APPROXIMATION ALGORITHMS FOR THE FIXED-TOPOLOGY PHYLOGENETIC NUMBER PROBLEM*

MARY CRYAN†, LESLIE ANN GOLDBERG ‡, AND CYNTHIA A. PHILLIPS§

Abstract. In the ℓ-phylogeny problem, one wishes to construct an evolutionary tree for a set of species represented by characters, in which each state of each character induces no more than ℓ connected components. We consider the fixed-topology version of this problem for fixed-topologies of arbitrary degree. This version of the problem is known to be $\mathcal{NP}$-complete for $\ell \geq 3$ even for degree-3 trees in which no state labels more than $\ell + 1$ leaves (and therefore there is a trivial $\ell + 1$ phylogeny). We give a 2-approximation algorithm for all $\ell \geq 3$ for arbitrary input topologies and we give an optimal approximation algorithm that constructs a 4-phylogeny when a 3-phylogeny exists. Dynamic programming techniques, which are typically used in fixed-topology problems, cannot be applied to ℓ-phylogeny problems. Our 2-approximation algorithm is the first application of linear programming to approximation algorithms for phylogeny problems. We extend our results to a related problem in which characters are polymorphic.

* Research Report CS-RR-327, Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom.

† maryc@dcs.warwick.ac.uk. Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom. This work was partly supported by ESPRIT LTR Project no. 20244 — ALCOM-IT.

‡ leslie@dcs.warwick.ac.uk. Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom. Part of this work took place during a visit to Sandia National Laboratories which was supported by University of Warwick Research and Teaching Innovations Grant 0951CSA and by the U.S. Department of Energy under contract DE-AC04-76AL85000. Part of this work was supported by ESPRIT LTR Project no. 20244 — ALCOM-IT.

§ caphill@cs.sandia.gov. Sandia National Laboratories, Albuquerque, NM. This work was performed under U.S. Department of Energy contract number DE-AC04-94AL85000.

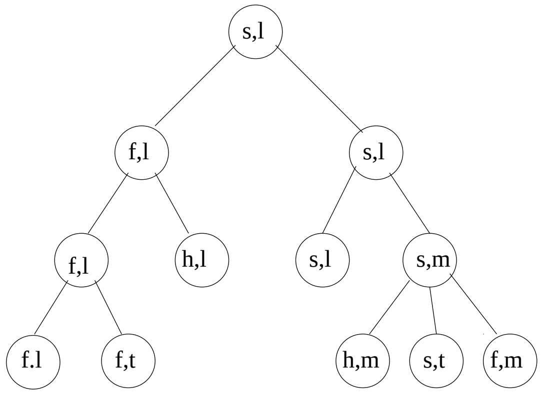

FIG. 1. *An example of a 2-phylogeny. States h and f in the first character and state t in the second character are each in two components.*

1. Introduction. Evolutionary biologists collect information on extant species (and fossil evidence) and attempt to infer the evolutionary history of a set of species. Most mathematical models of this process assume divergent evolution, meaning that once two species diverge, they never share genetic material again. Therefore, evolution is modelled as a tree (*phylogeny*), with extant species as leaves and (extant, extinct, or hypothesized) ancestors as internal nodes. Species have been modelled in several ways, depending upon the nature of available information and the mechanism for gathering that information. Based upon these representations, differing measures of evolutionary distance and objective function are used to evaluate the goodness of a proposed evolutionary tree.

In this paper we assume that input data is character-based. Let S be an input set of n species. A *character* c is a function from the species set S to a set R_c of **states**. The set of species in figure 1 has two characters. The first character represents skin covering and has three states: h for hair, s for scales, and f for feathers. The second character represents size, where t (tiny) means at most one foot long, m (medium) means one to three feet long and l (large) means greater than three feet long. If we are given a set of characters $c_1, \ldots, c_k$ for S, each species is a vector from $R_{c_1} \times \ldots \times R_{c_k}$ and any such vector can represent a hypothesized ancestor. For example, an anaconda (the largest known species of snake) would be represented on these simple characters as (s, l) and a hummingbird would be (f, t). Characters can be used to model biomolecular data, such as a column in a multiple sequence alignment, but in this paper, we think of characters as morphological properties such as coloration or the ability to fly.

Character-based phylogenies are typically evaluated by some *parsimony-like* measure, meaning that the total evolutionary change is somehow minimized. In this paper, we consider the *ℓ-phylogeny* metric introduced in [9]. Given a phylogenetic tree, a character c_i and a state $j \in R_{c_i}$, let ℓ_{ij} be the number of connected components in ${c_i}^{-1}(j)$ (the subtree induced by the species with state j in character i). A phylogeny is an ℓ-phylogeny if each state of each character induces no more than ℓ connected components. That is, $\max_{c_i, j \in R_{c_i}} \ell_{ij} \leq \ell$. More generally, if $r = max_{1 \leq i \leq k} |R_{c_i}|$ and we provide a value ℓ_j for each $1 \leq j \leq r$, then we say a phylogeny is an $(\ell_1, \ \ldots, \ \ell_r)$-phylogeny if for every $1 \leq i \leq k$ and every $j \leq |R_{c_i}|$, the number of connected components for the j-th state of c_i is at most ℓ_j ($max_{1 \leq i \leq k, \ j \in R_{c_i}} \ell_{ij} \leq \ell_j$, for every $1 \leq j \leq r$). The *ℓ-phylogeny problem* is to determine if an input consisting of a species set S and a set of characters $c_1, \ldots, c_k$ has an ℓ-phylogeny. The *phylogenetic number* problem is to determine the minimum ℓ such that the input has an ℓ-phylogeny. The *generalized ℓ-phylogeny problem*, which was introduced in [9], is to determine for some $\ell = (\ell_1, \ldots, \ell_r)$ if an input consisting of a species set S and a set of characters $c_1, \ldots, c_k$ has an $(\ell_1, \ldots, \ell_r)$-phylogeny.

