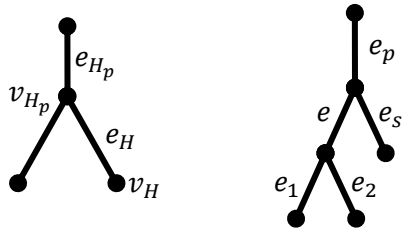
Dynamic programming algorithms for cophylogeny reconstruction problem for known host timing information, and unknown parasite timing information, and its modification to allow timing incompatibilities

Notations

Let n_H and n_P be the number of tips of the host tree and the parasite tree, respectively.
Let $n = n_H + n_P$.

Let the (almost binary) host tree be $G_H = (V_H, E_H)$ and the (almost binary) parasite tree be $G_P = (V_P, E_P)$. For convenience, we will also have a dummy root for the parasite tree as well. These graphs should be rooted and directed. In figures, we will use black for those related to the host, and gray for those related to the parasite. The higher part of the tree represents vertices and edges closer to the root occur earlier in history, while the bottom part represents those close to the tips and occur more recently.

Please assume that as we write v_H, e_H, v_P or e_P probably with subscript in this document, then we refer to $v_H \in V_H$, $e_H \in E_H$, $v_P \in V_P$ and $e_P \in E_P$, respectively. Also, by replacing e for edges with v, we refer to the vertex at the lower (later) endpoint of the edge e. For example, v_H is the tip of e_H that happens later, while v_{H_p} is the tip of v_H that happens earlier (see the left figure below).
Furthermore, for edge e, we refer to its children edges (edges whose earlier tip is v) as e_1 and e_2. We refer to its parent edge (whose lower tip is the higher tip of e) as e_p, and its sibling (the other edge that share the higher tip with e) as e_s (see the right figure below).



In case we have region information for host-tree, let $R = \{r_1, \dots, r_{|R|}\}$ be the set of all regions where all locations on the host tree (edges or vertices) belong to. Let $R(e_H)$ be the region where the host edge e_H resides.

In case we have limited host-switch distance, let k be the maximum host-switch distance, which is in $O(h)$ where h is the height of the tree, which we expect to be between $O(\log\ n)$ and $O(\sqrt{n})$, but can be as high as $O(n)$ theoretically.

Let the constants $\text{cost}_{\text{COSPEC}}$, cost_{DUP} and $\text{cost}_{\text{LOSS}}$ be the costs for cospeciation, duplication, and loss events, respectively. Let cost_{HS}, the cost for host-switch that includes the duplication cost already, be the following:

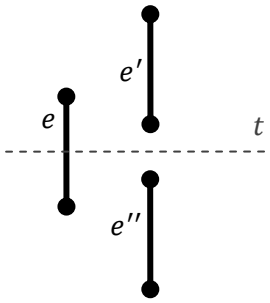
- For uniform host-switch cost, let the cost_{HS} be the constant representing that cost – it does not need any parameter
- With specified cost between each pair of host edges, let $\text{cost}_{\text{HS}}(e_H, e_{H'})$ be the function that gives cost of host-switch from host edge e_H to edge $e_{H'}$.

- With region information, let $\text{cost}_{\text{HS}}(r, r')$ be the function that gives the cost of host-switch from region r to region r'. Note that it can be considered a special case of the case with specified cost between each pair of edge – the reduction can be done using the composite function $\text{cost}_{\text{HS}}\big(R(e_H), R(e_{H'})\big)$ instead.

In case we have timing information, let T be the set of all possible times of a host event. In general, we require all elements in T are integers, the maximum element to be within $O(n)$, and the minimum to be non-negative. In Jane, the set of time $T = \{0,1, \dots, n_H\}$, where

- The dummy root of the host tree occurs at time 0,
- All host tips occurs at time n_H,
- For each time $1, 2, \dots, n_H - 1$, exactly one host speciation event occur.

An edge e_H is considered "alive" at some (not necessarily integral) time t if one of its endpoints occurs at or before time t, and the other endpoint occurs at or after time t. This represents a species being alive at time t in history. In the figure below, edge e is alive at time t, while edge e' and e'' are not alive at time t.



Algorithm 1: allow a host-switch cost function for all pair of host edges
Running time $O(n^4)$

Firstly, let us define two dynamic programming tables A and B.

Let A: $E_P \times E_H \times T \to \mathbb{R}$ be the function where $A(e_P, e_H, t)$ calculates the following value:
the **cost of optimal way** to place the **parasite edge** e_P on the **host edge** e_H **right after** time t, including the cost of all its **sub-tree**. By "**right after**", we mean that no other event occur after time t.

