the partial splits from the incertae sedis information.

2.1.1 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.

BDR: 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.

2.1.2 Exclude sets modified by *incertae sedis* marks

Equation (1) leads to the 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 is currently implemented in *otc-solve-subproblem.* We store the sets as *std::set.*

2.2 Decomposing into subproblems

Subproblem decomposition currently involves finding taxonomy edges that are consistent with every input tree, inserting these edges into each input tree, and dividing each input tree along that edge to create two subproblems.

If the taxon set of the input tree is entirely in the include set or entirely in the exclude set of the taxonomy edge, then the input tree will not be split, but will end up in one subproblem in its entirely. It will then not contribute to the other subproblem.

2.2.1 Uncontested edges in an *incertae sedis* world

Decomposition is based on the claims that

1. non-contested taxonomy edges will certainly be included in the final tree.
2. non-contested taxonomy edges can be inserted into each input tree, dividing it into two parts (one of which may be empty). We can then solve each of these parts separately and join them via the non-contested taxonomy edge.

The first claim is a heuristic assumption that is usually true. We may thus enforce this condition without (we hope) producing a tree that is very different from the tree we would get if we did not perform subproblem decomposition.

The second claim is true without *incertae sedis* taxa, but becomes untrue when we allow *incertae sedis* taxa, because taxonomy edges do not partition *incertae sedis* taxa. Thus they act as partitions up to, but not beyond, the issue of placement.

In order to use taxonomy edges to partition subproblems in the *incertae sedis* world, we must *extend* such taxonomy edges to partition all taxa by specifying which side of the edge each *incertae sedis* taxon is placed on. Furthermore, the collection of extended edges must be mutually consistent in order for them to be used jointly to define subproblems. Thus, in order to perform a partition in the *incertae sedis* world, we need a set of *extended* edges that are

1. uncontested (in their extended form)
2. mutually consistent (in their extended form).

While the original (unextended) collection of uncontested edges must be mutually compatible, it is possible for an extended edge to contradict another uncontested edge that groups two taxa A and B if the extended edge is extended to place A and B on different sides of the edge.

2.2.2 Simple approach

One approach would be to extend each taxonomy edge to be compatible with all input trees. Edges that cannot be compatibly extended are discarded, since the input trees must either conflict with the edge, or conflict in placement of taxa on each side of the edge. All extended edges that survive this thinning procedure would be checked for pairwise compatibility with all other extended edges, discarding any edge that is incompatible with another extended edge. The remaining edges may be few, but should be mutually consistent, and could be used to divide the taxonomy into subproblems.

2.2.3 Extending uncontested edges to place *incertae sedis* taxa

One approach would be to extend edges to place *incertae sedis* tips if all input trees agree on which side of the edge each tip should go on. If they do *not* agree, we would mark the edge as contested instead of extending it. By refusing to decompose along edges where different input trees disagree about placement, we would be punting the solution to conflicting placement to the subproblem solver instead of trying to solve it in the decomposer. This approach would probably work well if *incertae sedis* taxa can be placed independently.

This becomes more complicated when we consider that *incertae sedis* taxa may not be tip taxa. For example, we might have an *incertae sedis* clade (A, B) where A is placed on one side of an uncontested edge and B is placed on the other side of the edge. If these placements occur in a single tree, then that tree will conflict with the edge (A, B). However, if these placements occur in *different* trees, then the above procedure would extend the uncontested edge to conflict with the (A, B) edge. In that case, the taxonomy is ranked last, and we could simply mark the (A, B) edge as contested. When collapsing an edge to an *incertae sedis* taxon, we can preserve the correct information about taxon floating by marking all the children of the removed *incertae sedis* node as *incertae sedis*, except that this makes the children if *incertae sedis* taxa able to float into each other. Thus, contesting *incertae sedis* taxa makes this much more complicated...

Now, if the (A, B) grouping occurs in an input tree, then I think we will simply mark the edge that is extended to have A and B on different sides as contested. That would be best. Huh. This might actually work.

2.2.4 Example A

OK, so suppose we have

- cd|e, ac|d, bd|c, ab|cde
  - OK, so in this case ab|cde conflicts with a lot of edges. In synth, it would just be not included. In analysis for contesting, it seems like ac|d and bd|c should be contested.
- cd|e, ac|d, ab|cde, bd|c
  - OK, so in this case, it seems like bd|c contests ac|d and ab|cde
- cd|e, ab|cde, ac|d, bd|c
  - bd|c conflicts with abc|de

These cases seem complicated, but in fact in all cases the edge $cd|e$ is not really contested. It is just the *other* edges. And, in fact, we seem justified in concluding $abcd|e$ in all cases. This is an extension of $cd|e$ to place a and b tipward of it.

