Goal

We want to solve the cophylogeny reconstruction optimization problem (CROP) in case:

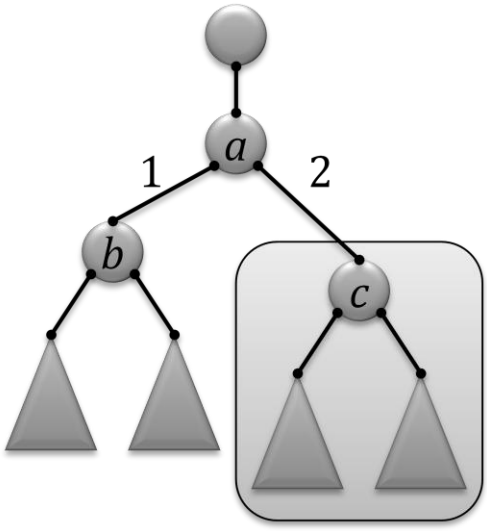
- Timing information is unknown for both trees. We can assign the time freely as long as there are no timing incompatibilities and no events occur at exactly the same time.
- The host-switch distance is limited to $d \leq 2$. That is, for a pair of edges (hosts) that host-switch could occur, there can be at most one edge (two nodes) between them.
- The cost for associating tips is 0.

Assumptions

- The cost for cospeciation is non-negative, and the costs for the other three events (duplication, host-switch and loss) are positive.
- The cost for cospeciation is not greater that the costs for the other three events.

Observation

Recall that on a tree, nodes represent speciations and edges represent species. Furthermore, if there is a host-switch between a pair of host edges, those edges must appear together at some time (in history). Consider the figure below for the host tree H, where a has two non-tip children b and c. Since we do not allow any events to happen at the same time, without loss of generality, suppose that b happens before c.



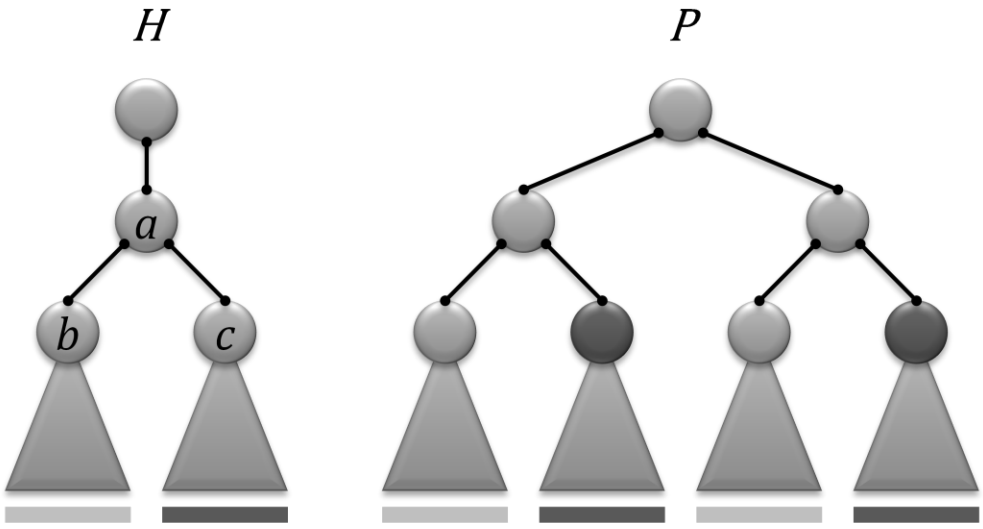
From the figure above, there cannot be host-switch between edges inside and outside the sub-tree rooted at c because for edges within sub-tree rooted at c,

- host-switch to edges in sub-tree of b is impossible since the distance will be more than 2,
- host-switch to edge 2 is impossible since edge 2 ends before they appears,
- host-switch to edge 1 is impossible since b happens before c, so edge 1 ends before edge 2

From this observation, we may be able to break the problem into two smaller parts and solve them independently. We will also try the case where c happens before b and choose a better answer.