Let B: $E_P \times E_H \times T \to \mathbb{R}$ be the function where $B(e_P, e_H, t)$ calculates the following value:
the **cost of optimal way** to place the **parasite edge** e_P on the **host edge** e_H **right before** time t, including the cost of all its **sub-tree**. By "**right before**", we mean that no other event occur before time t.

Our plan is to start from the tips of both trees by finding initial values of $B(e_P, e_H, n_H)$. Then we will compute $A(e_P, e_H, t-1)$ from $B(e_P, e_H, t)$, and $B(e_P, e_H, t)$ from $A(e_P, e_H, t)$, alternatively in decreasing order of t until we use all edges in the parasite tree, which will give us the cost of the solution. Note that we can always extend A and B to store not only the cost of the best association, but also the association (solution) itself. Note that "association" here is a set of mapping between each element in a subset of vertices in V_p, onto locations (edges E_H or vertices V_H) on the host tree.

The candidate for final solution (full tree of parasite on the tree of host) will be at any A or B with the edge connected to the dummy root of the parasite tree, which suggests that we have already place the entire parasite tree on the host tree already. Thus,

$$\text{SOLUTION} = \min_{e_H,t}\{A(e_{P_{\text{root}}}, e_H, t), B(e_{P_{\text{root}}}, e_H, t)\}$$

over all $e_H \in E_H$ and $t \in T$, where $e_{P_{\text{root}}}$ is the dummy root edge of the parasite tree. This solution can be computed after all cells on the tables are filled, or updated each time a cell in the tables is filled.

The overview of the algorithm is shown below. The necessity of filling the table in post-order of e_P will be explained later.

```
For each time t in decreasing order
    For each parasite edge e_P in post-order
        For each host edge e_H in any order
            Compute B(e_P, e_H, t)
        For each host edge e_H in any order
            Compute A(e_P, e_H, t − 1)
```

Calculation of $B(e_P, e_H, t)$ for $t = n_H$

This is the base case: to calculate $B(e_P, e_H, n_H)$, we already know the associations of all the tips. Furthermore, we cannot include any events at all as we compute the cost for right before time n_H. Therefore, only possible placement of edges is for those with a tip on one end. That is,

- If v_P and v_H are associated tips, then $B(e_P, e_H, n_H) = 0$.
- Otherwise, $B(e_P, e_H, n_H) = \infty$.

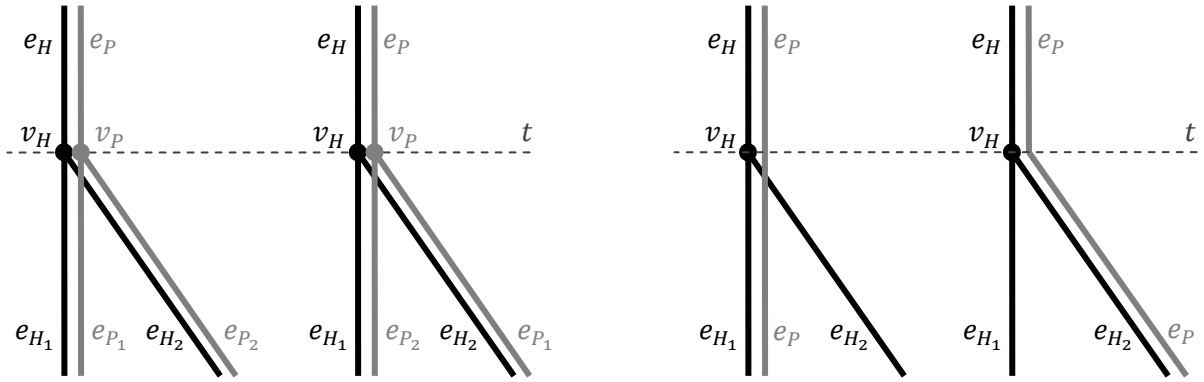
Calculation of $B(e_P, e_H, t)$ from $A(e_P, e_H, t)$ for $t < n_H$

From our definition, this will be a calculation of events happen exactly at time t. Assuming that no events can occur at the same time, the only allowed event at time t involves exactly one speciation of the host tree.