2.2.5 Example B

- Uncontested edge wx|y
- IS taxon vz|wxy
- tree 1 says vw|x
- tree 2 says xy|z
- optionally, tree 3 says vz|wxy

In this case we want to conclude that the edge $wx|y$ is contested by conflicting placement information. And I think we want to conclude that the IS taxon is also contested by placement information that would tend to break up the clade.

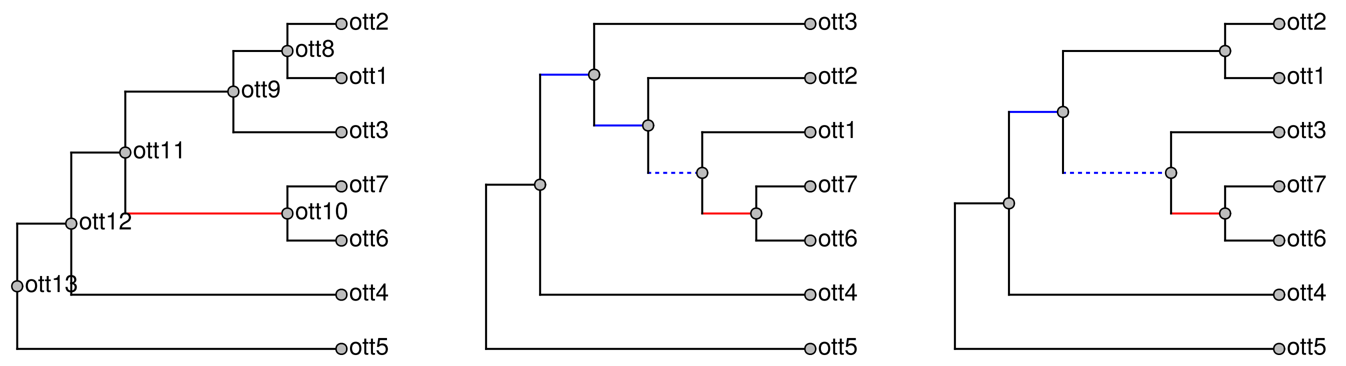
2.2.6 Example C - Nested *incertae sedis* taxa

We could have a clade that is *incertae sedis*. One of its descendant nodes could also be marked *incertae sedis*.

2.2.7 What to do

When *incertae sedis* taxa are involved, we must place *incertae sedis* taxa into subproblems in such a way that the subproblem solver can perform the placement inside the subproblem. However, we must also handle conflicting placements of *incertae sedis* taxa by different input trees.

I think we can solve this in a logically coherent way by recognizing that any taxonomy edges leading to conflicting placement of an *incertae sedis* taxon must end up in the same subproblem (i.e. they must be collapsed). Thus,



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

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,

if one input tree places the *incertae sedis* taxon X in $((X)B)A$ and another input phylogeny places *incertae sedis* taxon X in $((X)C)A$ then we must mark the 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.

The result of such a decomposition procedure would be that the subproblem solver will handle any conflicting placement information.

2.2.8 Example 1

In Figure 1, we can establish correspondences between branches in $\mathbb{T}|\tau_1$ and τ_1 as follows:

$$\begin{aligned}
\text{ott12: } 1\,2\,3\,4\,\mathbf{6}\,\mathbf{7}\,|\,5 &\sim 1\,2\,3\,4\,\mathbf{6}\,\mathbf{7}\,|\,5\\
\text{ott11: } 1\,2\,3\,\mathbf{6}\,\mathbf{7}\,|\,4\,5 &\sim 1\,2\,3\,\mathbf{6}\,\mathbf{7}\,|\,4\,5\\
\text{ott9: } 1\,2\,3\,|\,4\,5 &\sim 1\,2\,3\,\mathbf{6}\,\mathbf{7}\,|\,4\,5\\
\text{ott8: } 1\,2\,|\,3\,4\,5 &\sim 1\,2\,\mathbf{6}\,\mathbf{7}\,|\,3\,4\,5\\
? &\sim 1\,\mathbf{6}\,\mathbf{7}\,|\,2\,3\,4\,5
\end{aligned}$$

This mapping is based on the idea that, after conflicting branches are removed from $\mathbb{T}|\tau$, then its remaining splits and the splits of τ will be jointly compatible. Some of the splits of $\mathbb{T}|\tau$ can be extended using Z-closure (?).

