HMC CS Technical Report CS-2011-1: Faster Dynamic Programming Algorithms for the Cophylogeny Reconstruction Problem

A. Yodpinyanee B. Cousins J. Peebles T. Schramm
R. Libeskind-Hadas

Abstract

The *cophylogeny reconstruction problem* is fundamental in the study of coevolution. Although the problem is known to be NP-complete (Libeskind-Hadas & Charleston 2009, Ovadia, Fielder, Conow & Libeskind-Hadas 2011), several software tools have been developed that solve small instances optimally (e.g. TreeMap) or use heuristics to efficiently find good, but not necessarily optimal, solutions (e.g. TreeFitter, Tarzan, Jane, CoRe-PA). The latter approaches generally use dynamic programming (DP) algorithms that, while not identical, are fundamentally similar. In this paper we describe a new general "edge-based" dynamic programming approach that is substantially more efficient than existing approaches. The new edge-based approach can be used in lieu of the DP steps in existing systems, improving running time and obtaining equally good solutions. For example, the $O(n^3)$ DP in the CoRe-PA system can be replaced by a $O(n^2)$ edge-based DP and the $O(n^7)$ DP step in the Jane 1 cophylogeny tool has been replaced by $O(n^3)$ "edge-based" DP in Jane 2 (*Jane* n.d.), where n is the number of nodes in each tree.

1 Introduction

The *cophylogeny reconstruction problem* seeks to reconcile pairs of phylogenetic trees in order to understand the evolutionary history of two groups. The problem arises in parasitology (hosts and parasites), molecular systematics (genes and species), and biosystematics (areas and organisms). In this problem, we are given two phylogenetic trees and the mappings between their tips, representing associations of extant species. The objective is to map one tree onto the other, minimizing a weighted sum of four

types of biologically plausible events – called cospeciation, duplication, host switch, and loss – that are induced by the mapping.

Since the underlying computational problem is NP-complete (Libeskind-Hadas & Charleston 2009, Ovadia et al. 2011), existing approaches either find optimal solutions in exponential worst-case time or use fast heuristics that find good, but not necessarily optimal, solutions. An example of the former approach is the "jungles" method employed in TreeMap (Charleston 1998, Charleston & Page n.d.). Heuristics are used in TreeFitter (Ronquist 1998, Ronquist n.d.), Tarzan (Merkle & Middendorf 2005, Merkle & Middendorf n.d.), Jane (Conow, Fielder, Ovadia & Libeskind-Hadas 2010, *Jane* n.d.), and CoRe-PA (Merkle, Middendorf & Wieseke 2010, *CoRe-PA* n.d.).

A common feature of many heuristics is that they either relax the constraints of the problem or impose additional constraints. Under certain changes in constraints, the reconstruction problem can be solved optimally in polynomial time using dynamic programming techniques. For example, Tarzan and CoRe-PA relax the constraint on where host switch events can occur and this variant of the problem is solved using a $O(n^3)$ dynamic program, where n denotes the number of nodes in each of the two trees. This approach generally finds good solutions very quickly. However, such unconstrained host switches may lead to invalid solutions such as solutions in which one speciation event occurs before an ancestral event. In contrast, Jane ensures that all solutions found are valid solutions by imposing a relative ordering (or "timing") of the speciation events in the host tree and forcing host switch events to comply with this timing information. Under this constraint, Jane's dynamic programming algorithm finds an optimal solution in time $O(n^7)$. However, there are an exponential number of possible timings and thus Jane can only explore a subset of the solution space. Other similar dynamic programming ideas were described even earlier by Ronquist (Ronquist 1995, Ronquist 1998) and used in TreeFitter (Ronquist n.d.). Interestingly, all of these dynamic programming algorithms work in fundamentally the same way, with small variations due to the nature of the differences in constraints.

In this paper, we describe a more efficient general dynamic programming (DP) technique for cophylogeny reconstruction that can be used in place of the DPs used in existing heuristics. For example, the $O(n^3)$ DP used in CoRe-PA can be replaced by a $O(n^2)$ DP and the $O(n^7)$ DP used in Jane can be replaced by a $O(n^3)$ DP.[1] Finally, this approach can be easily adapted to add functionality such as preferential host switching, among others, indicating that it is may be of utility in future cophylogeny tools.

[1]The new DP is used in Jane 2.

2 Preliminaries

Let $H = (V_H, E_H)$ and $P = (V_P, E_P)$ denote two phylogenetic trees, henceforth referred to as the *host* and *parasite* trees, respectively. Each of these trees has an extra node connected by an edge to what would normally be the root of the tree. In the host tree, this node and edge are denoted v^H and e^H, respectively, whereas in the parasite tree they are denoted v^P and e^P. These node/edge pairs are referred to as "handles" of their respective trees. A handle is required in the host tree in order to account for evolutionary events that predate the most recent common ancestor in the phylogenetic tree. A handle is required in the parasite tree due to the particular nature of our dynamic programming approach. A host tree and parasite tree with their respective handles are shown in Figure 1(a).