Case 1 v_H occurs at time t
There are two possible event types: a cospeciation event or a loss event. So, let

$$B(e_P, e_H, t) = \min\{\text{COSPEC}(e_P, e_H, t), \text{LOSS}(e_P, e_H, t)\}$$

where $\text{COSPEC}(e_P, e_H, t)$ and $\text{LOSS}(e_P, e_H, t)$ are the minimum cost for mapping e_P on e_H right before time t with cospeciation and loss event occurring with v_P on v_H at time t, respectively.



For cospeciation, e_{P_1} and e_{P_2} must lie on e_{H_1} and e_{H_2} in some sequence. From the figure above, COSPEC can be computed as follows:

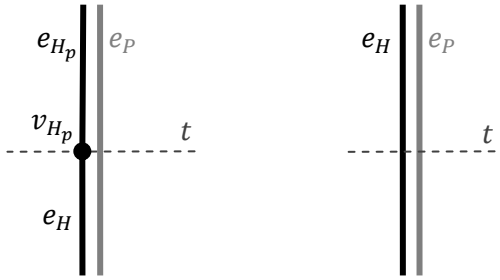
- If v_P is not a tip, then $\text{COSPEC}(e_P, e_H, t) =$
  $\min\left(A(e_{P_1}, e_{H_1}, t) + A(e_{P_2}, e_{H_2}, t), A(e_{P_1}, e_{H_2}, t) + A(e_{P_2}, e_{H_1}, t)\right) + \text{cost}_{\text{COSPEC}}$.
- Otherwise, $\text{COSPEC}(e_P, e_H, t) = \infty$.

In case of loss, e_P must come from one of the edges, either e_{H_1} or e_{H_2}. However, in case e_P is the dummy root edge, we do not allow a loss event on e_P (since in the original problem, we do not have a dummy root parasite edge). Therefore, LOSS can be computed as follows:

- If e_P is not a dummy root, then
  $\text{LOSS}(e_P, e_H, t) = \min\left(A(e_P, e_{H_1}, t), A(e_P, e_{H_2}, t)\right) + \text{cost}_{\text{LOSS}}$.
- Otherwise, $\text{LOSS}(e_P, e_H, t) = \infty$.

Case 2 v_H does not occur at time t but v_{H_p} (the endpoint of e_H that occurs earlier) occurs at time t
In this case, e_H first occur at time t. So it is not alive at time before t and therefore e_P cannot be associated on e_H before time t. So, $B(e_P, e_H, t) = \infty$.

Case 3 Neither of the endpoints of e_H occur at time t
In this case, there is no event that changes the state of e_H between alive and not alive at time t. Therefore, no events can occur between e_P and e_H at time t, and the solution is the same as that after time t. Therefore, $B(e_P, e_H, t) = A(e_P, e_H, t)$.



Case 2 Case 3

Calculation of $A(e_P, e_H, t-1)$ from $B(e_P, e_H, t)$

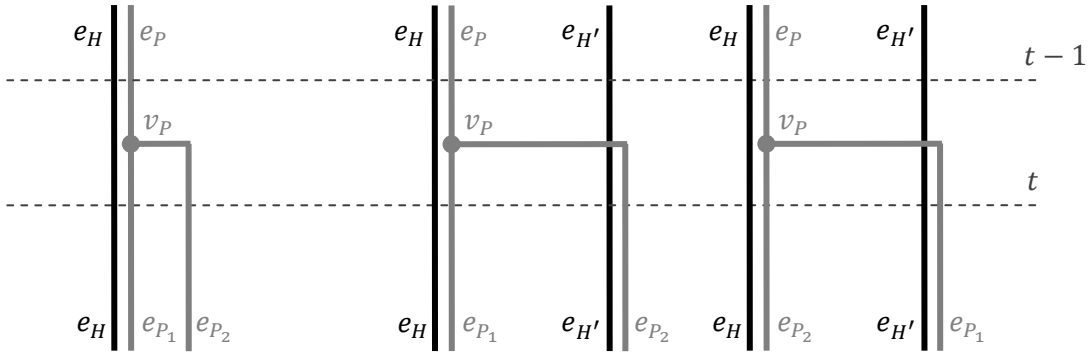
During the time after $t-1$ until before t, there are no changes in the host tree. Therefore, the events that can occur during that time are only duplications and host switching. Between time $t-1$ and t, more than one events can occur, However, there can be exactly one event that occur to e_P, which is a duplication or a (duplication with) host-switch. On the other hand, there may not be any events related to e_P during this time slice at all.