This is not completely clear though. τ contains only full splits, whereas splits from $\mathbb{T}|\tau$ can be partial.

2.2.9 Example 2

Question: Suppose that τ_1 contests the *incertae sedis* clade (ott6,ott7)ott9. What then?
Answer: Any taxa with ott6 or ott7 placed into them should not be broken if they would have allowed ott9.

Question: Suppose that τ_1 contests the *incertae sedis* clade (ott6,ott7)ott9 and attaches ott6 and ott7 separately to the ott3 terminal edge?
Answer: In this case the breaking of ott9 and the separate placement of ott6 and ott7 together constitute the "placement" of ott9. Thus, we need to be able to represent the conflict/annotation of the taxonomy tree and the input tree τ_1.

2.2.10 Interactions between input trees and *incertae sedis* taxa

Incertae sedis nodes are features of the taxonomy. Each input tree can relate to an *incertae sedis* taxon $(A, B, C)D$ in a number of ways

- it could place a descendant taxon of D
- descendants of incertae sedis taxa may be placed on branches, as well as nodes. That is the taxa will probably be places on new degree-2 nodes that bisect branches, instead of being places on nodes.
- it could place children of the branch 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. This is because none of A, B, or C is in the exclude set of the siblings of D.
- it could resolve A, B, C, or D.
- it could place some descendant taxa of D in a *different place* than another input tree.

An conflict analysis between the taxonomy $\mathbb{T}$ and an input tree τ should yield, for the induced tree $\mathbb{T}|\tau$

- which branches correspond between $\mathbb{T}|\tau \longleftrightarrow \tau$.
- which branches of $\mathbb{T}|\tau$ the input tree τ conflicts with.

2.2.11 Possible issue 1:

If two siblings are *incertae sedis*, then they could each be subgroups of the other. This means that the groups don't form a proper multi-connected tree, which could be problematic. We could therefore make *incertae sedis* groups impenetrable to other *incertae sedis* siblings.

2.2.12 Possible issue 2:

If the decomposer places IS taxa to subproblem resolution, then their new placement no longer specifies exactly which taxa they float over, right? Suppose X floats over (A, B), and A floats over B. Then if X and A are placed within B within subproblem *ottid*, then the fact that X floats over A but not *vice versa* will be lost.

2.2.13 Possible approach

One approach would be to modify the taxonomy to place *incertae sedis* taxa when scanning input phylogenies. A series of input phylogenies containing taxon X might not disagree, but might each place the taxon X successively more tipward. However, when reading an input phylogeny that conflicts with the placement of X resulting from previously processed input phylogenies, we would end up marking the branches connecting the two attachment points as conflicting with the most recently processed input phylogeny. These branches would then not be used to separate subproblems from each other, and would thus end up in the same subproblem. *This is not proved - expand and check.*

If this does work, then it is hopeful that this would decrease the number of subproblems too much. It seems like it might be OK, since we already contest edges result from alternative placement of non-*incertae sedis* taxa.

2.3 Annotations

After we allow *incertae* sedis taxa in the taxonomy, the relationships of edges in the taxonomy to edges in each input tree become more interesting. Without *incertae sedis* taxa, taxonomy splits always divide the full leaf set, and so we can say that taxonomy split A displays an input split B if $B_1 \subseteq A_1$ and $B_2 \subseteq A_2$. Thus, a taxonomy split always has more information than an input split B that it displays. However, with *incertae sedis* taxa in the taxonomy, it is possible that the input split B "aligns" to a taxonomy split A and *also* performs a placement. In this case the input split B has more information than the taxonomy split A that it aligns to.

BDR: Question: when comparing the taxonomy to an input tree, how do we deal with conflict & alignment?

- If we remove all taxonomy splits that are not pairwise compatible with each input tree edge, are all taxonomy edges jointly compatible with all input tree edges?
    1. It is possible for two taxonomy edges to be implied by the same input tree edge.
    2. It is possible for two input tree edges to imply the same taxonomy edge.
    3. Each input tree edge is full rank on the induced tree.
    4. Now, each taxonomy edge is either equivalent an input tree edge, or on one side of it, since they are compatible.
    5. This imposes a directed flow on the input tree for each taxonomy edge, so that it moves to a node or a connected group of edges, all of which imply it.
- We annotate the synthesis tree, not the taxonomy tree.
- Adding incertae sedis info would complicate performing conflict analysis against the taxonomy.