The internal vertices, other than the handle vertices v^H and v^P, represent speciation events and edges represent the historical "lifetimes" of the taxa. The leaves, or "tips", represent the current (extant) taxa and a mapping ϕ from the tips of P to the tips of H represents the occurrence of the current parasites on the current host species. For simplicity of exposition, we assume that the two trees have the same number of tips – and thus the same number of total nodes, henceforth denoted n – and that ϕ is a bijection (and thus maps each extant parasite species to a unique extant host species). These assumptions can be relaxed with only modest changes required to the algorithms described here. Figure 1(a) uses dashed lines to illustrate a tip mapping between a host and parasite tree.

The objective of the *Cophylogeny Reconstruction Problem* is to find a mapping Φ of the vertices of P to the vertices and edges of H such that Φ extends ϕ (i.e. is consistent with the observed current tip mapping) and can be constructed from four biologically plausible types of "events": cospeciation, duplication, host switch, and loss. *Cospeciation* maps a node of the parasite tree onto a node of the host tree while *duplication* maps a node of the parasite tree onto an edge of the host tree. A duplication event may also be paired with a contemporaneous *host switch* event in which exactly one of the children of the parasite node "switches" hosts and is mapped to a different subtree in the host tree. The point at which the duplication occurs is called the *takeoff site* and the point at which the switched child is mapped is called the *landing site*. Finally, a *loss* event arises when a parasite p and its child p' are mapped to two locations in the host tree and the edge from p to p' passes through a host node. For the host/parasite pair and tip mapping shown in Figure 1(a), these events are illustrated in two different possible mappings in Figures 1(b) and (c). Each of the four event types is assigned a numerical cost and the objective is to find a mapping of the parasite tree onto the host tree that minimizes the total cost of the

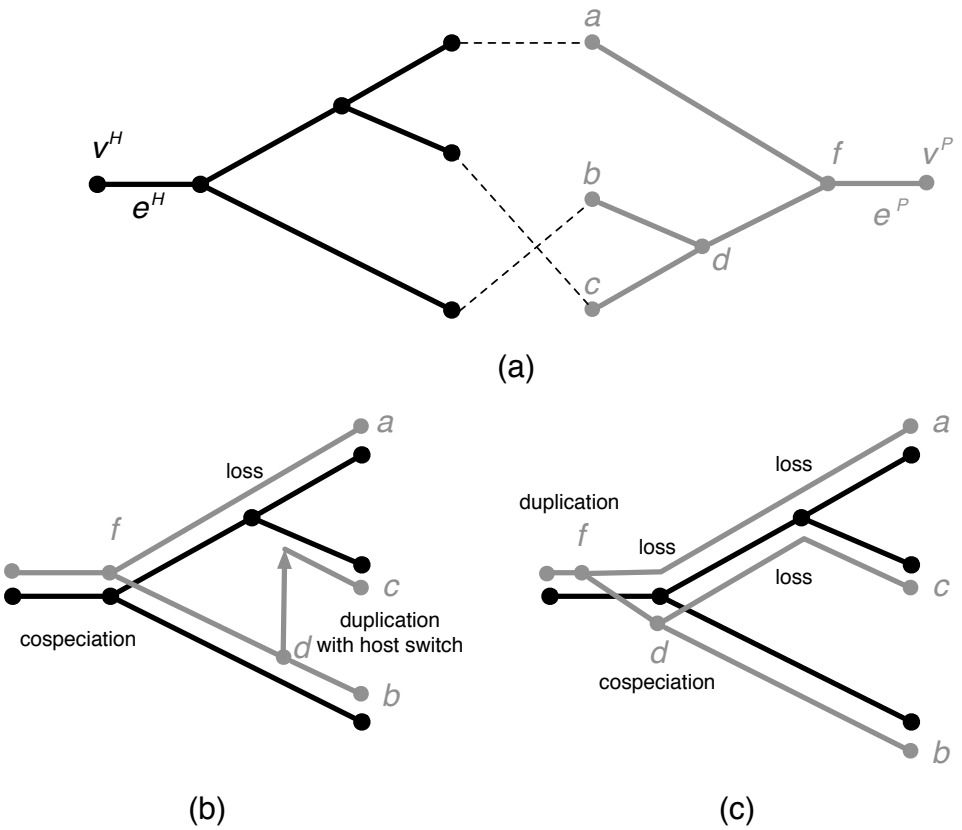

Figure 1: (a) A host tree (black) and parasite tree (gray) with the tip mapping shown with dotted lines. (b) and (c) Two different mappings of the parasite tree onto the host tree with event labeled by type.

events in that mapping.[2]

The Cophylogeny Reconstruction Problem is solvable by a greedy algorithm in time $O(n)$ if host switches are not permitted. However, host switch events are biologically important and cannot be disregarded. Unfortunately, the problem becomes NP-complete in the presence of host switches (Libeskind-Hadas & Charleston 2009, Ovadia et al. 2011). Host switch events can lead to complex timing relationships between events. Figure 2(a) shows how a set of host switch events can result in a solution that is not valid because the times of the events are not reconcilable. In effect, the

[2]Several slightly different schemes are used in the literature and software tools for accounting for event costs.. For simplicity of exposition, and without loss of generality, we use a scheme in which the cost of a host switch includes the cost of the concomitant duplication event.

precedence relationships of the host switches contains a cycle, implying that some event occurs before itself. In this example, event a involves a host switch to b, and thus b occurs at the same time or after a. Event c occurs strictly after b and event d occurs at the same time or after c. Finally, event a occurs strictly after d, and thus by transitivity event a occurs after itself. Such a set of host switch events is called "strongly incompatible." In other cases, a set of host switch events may cause timing inconsistencies that can be resolved by moving the landing sites of one or more host switches to an earlier time at the expense of adding extra loss events. Such a set of switching events is called "weakly incompatible." Figure 2(b) shows a set of associations with weakly incompatible host switches and Figure 2(c) shows how these host switches can be modified to construct a valid mapping. Unfortunately, the problem of finding the optimal way to repair a weakly incompatible solution by moving the landing sites is also known to be NP-complete (Merkle & Middendorf 2005).