Case 1 e_P undergoes duplication or host-switch between time $t-1$ and t

In the same fashion as calculation for entries of B table, we let

$$A(e_P, e_H, t-1) = \min\{\mathrm{DUP}(e_P, e_H, t-1), \mathrm{HS}(e_P, e_H, t-1)\}$$

where $\mathrm{DUP}(e_P, e_H, t-1)$ and $\mathrm{HS}(e_P, e_H, t-1)$ are the minimum cost for mapping e_P on e_H right after time $t-1$ with duplication and host-switch occurring on e_P over e_H at time t, respectively.



Duplication without host switching

Duplication with host switching

For duplication, both e_{P_1} and e_{P_2} must be associated with e_H. However, since there can be more events occurring with e_{P_1} and e_{P_2} in this time slice, we need to refer to the A table for time $t-1$ instead of the B table. If we have filled out those cells related to e_{P_1} and e_{P_2}, we can compute DUP as follows:

- If v_P is not a tip, then $\mathrm{DUP}(e_P, e_H, t-1) = A(e_{P_1}, e_H, t-1) + A(e_{P_2}, e_H, t-1) + \mathrm{cost}_{\mathrm{DUP}}$.
- Otherwise, $\mathrm{DUP}(e_P, e_H, t-1) = \infty$.

For host-switch, either e_{P_1} or e_{P_2} needs to switch to a different host edge $e_{H'}$ alive right after time $t-1$. So, we can compute HS as follows:

- If v_P is not a tip, then
  $\mathrm{HS}(e_P, e_H, t-1) = \min_{e_{H'}} \begin{Bmatrix} A(e_{P_1}, e_H, t-1) + A(e_{P_2}, e_{H'}, t-1), \\ A(e_{P_2}, e_H, t-1) + A(e_{P_1}, e_{H'}, t-1) \end{Bmatrix} + \mathrm{cost}_{\mathrm{HS}}(R(e_H), R(e_{H'}))$
  for all $e_{H'}$ where a host-switch from e_H to $e_{H'}$ is allowed and alive right after time $t-1$.
- Otherwise, $\mathrm{HS}(e_P, e_H, t-1) = \infty$.

Case 2 e_P does not undergo any events between time $t-1$ and t
Then, $A(e_P, e_H, t-1) = B(e_P, e_H, t)$ as the cost cannot change when no event occurs.

Since the optimal solution can be from any of these two cases, then we obtain the relationship
$A(e_P, e_H, t-1) = \min\{\mathrm{DUP}(e_P, e_H, t-1), \mathrm{HS}(e_P, e_H, t-1), B(e_P, e_H, t)\}$.

Running time

From the formulas for computing A and B, we only refer to value in cells that have later (higher) time t, and only children edges. This explains why we need to compute the table in decreasing order of t and post-order of e_P.

To check whether a host-switch from e_H to $e_{H'}$ is allowed, if the condition depends on the host-switch distance, then we can pre-calculate the distance between all pair of hosts first, so that this check can be done in constant time. The calculation of distance can be done with $O(n)$ depth-first searches, which has running time of $O(n^2)$ in total – alternatively, compact algorithms like the Floyd-Warshall algorithm will not increase the asymptotic running time.

Since all referenced cells in the formulas are pre-computed, then the computation of each cell of B takes constant time. For A in case of host-switch, however, we need to look through all $e_{H'}$ that are alive at the given time, which can take as high as $O(n)$ running time. Therefore, in the worst case, the computation of each cell in A is $O(n)$. We need to fill in all $O(n^3)$ cells, and the highest running time needed is $O(n)$ in case of host-switch. Therefore, the total running time for filling out the tables is $O(n^4)$.

The finding of final solution can be done by looping through each e_H and t in $O(n^2)$. Alternatively, we can update the final solution when each cell of A and B is computed. At any rate, this will not increase the asymptotic running time. As a result, the running time of this algorithm is $O(n^4)$.

Implementation

The implementation should include the outer loop for time, with two loops inside – one for calculation of B from A and another for A from B. For each loop, go through the edges in E_P first in post-order. Then go through E_H in any order, and calculate the value as described above.

In each calculation of A from B, we can simply set the initial value to ∞, and try to update the value (and association) as we look at each possible case.

In calculation of B from A, however, we may need to construct an array for A to avoid using the speciation on the host tree more than once, and then free B later. This way, the memory needed is $O(n^2)$ cells (the t dimension does not need to be stored). We should also consider calculating the list of active host edges at the beginning of the iteration or even keep track of it the whole time instead of looping through the edges every time. We can also pre-compute the post-ordering of the parasite edges for convenience.