Problem breakdown

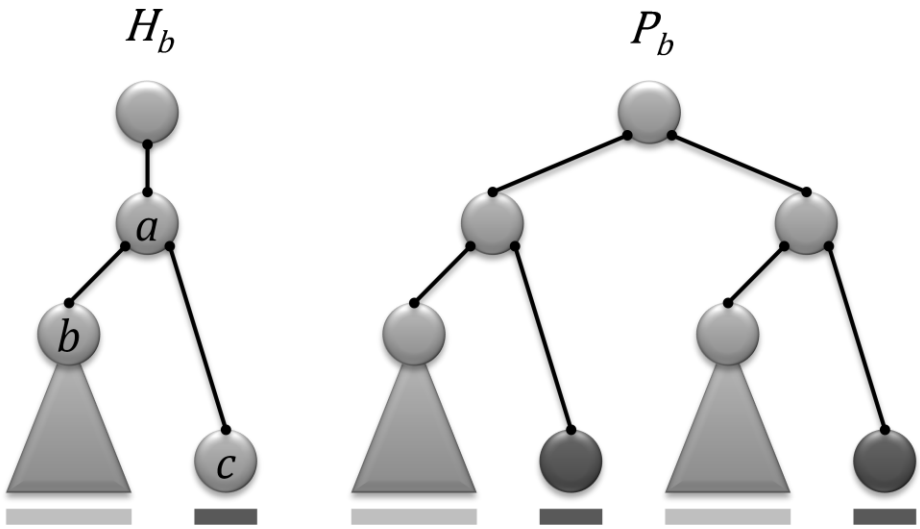
We will break the problem down until the host does not contain any node with two non-tip children, and then solve this problem with dynamic programming. In the figure, suppose that a is the node closest to the root with two non-tip children b and c. We will use the gray underlines to represent the association between tips of each tree.



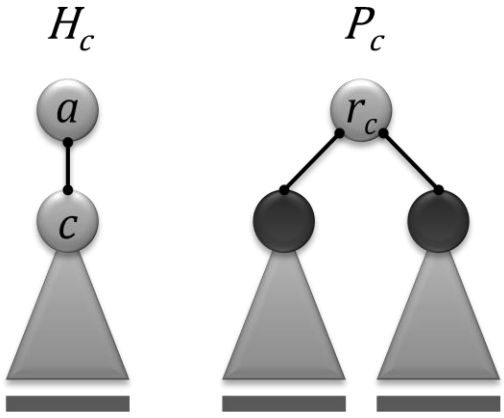
Let us consider the position of these darker nodes when mapped optimally on H. They cannot be mapped on the left sub-tree of a (including the edge ab), as we have shown that their children cannot get back to the right sub-tree of a via host-switch. They also cannot be mapped at a or above, because the number of loss events can be reduced by moving them down under a. Thus, they will be mapped on the right sub-tree of a (including the edge ac).

Now we will construct two sub-problems. Firstly, let us consider the part that involves the tip descendants of b and not c – let us call the host tree and parasite tree for this case H_b and P_b, respectively. For the host tree, we construct H_b simply by removing all the descendants of c. Since all tips in P_b must be associated with tip descendants in H_b, we can only remove the sub-trees with all tips associated to tip descendants of c. Therefore, we construct P_b as follows: for each node with all tip descendants associated to tip descendants of c, and its parent does not have this property, remove all its descendants, and associate this node to c in P_b. These nodes are marked with darker color in the figure above.

The cheapest association of H_b and P_b will give the optimal solution for associations of all parasite nodes outside the sub-trees we removed. Additionally, by associating marked nodes at c, the cost of this association will also include the cost for association of these marked nodes, only for the section outside of the right sub-tree of a. For example, the cost of a loss at a is included in this sub-problem, but the cost of a loss at c is not accounted for yet. The second sub-problem will take care of the cost of the right sub-tree of a and the association of all marked nodes as their descendants.



Now we need to construct the second sub-problem, consisting of the host tree H_c and the parasite tree P_c. In H_c we simply remove all nodes except for a, c, and descendants of c. For P_c, we will need to take care of all of the sub-trees rooted at the marked nodes. These trees must not create timing incompatibilities with one another. So, we need to place all these trees together on the right sub-tree of a. We have showed that the marked nodes will be placed somewhere on the right sub-tree of a including the edge ac. So, we will create a new root r_c that has all the marked nodes as its children. Furthermore, we will associate r_c with a without any cost. With this, each of the sub-trees can be placed under a on H_c together. Let us call this problem the "modified cophylogeny reconstruction optimization problem" (MCROP), where the root of the host tree and the parasite trees are required to associate and does not have any costs, and the parasite tree may not be a binary tree.