The CoRe-PA and Jane tools differ in the way that they deal with host switching and the concomitant incompatibilities that can arise. CoRe-PA uses an "optimistic" approach that allows any host switches to be made. By relaxing the time compatibility constraints on host switches, a dynamic programming algorithm can find an optimal solution in time $O(n^3)$. After the dynamic programming algorithm has found a solution, CoRe-PA tests to see if the solution has timing incompatibilities, in which case it can either discard the solution or attempt to repair it by moving landing sites if it is a weak incompatibility. Since the problem of optimally moving landing sites is NP-compelte, a heuristic must be used for that step, resulting in potentially suboptimal solutions.

In contrast, Jane adds additional constraints to the problem to ensure that no timing incompatibilities can occur. Specifically, Jane imposes an ordering on the relative times of the internal nodes of the host tree. Such an ordering is called a *timing*. Figure 3 shows two different timings for the same host tree. For a given timing we can enforce that a host switch that takes off from one edge of the host tree must land at another edge that is present at the same time, thus ensuring that no timing incompatibilities arise. For a given timing, a $O(n^7)$ dynamic programming algorithm is used to find an optimal solution (Conow et al. 2010). Since there are an exponential number of possible timings, Jane uses a metaheuristic that searches through a sample of timings and finds the optimal solution for each one. Thus, Jane constructs valid, but not necessarily optimal, solutions.

The dynamic programming algorithms employed by TreeFitter, CoRe-PA, and Jane are very similar and are documented in detail in earlier papers (Conow et al. 2010, Merkle et al. 2010, Ronquist 1998). For completeness, however, we sketch the common basic idea here. For simplicity, consider the case that no timing information

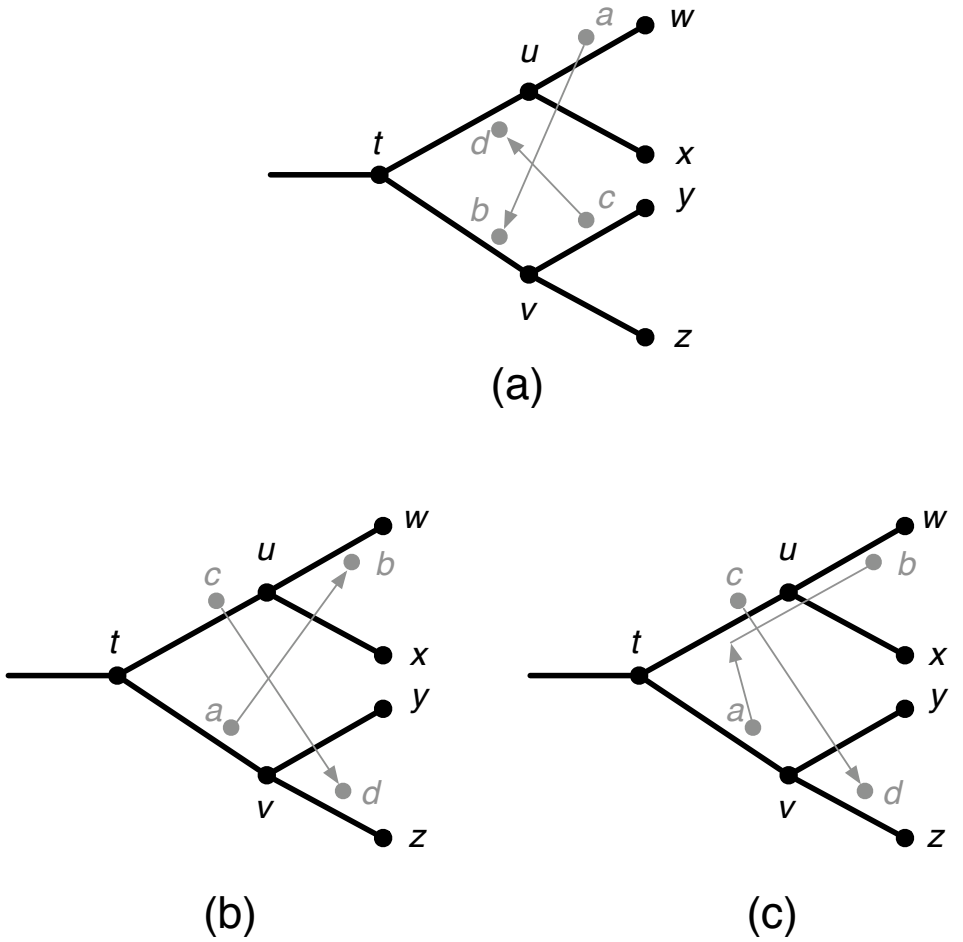

Figure 2: (a) A strongly incompatible set of host switches. (b) A weakly incompatible set of host switches. (c) A modification of the solution in (b) that eliminates the timing incompatibility.