2.4 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 taxonomy node n is the tipward vertex of an edge in the taxonomy tree, and thus corresponds to a split *include/exclude*. Without *incertae sedis*, the split information reduces to the set of tip nodes in the *include* group. Thus, we may simply search the tree for a node m that has the same include group as our taxonomy node n and then draw a correspondence between the two nodes by assigning the taxon name for n in the taxonomy to m in the supertree.

In the *incertae sedis* framework, this is complicated by the fact that the exclude group is not just $\mathcal{L} - include$, but may be smaller. This raises the question of whether one name could apply to multiple nodes, or whether multiple names could apply to one node.

2.4.1 Multiple nodes that fit one name

A node m is consistent with a split $A_1|A_2$ for a taxon name if the node m is the tip-ward vertex of a branch with split $B_1|B_2$ and $A_1 \subseteq B_1$ and $A_2 \subseteq B_2$. So, suppose that node m in the supertree is consistent with split $A_1|A_2$ for node n in the taxonomy tree. Now suppose that in the supertree, m has only one other sibling, which is an *incertae sedis* taxon placed here from a more root-ward position in the taxonomy. Since the sibling is incertae sedis, it won't be in the exclude group A_2, and therefore the split $A_1|A_2$ will apply to the parent of n as well. Thus, it is possible to find a series of connected nodes that satisfy the same taxonomy split, if the include groups of the nodes differ only in their inclusion of *incertae sedis* taxa.

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

2.4.2 Multiple names fit a single node

Suppose that n_2 is the child of n_1, and the only other child n_3 of n_1 is incertae sedis. Also suppose that there is a supertree node m that contains the children of n_2 and that n_3 is placed deeply inside m. This can happen because n_2 does not exclude n_3, since n_3 is *incertae sedis*. In this case, the names n_1 and n_2 will *both* apply to m.

In general, if a taxon contains 2 non-IS taxa, then it cannot be identical with any of its children in the synthesis tree.

In general, 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.

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.

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.

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.

2.4.3 Implementation

So, if we know which nodes in the taxonomy are consistent with the solution, then we can simply find the MRCA and assign a name to it. If we attempt to assign a name to an already-named node, we create a new monotypic ancestor that is rootward of all the existing monotypic ancestors, and assign the name there. This requires that we process the displayed taxonomy nodes in post-order, so we add more root-ward monotypic nodes second, so that they end up more root-ward on the solution tree.

2.5 Unpruning

Currently the unpruner might require that the OTT ids are named in the grafted solution before unpruning starts.

The unpruner should record when unpruned nodes are *incertae sedis*. In combination with the

2.6 Labelled 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.

2.7 Annotation

The synthesis tree itself does not currently have *incertae sedis* groups. Furthermore, the primary goal is to avoid excluding *incertae sedis* taxa, whereas displaying *incertae sedis* status for solution nodes in the tree viewer is a secondary goal. Therefore, solution nodes can be regarded (for the moment) as always implying full splits.

Actually we do not need to change annotation as long as we are not running conflict analysis on the synthesis tree. Basically naming the nodes *is* a (almost) run of conflict analysis on the synthesis tree.

However, we would like to be able to do this.

2.7.1 Conflict algorithm

1. Get induced trees on intersection of leaf sets
2. Compute 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

(f) leaves2 <- get list of corresponding leaves (in synth)

(g) nodes <- find all nodes between leaves2 and the MRCA (in synth)

(h) MRCA <- mrca of leaves2 (in synth)

(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.

3 Unified handling of incompatible splits and broken taxa

Currently when an input tree split conflicts with a previously incorporated split during subproblem solution we discard it altogether. However, when handling a broken taxa during unpruning we do not discard the broken taxon altogether, but instead attach its children at the MRCA of included children. This unpruning can be seen as the solution of a special kind of subproblem that contains the grafted solution (which is pruned) and the full taxonomy (which is not). If the two procedures are not consistent, then pruning will lead to different results. Specifically, lack of pruning would lead not just to slower computation, but to a tree which does not attach the children of broken taxa at the MRCA. Thus, we desire to achieve consistency between handling of broken taxa during pruning and incompatible splits during subproblem solution.

3.0.1 Hypothetical/Platonic: Make the subproblem solver perform unpruning?

Would it be theoretically possible to make the subproblem solver able to perform unpruning by simply feeding the special subproblem

```
grafted solution (pruned)
cleaned ott (unpruned)
```

to the subproblem solver? If so, that would simplify a lot of things conceptually. However, this seems computationally challenging.