Note that since we create dummy edges, there is a bijection between edges and vertices. In implementation, it should be easier to refer to edges by its endpoint with higher (later) time.

Algorithm 2: for no limit on host-switch distance and no region information (uniform cost_{HS}) Running time $O(n^3)$

With these two extra assumptions, we obtain the equation for HS:

$$\text{HS}(e_P, e_H, t-1) = \min\begin{Bmatrix} A(e_{P_1}, e_H, t-1) + \min_{e_{H'}} A(e_{P_2}, e_{H'}, t-1), \\ A(e_{P_2}, e_H, t-1) + \min_{e_{H'}} A(e_{P_1}, e_{H'}, t-1) \end{Bmatrix} + \text{cost}_{\text{HS}}$$

for all $e_{H'}$ since cost_{HS} does not depend on the take-off and landing sites. However, we can compute $\min_{e_{H'}} A(e_{P_1}, e_{H'}, t-1)$ and $\min_{e_{H'}} A(e_{P_2}, e_{H'}, t-1)$ easily by keeping track of the minimum $A(e_{P_1}, e_{H'}, t-1)$ and $A(e_{P_2}, e_{H'}, t-1)$ found so far as we compute them. Let us store this value in table D where $D(e_P, t)$ is defined as

$$D(e_P, t) = \min_{e_{H'}} A(e_P, e_{H'}, t-1)$$

and is computed as follows:

- The original value of each cell of $D(e_P, t)$ is ∞.
- As we compute $A(e_P, e_H, t)$ for any e_H, we update the cost of $D(e_P, t)$ to $A(e_P, e_H, t)$ if the new cost is lower: $D(e_P, t) = \min\{D(e_P, t), A(e_P, e_H, t)\}$.

With this, we can compute the optimal host-switch cost for e_{P_2} switching to some active via the formula

$$\text{HS}(e_P, e_H, t-1) = \min\begin{Bmatrix} A(e_{P_1}, e_H, t-1) + D(e_{p_2}, t-1), \\ A(e_{P_2}, e_H, t-1) + D(e_{p_1}, t-1) \end{Bmatrix} + \text{cost}_{\text{HS}}$$

in constant time.

Occasionally, the optimal cost may be obtained when the take-off and landing sites are the same. If cost_{HS} is non-negative, then a duplication will provide a better cost. However, in case we allow $\text{cost}_{\text{HS}} < \text{cost}_{\text{DUP}}$, then we will need to keep track of the best **two** values of $D(e_P, t)$, and also the landing site of the best one. This will allow us to check and prevent host-switch from and to the same edge within constant running time.

As a result, the computation of each value in A table can be done in $O(1)$ running time. Therefore, the total running time of the algorithm becomes $O(n^3)$.

The overall algorithm can be modified as shown below.

```
For each time t in decreasing order
    For each parasite edge e_P in post-order
        D(e_p, t − 1) ← ∞
        For each host edge e_H in any order
            Compute B(e_P, e_H, t)
        For each host edge e_H in any order
            Compute A(e_P, e_H, t − 1)
            D(e_P, t − 1) ← min{D(e_P, t − 1), A(e_P, e_H, t − 1)}
```

Algorithm 2A: allow limited host-switch distance $\leq k$, no region information
Running time $O(n^3k)$

For this case, let us redefine the table D as follows:

$$D(e_P, e_H, t, d) = \min_{e_{H'}} A(e_P, e_{H'}, t)$$

over all edges $e_{H'}$ that is exactly at distance d from A and alive right after time t, and $e_H \neq e_{H'}$ (i.e., host-switch form edge e_H to $e_{H'}$ is allowed because the switch distance does not exceed k, with an assumption that host-switch with same take-off and landing sites is not allowed), for $0 \leq d \leq k$, where k is the highest possible/allowed host-switch distance. Notice that $e_{H'}$ cannot be an ancestor or a descendant of e_H since they cannot also occupy the time right after t – our algorithm will not consider those edges for $e_{H'}$ as candidate landing sites. With this, if D for e_{P_1} and e_{P_2} at time t are pre-computed, then we can compute

$$\mathrm{HS}(e_P, e_H, t-1) = \min \begin{Bmatrix} A(e_{P_1}, e_H, t-1) + \min_d D(e_{P_1}, e_H, t-1, d), \\ A(e_{P_2}, e_H, t-1) + \min_d D(e_{P_2}, e_H, t-1, d) \end{Bmatrix} + \mathrm{cost}_{\mathrm{HS}}$$

in running time $O(k)$, resulting in total running time of $O(n^3k)$ as needed. However, we then need to be able to compute each cell $D(e_P, e_H, t, d)$ in constant time.