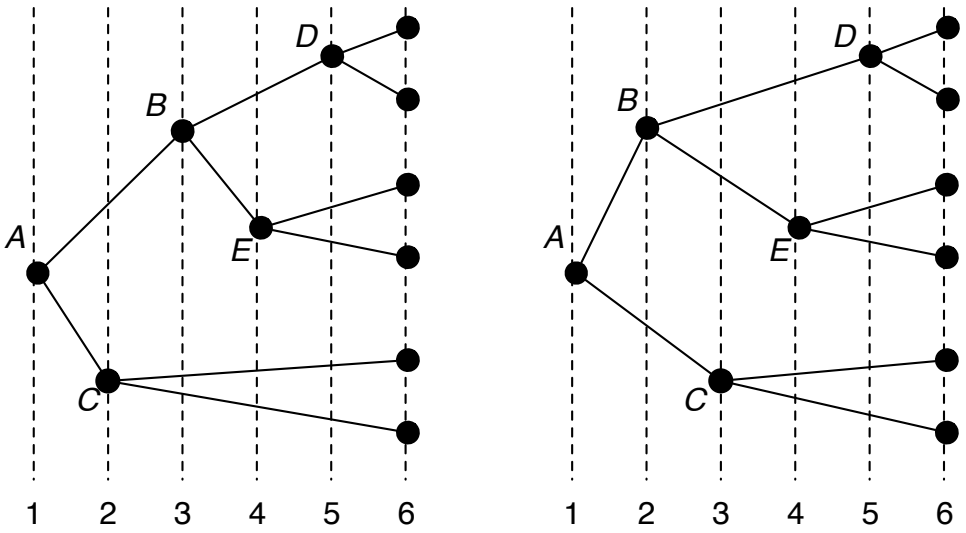

Figure 3: Two different orderings of the same host tree.

is used for the host tree. Let v_p denote a node in the parasite tree. Let e_h denote an edge in the host tree and let v_h denote the end node of that edge (the node representing the end of that lineage). Let $C(v_p, v_h)$ and $C(v_p, e_h)$ denote the cost of an optimal solution for the subproblem comprising the subtree of the parasite tree rooted at node v_p under the assumption that v_p is associated with node v_h or with edge e_h, respectively. The cost function C is, in practice, a dynamic programming table.

The dynamic program begins by computing and storing the base case costs $C(v_p, v_h)$ for all pairs of tips v_p and v_h. If parasite tip v_p is associated with host tip v_h, that is $\phi(v_p) = v_h$, then the cost is 0 and otherwise it is ∞. Now, the DP moves up the parasite tree, considering a node v_p only after the costs have been computed for all descendants of v_p. The algorithm computes $C(v_p, v_h)$ and $C(v_p, e_h)$ as follows: $C(v_p, v_h)$ implies a cospeciation event and is computed as the sum of the cospeciation event cost added to the least cost of placing each of the two children of v_p on descendant vertices or edges of host node v_h. To this end, we examine all of the possible pairs of locations in the subtree of v_h where the two children of v_p could be mapped. For each such placement, the costs of mapping the subtrees of the children of v_p can be found by examining previously computed values of the DP table C since the costs C have already been computed for all descendants of v_p. In addition, some number of losses may be induced by the paths from v_p to each of its two children for this candidate mapping. By iterating over each possible mapping,

the optimal value is found and stored.

Similarly, $C(v_p, e_h)$ corresponds to placing parasite node v_p on host edge e_h, implying that either a duplication or a duplication with host switch occurs at this point. These costs are computed analogously to the cospeciation case: The algorithm examines each possible combination of locations where the two children of v_p could be placed under either a duplication or a duplication with host switch. For each such placement, the costs of those placements are found in the DP table C and the costs of a duplication (or duplication with host switch) are added along with the costs of any losses induced by this placement. By iterating over all possible valid placements, the optimal value of $C(v_p, e_h)$ is computed.

When the DP is complete, we look at all of the cells in the DP table corresponding to placing the root of the parasite tree at all possible locations (vertices and edges) of the host tree. The best of these costs is the optimal cost (although, as mentioned earlier, this solution may have timing incompatibilities).

There are $O(n)$ parasite nodes and $O(n)$ host nodes and edges. Thus, the DP table C has $O(n^2)$ entries to be completed. As each entry is computed, we may need to consider all possible placements of the two parasite children vertices on the host tree. Since the host tree has $O(n)$ vertices and edges, this amounts to checking $O(\binom{n}{2}) = O(n^2)$ table entries, for a total of $O(n^4)$ time. A clever optimization proposed in (Merkle et al. 2010) reduces the running time to $O(n^3)$.

We call this a *vertex-based* dynamic program since it maps vertices of the parasite tree onto the host tree. In contrast, the new faster approach described in the next section is *edge-based*: It operates exclusively on edges, mapping edges of the parasite tree onto edges of the host tree. By operating on edges rather than vertices, optimal solutions can be found much more efficiently. In the next section we describe the edge-based approach for untimed trees, reducing the running time from $O(n^3)$ to $O(n^2)$. In Section 4 we show how this method can be extended to the situation when the host tree has a given timing, reducing the running time to find an optimal mapping from $O(n^7)$ to $O(n^3)$.