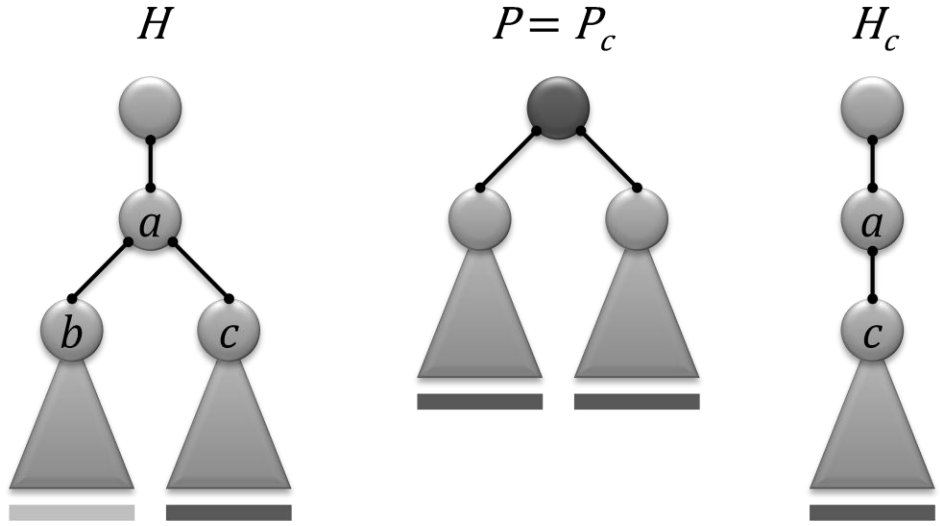


Lastly, in order to obtain the final solution, the position of a parasite node v on the host tree can be obtained as follows:

- if v is in P_c, then the association of v is the same as that of the solution to P_c on H_c;
- otherwise, the association is the same as that of the solution to P_b on H_b.

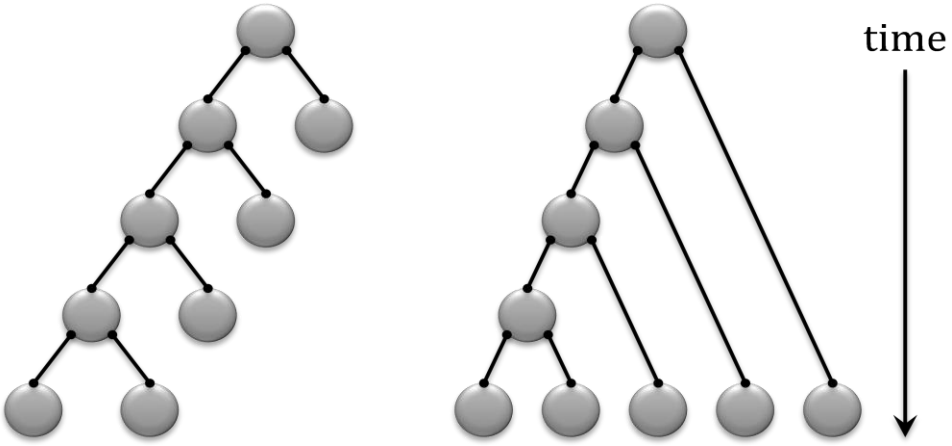
Special case for problem breakdown

If we are breaking down a MCROP instance and r_c becomes the marked node itself, then we do not need to change P_c from the original one. For H_c, we simply remove b and all its descendant. Again, the two roots are required to associate without any cost. Note that, we may skip one of the recursive calls if the parasite tree is empty.



The base case

The base case is where we do not break the problem down any further. This case occurs only when there are no vertices with two non-tip children in the host tree. The tree will have shape as in the figure below on the left hand side.



One possible timing is given on the right hand side, where all host-switch of distance $d \leq 2$ are allowed and no timing incompatibility could occur. So, we can solve this problem with dynamic programming easily by finding the optimal solution for placing a parasite edge e_p on a host edge e_h using the sweep-time algorithm with the timing above. However, since there are only $O(1)$ edges where the child edge of e_p can switch to as the distance is limited, the running time for each update of each e_p is $O(1)$, and the overall running time of this algorithm becomes $O(n^3)$.

Overall Algorithm

On the host tree, find a node a closest to the root with two non-tip children b and c.
If no such node exists, then solve the problem with the described algorithm, and return the solution.

Split the problem in two sub-problems where b is specified to occur before c. Solve them recursively, and construct the solution Φ_1

Split the problem in two sub-problems where c is specified to occur before b. Solve them recursively, and construct the solution Φ_2

Return the solution with lower cost among Φ_1 and Φ_2.

Running time

We will measure the problem size h from the height of the sub-tree of the host tree, rooted at the node closest to the root with two non-tips children. (If there is no such vertex, let the size be 1). By definition, the base case has $h = 1$.

In the problem breakdown process, first consider H_b. Since we remove all descendants of c, c becomes a tip, and therefore a no longer has two non-tips children. As a result, the problem size decreases by 1. Now for H_c, a also does not have two non-tips children any more, and the problem size is also decreased by 1.

From our algorithm, in order to solve a problem of size h, we make 4 recursive calls to solve sub-problems of size $h - 1$, two for each timing. Therefore, the running time of this algorithm for the recursion part is $T(h) = 4T(h - 1)$. For the base case, we have shown that the running time is $T(1) \in O(n^3)$. Therefore, the overall running time of the algorithm is $T(h) \in O\left(n^3 4^h\right)$.

Since h is bounded by the height of the tree which can be at most $n - 1$, the running time of the algorithm is $O(n^3 4^n)$. However, if the tree is known to be sufficiently balanced that the height of the tree is at most $k \log_2 n$ for some constant k, then the running time of this algorithm becomes $O\left(n^{2k+3}\right)$.