To achieve this, notice that because of the tree structure, the distance of host-switch is given by the unique path length between those two edges. However, since the tree is rooted, then to travel between two edges, we need to travel up (closer to the root) at most once, then travel down (away from the root) at most once. With this observation, we introduce a new table C as follows:

$$C(e_P, e_H, t, d) = \min_{e_{H'}} A(e_P, e_{H'}, t)$$

over each edge $e_{H'}$ that is exactly at distance d from A and alive right after time t, and $e_{H'}$ is either e_H or in the sub-tree rooted at v_H, for $0 \leq d \leq k$. Please note that C is a table that we introduce to support the calculation, and does not have a biological implication about the host-switch.

We can rewrite C into a recursive form as following:

- If $d = 0$ and e_H is alive right after time t, then $C(e_P, e_H, t, d) = A(e_P, e_H, t)$.
- Otherwise, if $d \geq 0$ and v_H is not a tip,
  then $C(e_P, e_H, t, d) = \min\{C(e_P, e_{H_1}, t, d-1), C(e_P, e_{H_2}, t, d-1)\}$.
- Otherwise, $C(e_P, e_H, t, d) = \infty$.

Therefore, we can compute each cell of $C(e_P, e_H, t, d)$, for each e_P and t after the table A is computed for this pair of e_P and t, in post-order of e_H, and in any order d, in $O(1)$. So, the total running time does not increase from $O(n^3k)$.

Now we can write D recursively as follows:

- If e_H is the dummy root edge, then $D(e_P, e_H, t, d) = \infty$.
- Otherwise, $D(e_P, e_H, t, d) = \min\{C(e_P, e_{H_s}, t, d-1), D(e_P, e_{H_p}, t, d-1)\}$.

Therefore, we can compute each cell of $D(e_P, e_H, t, d)$, for each e_P and t after the table C is computed for this pair of e_P and t, in pre-order of e_H, and in any order of d, in $O(1)$. Again, the running time is $O(n^3k)$ in total.

The correctness of both recursive formulas for the computation of C and D can be proved inductively. We can also think of the distance in table C as the distance we travel in the downward direction from some ancestor of e_H to $e_{H'}$, while the D table accounts for the upward travel, and keep the total distance from e_H to $e_{H'}$. The overall algorithm can be summarized as shown below:

For each time t in decreasing order
    For each parasite edge e_P in post-order
        For each host edge e_H in any order
            Compute $B(e_P, e_H, t)$
        For each host edge e_H in any order
            Compute $A(e_P, e_H, t-1)$
        For each host edge e_H in post-order, for each d in any order
            Compute $C(e_P, e_H, t-1, d)$
        For each host edge e_H in post-order, for each d in any order
            Compute $D(e_P, e_H, t-1, d)$

Note that to allow host-switch between same edges, the algorithm needs to be modified. The easiest way to do so is to add the comparison with host-switch on the same edge in the formula for computation of HS.

Algorithm 2B: allow region information, no limited host-switch distance
Running time $O(n^3|R|)$

For this case, let us redefine the table D as follows:

$$D(e_P, t, r) = \min_{e_H} A(e_P, e_H, t)$$

over all edges $e_{H'}$ that is in region r and alive right after time t. With this, if D for e_{P_1} and e_{P_2} at time t are pre-computed, then we can compute

$$\mathrm{HS}(e_P, e_H, t-1) = \min \begin{Bmatrix} A(e_{P_1}, e_H, t-1) + \min_r \left(D(e_{P_1}, t-1, r) + \mathrm{cost}_{\mathrm{HS}}(R(e_H), r) \right), \\ A(e_{P_2}, e_H, t-1) + \min_r \left(D(e_{P_2}, t-1, r) + \mathrm{cost}_{\mathrm{HS}}(R(e_H), r) \right) \end{Bmatrix}$$

in running time $O(|R|)$, resulting in total running time of $O(n^3|R|)$ as needed.