3 Edge-Based Dynamic Programming

The edge-based dynamic programming approach seeks to map edges of the parasite tree onto edges of the host tree. Let COST_{co}, COST_{dup}, $\text{COST}_{\text{switch}}$, and $\text{COST}_{\text{loss}}$ denote the non-negative costs associated with cospeciation, duplication, host switch, and loss, respectively. (Since a host switch takes place with a duplication, we assume that $\text{COST}_{\text{switch}}$ is the total cost of both the duplication and the switch and thus $\text{COST}_{\text{switch}} \geq \text{COST}_{\text{dup}}$.) Let e_p denote an edge of the parasite tree. Let v^p denote

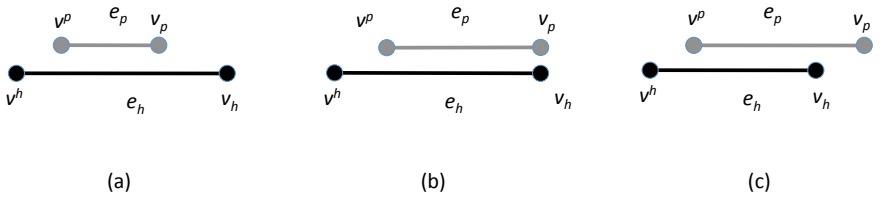

Figure 4: Three ways in which edge parasite edge e_p can be placed on host edge e_h. Time goes from left to right. In placements (b) and (c), edge e_p is alive at v_h. In placement (c), edge e_p continues on some edge beginning at v_h.

the start node of that edge and let v_p denote the end node of that edge. Similarly, let e_h denote an edge of the host tree and let v^h and v_h denote its start and end vertices, respectively. Henceforth, we denote the children edges of e_p (those with start vertex v_p) by e_{p_1} and e_{p_2} and the children edges of e_h (those with start vertex v_h) by e_{h_1} and e_{h_2}. The tree rooted at e_p is defined to be the subtree of the parasite tree comprising edge e_p and all descendant vertices and edges. Note that e_p can be viewed as a handle of the binary tree rooted at the end node of e_p.

We say that parasite edge e_p is placed on host edge e_h if v^p is mapped onto edge e_h at or after v^h but not at the end node v_h. In this case, edge e_p may terminate on edge e_h or may terminate later on some other edge. We say that edge e_p *is alive* at v_h if e_p and v_h overlap. In other words, parasite lineage e_p is present at the time of the host speciation event v_h. In this case, edge e_p may terminate at v_h or may terminate later on some other edge. These scenarios are depicted in Figure 4.

Let $C(e_p, e_h)$ denote the optimal cost of a mapping of the subtree of the parasite tree rooted at e_p such that e_p is placed on e_h. Let $A(e_p, e_h)$ denote the optimal cost of a mapping of the subtree of the parasite tree rooted at e_p such that e_p is placed on e_h *and* edge e_p is alive at v_h. (The name A is intended to indicate the "aliveness" property.) Unlike the vertex-based approach, the edge-based approach uses two DP tables, one for C and one for A. Computing entries for one table requires previously computed entries in the other.

By definition, the cost of an optimal solution is $\min_{e_h \in E_H} C(e^P, e_h)$. In other words, we wish to find the least expensive mapping of the subtree of the parasite tree rooted at its handle, e^P, over all possible edges e_h.

The algorithm computes the C and A tables by considering edges in the parasite tree bottom-up (henceforth referred to as *postorder*): An edge e_p is considered if

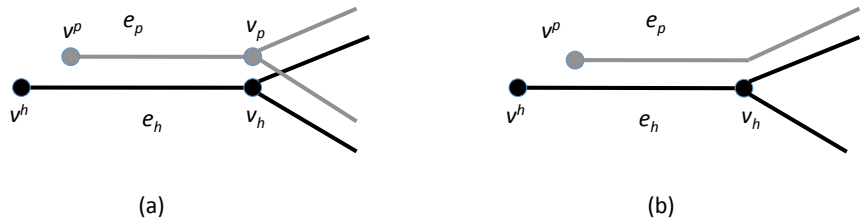

Figure 5: (a) v_p cospeciates with v_h. (b) v_p continues beyond v_h and incurs a loss at v_h.

either v_p is a tip or the children edges e_{p_1} and e_{p_2} have already been considered (and thus all $C(e_{p_i}, \cdot)$ and $A(e_{p_i}, \cdot)$ values have been previously computed for $i \in \{1, 2\}$). For each edge e_p under consideration, we now consider each edge e_h in the host tree in postorder, so that e_h is considered if either v_h is a tip or the children edges e_{h_1} and e_{h_2} have already been considered.