$D(e_P, t, r)$ only consider $A(e_P, e_H, t)$ where $e_{H'}$ is in region r. On the other hand, it means that as we compute a new value $A(e_P, e_H, t)$, we only need to update $D(e_P, t, r)$ for $r = R(e_H)$, which takes constant time and does not increase the total running time. The overview of the algorithm is summarized as shown below:

For each time t in decreasing order
    For each parasite edge e_P in post-order
        For each region r in any order
            $D(e_p, t-1, r) \leftarrow \infty$
        For each host edge e_H in any order
            Compute $B(e_P, e_H, t)$
        For each host edge e_H in any order
            Compute $A(e_P, e_H, t-1)$
            $D(e_P, t-1, R(e_H)) \leftarrow \min\{D(e_P, t-1, R(e_H)), A(e_P, e_H, t-1)\}$

Note that to not allow host-switch between same edges, the algorithm needs to be modified. The easiest way to do so is to store best two host-switch sites for each region r in same fashion as explained in Algorithm 2.

Algorithm 2AB: allow both limited host-switch distance $\leq k$ and region information: Running time $O(n^3k|R|)$

It is not hard to see that we can join the relaxations in Algorithm 2A and Algorithm 2B together. This can be done by making modifications to the tables C and D as follows:

$C(e_P, e_H, t, d, r)$ is the minimum value of $A(e_P, e_{H'}, t)$ for all $e_{H'}$ such that $e_{H'}$

- is e_H or its descendant
- is exactly at distance d from e_H
- is alive right after time t
- is in region r

It can be written recursively and computed in constant time as follows:

- If $d = 0$, and e_H is alive right after time t and in region r, then $C(e_P, e_H, t, d, r) = A(e_P, e_H, t)$.
- Otherwise, if $d > 0$ and v_H is not a tip, then $C(e_P, e_H, t, d, r) = \min\{C(e_P, e_{H_1}, t, d-1, r), C(e_P, e_{H_2}, t, d-1, r)\}$.
- Otherwise, $C(e_P, e_H, t, d, r) = \infty$.

$D(e_P, e_H, t, d, r)$ is the minimum value of $A(e_P, e_{H'}, t)$ for all $e_{H'}$ such that $e_{H'}$

- is not e_H, its descendant, or its ancestor
- is exactly at distance d from e_H
- is alive right after time t
- is in region r

It can be written recursively and computed in constant time as follows:

- If e_H is the dummy root edge, then $D(e_P, e_H, t, d, r) = \infty$.
- Otherwise, $D(e_P, e_H, t, d, r) = \min\left\{C(e_P, e_{H_s}, t, d-1, r), D\left(e_P, e_{H_p}, t, d-1, r\right)\right\}$.

With D, we can compute the candidate for host-switch at $A(e_P, e_H, t)$ in $O(k|R|)$ as follows:

$$\mathrm{HS}(e_P, e_H, t) = \min_{d,r} \begin{Bmatrix} A(e_{P_1}, e_H, t) + D(e_{P_2}, e_H, t, d, r) + \mathrm{cost}_{\mathrm{HS}}(R(e_H), r), \\ A(e_{P_2}, e_H, t) + D(e_{P_1}, e_H, t, d, r) + \mathrm{cost}_{\mathrm{HS}}(R(e_H), r) \end{Bmatrix}.$$

Since there are $O(n^3)$ cells of A and B that takes $O(k|R|)$ time to compute and $O(n^3k|R|)$ cells of C and D that takes $O(1)$ time to compute, the overall algorithm takes $O(n^3k|R|)$ as needed.

The overall algorithm can be summarized as shown below:

```
For each time t in decreasing order
    For each parasite edge e_P in post-order
        For each host edge e_H in any order
            Compute B(e_P, e_H, t)
        For each host edge e_H in any order
            Compute A(e_P, e_H, t − 1)
        For each host edge e_H in post-order, for each d and r in any order
            Compute C(e_P, e_H, t − 1, d, r)
        For each host edge e_H in post-order, for each d and r in any order
            Compute D(e_P, e_H, t − 1, d, r)
```

Modifications for Algorithms 1 and 2 to allow timing incompatibilities for unknown host and parasite timing information, reducing the running time by a factor of n

All of the described algorithms can be modified so that the running time is faster by a factor of n if timing incompatibilities are allowed. For simplicity, we will demonstrate this modification with Algorithm 2, where host-switch distance is not limited and host-switch has uniform cost, in case host-switch is allowed between any pair of edges (including same landing and take-off sites, and host-switch between ancestor-descendant). The modification of other algorithms can be done in the same fashion.