The values $A(e_p, e_h)$ are computed as follows. If v_h is a tip then

$$A(e_p, e_h) = \begin{cases} 0 & \text{if } v_p \text{ is a tip and } \phi(v_p) = v_h \\ \infty & \text{otherwise} \end{cases}$$

Next, consider the case that v_h is not a tip. If e_p is alive at v_h then either e_p terminates at v_h, and thus v_p cospeciates with v_h, or e_p continues beyond v_h and incurs a loss at v_h. These scenarios are illustrated in Figure 5. Therefore,

$$A(e_p, e_h) = \min\{\textsc{Co}(e_p, e_h), \textsc{Loss}(e_p, e_h)\}$$

where

$$\textsc{Co}(e_p, e_h) = \begin{cases} \textsc{cost}_{\text{co}} + \min \left\{ \begin{array}{l} C(e_{p_1}, e_{h_1}) + C(e_{p_2}, e_{h_2}), \\ C(e_{p_1}, e_{h_2}) + C(e_{p_2}, e_{h_1}) \end{array} \right\} & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

and

$$\textsc{Loss}(e_p, e_h) = \begin{cases} \textsc{cost}_{\text{loss}} + \min\{C(e_p, e_{h_1}), C(e_p, e_{h_2})\} & e_p \neq e^P \\ 0 & \text{otherwise} \end{cases}$$

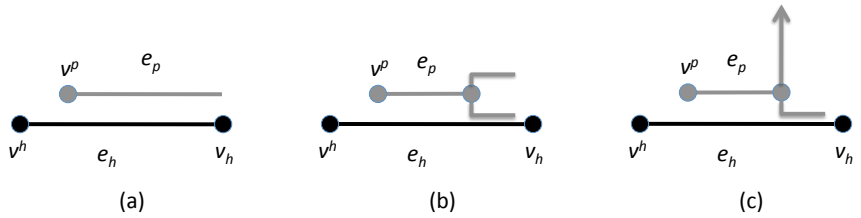

Figure 6: (a) e_p reaches v_h. (b) e_p duplicates on e_h. (c) e_p duplicates with host switch on e_h.

The cospeciation term considers both ways in which the children edges of e_p can be mapped onto the children edges of e_h in a cospeciation event, unless v_p is a tip in which case cospeciation is not possible. The loss term considers both ways in which edge e_p can continue, either on one child of e_h or the other. However, in the special case that e_p is the handle of the parasite tree, e^P, no losses should be incurred since v^P does not correspond to a speciation event.

The values $C(e_p, e_h)$ are computed as follows. If e_p is placed on e_h then e_p may reach v_h, it may duplicate on e_h, or it may duplicate and host switch on e_h. The three cases are illustrated in Figure 6. The best solution in the first case is, by definition, $A(e_p, e_h)$. Thus,

$$C(e_p, e_h) = \min\{A(e_p, e_h), \textsc{Dup}(e_p, e_h), \textsc{Switch}(e_p, e_h)\}$$

where

$$\textsc{Dup}(e_p, e_h) = \begin{cases} \textsc{cost}_{\text{dup}} + C(e_{p_1}, e_h) + C(e_{p_2}, e_h) & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

and

$$\textsc{Switch}(e_p, e_h) = \begin{cases} \textsc{cost}_{\text{switch}} + \min \left\{ \begin{array}{l} C(e_{p_1}, e_h) + \textsc{Best}(e_{p_2}), \\ C(e_{p_2}, e_h) + \textsc{Best}(e_{p_1}) \end{array} \right\} & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

where

$$\textsc{Best}(e) = \min_{e_h \in E_H} C(e, e_h)$$

The host switch term accounts for one of the two child edges of e_p remaining in the subtree rooted at e_h while the other child edge switches to the best possible host

edge; the host edge that incurs the least cost, indicated by the function BEST. Since $C(e_{p_1}, e_h)$ and $C(e_{p_2}, e_h)$ were computed for every e_h in a previous iteration of the dynamic program, we can compute $\text{BEST}(e_{p_1})$ and $\text{BEST}(e_{p_2})$ during those iterations.

The algorithm is summarized in Algorithm 1, where the edges in each of the two trees are visited in postorder to satisfy the requirement that an edge is considered only after its children edges have been considered. The worst-case asymptotic running

```
for each parasite edge e_p ∈ E_P in postorder do
    BEST(e_p) = ∞
    for each host edge e_h ∈ E_H in postorder do
        Compute A(e_p, e_h)
        Compute C(e_p, e_h)
        BEST(e_p) = min{BEST(e_p), C(e_p, e_h)}
    end
end
```

Algorithm 1: The edge-based DP algorithm.

time of this algorithm is $O(n^2)$ since each of the two loops is performed $O(n)$ times and each computation in the inner loop can be performed in constant time by looking up precomputed entries in the DP tables. Once the algorithm is complete, we can find the cost of an optimal solution by computing $\min_{e_h \in E_H} C(e^P, e_h)$ in $O(n)$ time. If we wish to reconstruct an optimal solution, the DP tables can be annotated in the standard way, allowing the solutions to be reconstructed by tracing through the table. Note that if we wish only to compute the optimal cost then table A need not be stored in its entirety since $A(e_p, e_h)$ is only used when computing $C(e_p, e_h)$. Thus, we can discard each entry of A immediately after using it to compute the corresponding entry of C.

4 Edge-Based Dynamic Programming for Host Trees with Timings

In this section we extend the edge-based dynamic program for timed host trees (e.g. as used in Jane). Recall that n denotes the total number of nodes in each of the two trees. Thus, there are $n-1$ nodes in the host tree excluding the handle node v^H. Therefore, there are $n/2-1$ internal nodes in the host tree, excluding v^H. A timing of the host tree is a one-to-one mapping of the integers $1, \ldots, n/2-1$ to the $n/2-1$ internal nodes of the host tree such that each node is assigned a smaller number (i.e.

an earlier "time") than all of its descendants. Thus, a timing represents a possible ordering of the speciation events in the host tree. All of the tips are assumed to occur at current time and thus they all receive time $n/2$. The handle node v^H is assigned time 0.

The objective of the cophylogeny problem with timing is to find a mapping of the parasite tree onto the timed host tree such that for each host switch event, the takeoff and landing sites of the host switch are contemporaneous. In this way, the mapping is assured to contain no timing incompatibilities. Specifically, for each node in the parasite tree, the mapping must specify both its location in the host tree and the integer time t or an open interval $(t-1,t)$ where that node is placed, where $t \geq 1$. While cospeciation events necessarily occur at integer times t (since these are the times associated with host speciation events), without loss of generality all duplications and host switches occur on edges and thus in open intervals $(t-1,t)$. For a host switch with takeoff site in interval $(t-1,t)$ the landing site must be on another edge in the same interval.

We say that a host edge e_h *spans* the time interval $(t-1,t)$ if its starting vertex v^h occurs at or before time $t-1$ and its ending vertex occurs at or after time t. A parasite edge e_p is said to be placed on host edge e_h *just prior to time* t if edge e_h spans the time interval $(t-1,t)$ and the start node v^p of edge e_p is mapped onto edge e_h at time $t-1$ or in the time interval $(t-1,t)$ as depicted in Figure 7. Let $C(e_p,e_h,t)$ denote the optimal cost of a mapping of the subtree of the parasite tree rooted at e_p such that e_p is placed on e_h just prior to time t. Let $A(e_p,e_h,t)$ denote the optimal cost of a mapping of the subtree of the parasite tree rooted at e_p such that e_p is placed on e_h just prior to time t and edge e_p is alive at time t.

By definition, the cost of an optimal solution is $\min_{e_h \in E_H, t \in [1,n/2]} C(e^P,e_h,t)$. In other words, we wish to find the least expensive mapping of the subtree of the parasite tree rooted at its handle, e^P, over all possible edges e_h at all possible times.

We compute the entries of C and A similarly to the untimed case, but now using three nested loops: The outer loop iterates over time in decreasing order. The next loop iterates over parasite edges from bottom-up (postorder). The innermost loop iterates over host edges in postorder. The structure of the algorithm is shown in Algorithm 2 where $\textsc{Best}(e_p,t)$ represents the best possible cost of a solution in which edge e_p is placed on a host edge just prior to time t.

We now examine the computations in detail. The first time t that must be considered is $n/2$, the time associated with tip events. At time $n/2$, the values of A are computed as

$$A(e_p,e_h,n/2) = \begin{cases} 0 & \text{if } v_h \text{ and } v_p \text{ are tips and } \phi(v_p) = v_h \\ \infty & \text{otherwise} \end{cases}$$

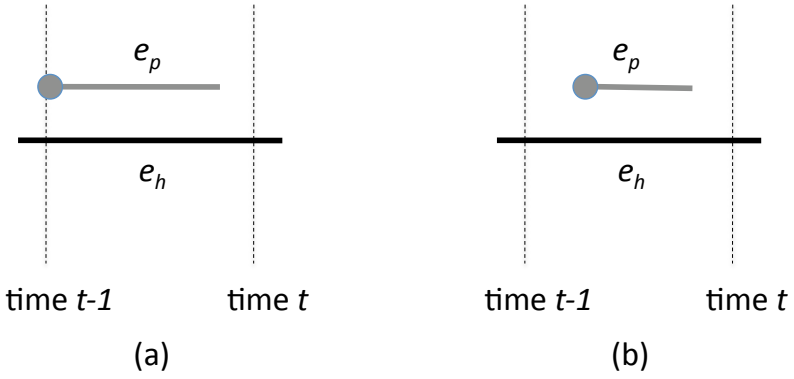

Figure 7: The two ways that parasite edge e_p can be placed on host edge e_h just prior to time t. (a) The edge begins at time $t-1$. (b) The edge begins in the open interval $(t-1, t)$.

```
for each time t in descending order do
    for each parasite edge e_p ∈ E_P in postorder do
        BEST(e_p, t) = ∞
        for each host edge e_h ∈ E_H in postorder do
            Compute A(e_p, e_h, t)
            Compute C(e_p, e_h, t)
            BEST(e_p, t) = min{BEST(e_p, t), C(e_p, e_h, t)}
        end
    end
end
```

Algorithm 2: The edge-based DP algorithm for timed host trees.

In general, $A(e_p, e_h, t)$ is computed using one of two cases, depending on whether v_h occurs at time t or not. If v_h occurs at time t then either v_p occurs at time t (inducing a cospeciation event if v_p is not a tip) or v_p occurs at a later time (inducing a loss at v_h). Thus,

$$A(e_p, e_h, t) = \min(\textsc{Co}(e_p, e_h, t), \textsc{Loss}(e_p, e_h, t))$$

where

$$\textsc{Co}(e_p, e_h, t) = \begin{cases} \textsc{cost}_{\text{co}} + \min \left\{ \begin{array}{l} C(e_{p_1}, e_{h_1}, t+1) + C(e_{p_2}, e_{h_2}, t+1), \\ C(e_{p_1}, e_{h_2}, t+1) + C(e_{p_2}, e_{h_1}, t+1) \end{array} \right\} & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

and

$$\textsc{Loss}(e_p, e_h, t) = \begin{cases} \textsc{cost}_{\text{loss}} + \min\{C(e_p, e_{h_1}, t+1), C(e_p, e_{h_2}, t+1)\} & e_p \neq e^P \\ 0 & \text{otherwise} \end{cases}$$

If v_h does not occur at time t then $A(e_p, e_h, t) = C(e_p, e_h, t+1)$.