To modify Algorithm 2, firstly, we will remove the timing information from our dynamic programming. Let us define two new tables as following:

$S(e_P, e_H)$ stores the minimum cost of associating e_P right after the time that v_{H_p} (the earlier endpoint of e_H) occurs (i.e., where edge e_H **starts**), including the cost of associating the sub-trees of e_P and e_H. This will not include the host speciation that creates e_H. This table is comparable to the A table.

$E(e_P, e_H)$ stores the minimum cost of associating e_P right before the time v_H (the later endpoint of e_H) occurs (i.e., where edge e_H **ends**), including the cost of associating the sub-trees of e_P and e_H. This must include the host speciation v_H. This table is comparable to B table.

With the same reasoning as Algorithm 2, this table can be filled out in the post-order of the parasite edges, and for each parasite edge, in any order of the host edges. Again, we will keep the table D defined as $D(e_P) = \min_{e_H} S(e_P, e_H)$ so that we can compute the cost for host-switch case in $O(1)$. $D(e_P)$ is updated if a new $S(e_P, e_H)$ is better than the value stored in $D(e_P)$, so it does not increase the asymptotic running time. The overall algorithm can be summarized as shown below:

```
For each parasite edge e_P in post-order
    D(e_P) ← ∞
    For each host edge e_H in post-order
        Compute E(e_P, e_H)
    For each host edge e_H in post-order
        Compute S(e_P, e_H)
        D(e_P) ← min{D(e_P), S(e_P, e_H)}
```

Our goal is to compute $E(e_P, e_H)$ and $S(e_P, e_H)$ in $O(1)$, so that the overall running time is $O(n^2)$. They can be computed in the same fashion with B and A, respectively, as follows:

Computation of entries of $E(e_P, e_H)$ when v_H is a tip

This is the base case for the loop – there cannot be events except for tips association. So:

- If v_P is a tip associated with v_H, then $E(e_P, e_H) = 0$.
- Otherwise, $E(e_P, e_H) = \infty$.

Computation of entries of $E(e_P, e_H)$ from $S(e_P, e_H)$ when v_H is not a tip

Since v_H is not a tip, an either a cospeciation or a loss must occur at v_H.
Therefore, we obtain $E(e_P, e_H) = \min\{\text{COSPEC}(e_P, e_H), \text{LOSS}(e_P, e_H)\}$, where:

- If v_H is not a tip, then
  $\text{COSPEC}(e_P, e_H) = \min\{S(e_{P_1}, e_{H_1}) + S(e_{P_2}, e_{H_2}), S(e_{P_1}, e_{H_2}) + S(e_{P_2}, e_{H_1})\} + \text{cost}_{\text{COSPEC}}$.
  Otherwise, $\text{COSPEC}(e_P, e_H) = \infty$.
- $\text{LOSS}(e_P, e_H) = \min\{S(e_P, e_{H_1}), S(e_P, e_{H_2})\} + \text{cost}_{\text{LOSS}}$.

Computation of entries of $S(e_P, e_H)$ from $E(e_{P_1}, e_H)$ and $E(e_{P_2}, e_H)$

There can be either no event, a duplication, or a (duplication with) host-switch event occurring with e_P on e_H. So, we obtain $S(e_P, e_H) = \min\{E(e_P, e_H), \text{DUP}(e_P, e_H), \text{HS}(e_P, e_H)\}$, where

- If v_P is not a tip, then $\text{DUP}(e_P, e_H) = S(e_{P_1}, e_H) + S(e_{P_2}, e_H) + \text{cost}_{\text{DUP}}$.
  Otherwise, $\text{DUP}(e_P, e_H) = \infty$.
- If v_P is not a tip, then
  $\text{HS}(e_P, e_H) = \min\{S(e_{P_1}, e_H) + D(e_{P_2}), S(e_{P_2}, e_H) + D(e_{P_1})\} + \text{cost}_{\text{HS}}$.
  Otherwise, $\text{HS}(e_P, e_H) = \infty$.

Since we remove the loop that account for the decreasing time, then the running time will decrease by the factor of n as needed. In this modification of Algorithm 2, for example, we only need to compute $O(n^2)$ cells in A and B, and each of them can be done in constant time, resulting in running time $O(n^2)$ as needed.

General rules to modify the algorithms to allow timing incompatibilities

- Replace A table with E table, and replace B table with S table
- Remove all timing information t
- Remove the outer loop of algorithm that repeats for decreasing t
- Fill in the table E and S in post-order of e_H
- For table C and D, ignore the conditions related to timing; i.e., we only need the candidate to be in the allowed position on the host tree, and in correct region and distance from e_H, the specified take-off site on host tree.