Next, we compute $C(e_p, e_h, t)$. Edge e_p may pass through the interval $(t-1, t)$ or may be involved in either a duplication or duplication with host switch in this interval. Thus,

$$C(e_p, e_h, t) = \min\{A(e_p, e_h, t), \textsc{Dup}(e_p, e_h, t), \textsc{Switch}(e_p, e_h, t)\}$$

where

$$\textsc{Dup}(e_p, e_h, t) = \begin{cases} \textsc{cost}_{\text{dup}} + C(e_{p_1}, e_h, t) + C(e_{p_2}, e_h, t) & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

and

$$\textsc{Switch}(e_p, e_h) = \begin{cases} \textsc{cost}_{\text{switch}} + \min \left\{ \begin{array}{l} C(e_{p_1}, e_h, t) + \textsc{Best}(e_{p_2}, t) \\ C(e_{p_2}, e_h, t) + \textsc{Best}(e_{p_1}, t) \end{array} \right\} & \text{if } v_p \text{ is not a tip} \\ \infty & \text{otherwise} \end{cases}$$

where

$$\textsc{Best}(e, t) = \min_{e_h \in E_H} C(e, e_h, t).$$

The worst-case asymptotic running time of this algorithm is $O(n^3)$ since each of the loops is performed $O(n)$ times and each computation in the body of the inner loop can be performed in constant time by looking up precomputed entries in the DP tables. As in the untimed case, the table can be annotated to reconstruct optimal solutions. If only the cost of an optimal solution is desired then table A requires only $O(1)$ space since the value $A(e_p, e_h, t)$ is only used when computing $C(e_p, e_h, t)$.

5 Conclusions

In this paper we have described a new dynamic programming formulation for cophylogeny reconstruction. The approach differs from previous DP solutions by examining the placement of *edges* of the parasite tree onto the host tree rather than placement of *vertices* of the parasite tree onto the host tree. In the case of untimed trees, the running time improves from $O(n^3)$ to $O(n^2)$. In the case of timed trees, the running time improves from $O(n^7)$ to $O(n^3)$. This approach is evidently quite general and can be used in lieu of vertex-based dynamic programming in a number of different heuristics for the cophylogeny reconstruction problem.

Acknowledgements

This work was supported by the U.S. National Science Foundation under grant 0753306 to Harvey Mudd College and by the Howard Hughes Medical Institute under grant 52006301 to Harvey Mudd College. RLH thanks Dr. Michael Charleston for many fruitful discussions on the cophylogeny problem.

References

Charleston, M. (1998), 'Jungles: A new solution to the hostparasite phylogeny reconciliation problem', *Mathematical Biosciences* **149**, 191–223.

Charleston, M. & Page, R. D. M. (n.d.), 'TreeMap'. `http://www.it.usyd.edu.au/~mcharles/software/treemap/treemap.html`.

Conow, C., Fielder, D., Ovadia, Y. & Libeskind-Hadas, R. (2010), 'Jane: A new tool for cophylogeny reconstruction problem', *Algorithms for Molecular Biology* **5**(16). `http://www.almob.org/content/5/1/16`.

CoRe-PA (n.d.). `http://pacosy.informatik.uni-leipzig.de/pv/Software/CoRe-PA/CoRe-PA-index.engl.html`.

Jane (n.d.). `http://www.cs.hmc.edu/~hadas/jane`.

Libeskind-Hadas, R. & Charleston, M. (2009), 'On the computational complexity of the reticulate cophylogeny reconstruction problem', *Journal of Computational Biology* **16**(1), 105–117.

Merkle, D. & Middendorf, M. (2005), 'Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information', *Theory of Biosciences* **123**(4), 277–299.

Merkle, D. & Middendorf, M. (n.d.), 'Tarzan'. `http://pacosy.informatik.uni-leipzig.de/pv/Software/Tarzan/PV-Tarzan.engl.html`.

Merkle, D., Middendorf, M. & Wieseke, N. (2010), 'A parameter-adaptive dynamic programming approach for inferring cophylogenies', *BMC Bioinformatics* **11**.

Ovadia, Y., Fielder, D., Conow, C. & Libeskind-Hadas, R. (2011), 'The cophylogeny reconstruction problem is NP-complete', *Journal of Computational Biology* **18**(1), 59–65.

Ronquist, F. (1995), 'Reconstructing the history of host-parasite associations using generalized parsimony', *Cladistics* **11**, 73–89.

Ronquist, F. (1998), 'Three-dimensional cost matrix optimisation and maximum cospeciation', *Cladistics* **14**, 167–172.

Ronquist, F. (n.d.), 'TreeFitter'. `http://www.ebc.uu.se/systzoo/research/treefitter/treefitter.html`.