The classic parsimony problem is to find a tree that minimizes the total number of evolutionary changes: $\sum_{c_i, j \in R_{c_i}} \ell_{ij}$. The compatibility problem is to maximize the number of characters that are *perfect*, meaning that all states of that character induce only one connected component. Thus the compatibility problem is to maximize $|\{c_i : \ell_{ij} = 1 \text{ for all } j \in R_{c_i}\}|$. A 1-phylogeny is called a *perfect phylogeny.* All three problems (ℓ-phylogeny for $\ell \geq 1$, parsimony, compatibility) are $\mathcal{NP}$-complete [1, 9, 4, 5, 6, 8, 18]. Papers [4, 5, 8] prove that different restrictions of the classic parsimony problem are $\mathcal{NP}$-complete[1]. Parsimony, ℓ-phylogeny, and compatibility all allow states of a character to evolve multiple times. However, both parsimony and compatibility allow some characters to evolve many times. The ℓ-phylogeny metric requires *balanced* evolution, in that no one character can pay for most of the evolutionary changes. Thus, ℓ-phylogeny is a better measure than parsimony or compatibility in biological situations in which all characters are believed to evolve slowly.

In this paper we consider the *fixed-topology* variant of the ℓ-phylogeny problem, where in addition to the species set and characters, we are also given a tree T in which internal nodes are unlabelled, each leaf is labelled with a species $s \in S$ and each species $s \in S$ is the label of exactly one leaf of T. The *fixed-topology ℓ-phylogeny problem* is the problem of determining labels for the internal nodes so that the resulting phylogeny is an ℓ-phylogeny, or determining that such a labelling does not exist. In figure 1, the hypothesized ancestor (s, m) labels one node. This example is a 2-phylogeny.

Fixed-topology algorithms can be used as *filters.* Current phylogeny-producing software can generate thousands of trees which are (approximately) equally good under some metric such as maximum likelihood or parsimony. We can think of these outputs as proposed topologies. One way to differentiate these hypotheses is to see which topologies also have low phylogenetic number. For example, the original trees can be generated by biomolecular sequence data, and they can then be filtered using morphological data with slowly-evolving traits.

It will be convenient to allow a node to remain unlabelled in one or more characters in a fixed topology. In this case, the node disagrees with all of its neighbors on all unlabelled characters. We can easily extend such a labelling to one in which every node is labelled without increasing ℓ_{ij} for any i or j: for any character c_i, for each connected component of nodes which are not labelled, choose any neighbouring node v which is labelled i_v and label the entire component with the state i_v. This does not introduce any extra component for i_v, nor does it break components of any other state that weren't already broken.

In the fixed-topology setting, optimal trees for the parsimony and compatibility metrics can be found in polynomial time [7]. The fixed-topology ℓ-phylogeny problem can be solved in polynomial time for $\ell \leq 2$, but is $\mathcal{NP}$-complete for $\ell \geq 3$ even for degree-3 trees in which no state labels more than $\ell + 1$ leaves (and therefore there is a trivial $\ell + 1$ phylogeny) [9]. Fitch's algorithm for parsimony uses dynamic

[1] Wareham[21] describes and corrects a minor error in the reduction used by Day in [4] to show the $\mathcal{NP}$-completeness of the problem for Wagner characters.

programming. Dynamic programming also gives good algorithms in some cases for finding phylogenies when characters are polymorphic [2].

Jiang, Lawler, and Wang [13] consider the fixed-topology tree-alignment problem, where species are represented as biomolecular sequences, the cost of an edge in the tree is the edit distance between the labels at its endpoints, and the goal is to minimize the sum of the costs over all edges. They give a 2-approximation for bounded-degree input topologies and extend this to obtain a polynomial-time approximation scheme (PTAS). In Lemma 3 of [13], they prove that the best lifted tree (in which the label of each internal node is equal to the label of one of its children) is within a factor of 2 of the best tree with arbitrary labels. The proof only uses the triangle inequality (it does not use any other facts about the cost measure). Therefore, the result holds for several other cost measures, including ℓ-phylogeny, parsimony, and the minimum-load cost measure for phylogenies with polymorphic characters which was introduced in [2]. It also holds for the *generalized ℓ-phylogeny problem,* where ℓ_j is specified separately for the jth character state. In fact, Lemma 3 of [13] holds for the fixed-topology problem with *arbitrary* input topologies, though the authors do not state this fact since they do not use it. Despite the applicability of Lemma 3, the algorithmic method of Jiang, Lawler and Wang does not seem to be useful in developing approximation algorithms for the fixed-topology ℓ-phylogeny problem (or for related problems). Jiang et al. use dynamic programming to find the minimum-cost lifted tree. Dynamic programming is not efficient for the more global metric of ℓ-phylogeny. The dynamic programming proceeds by computing an optimal labelling for a subtree for each possible labelling of the root of the subtree. For metrics where cost is summed over edges (such as parsimony or tree alignment), one only needs to find the lowest-cost labelling for a given root label. For the ℓ-phylogeny problem, the cost of a tree depends upon how many times each state is broken for a given character. One cannot tell *a priori* which state will be the limiting one. Therefore, instead of maintaining a single optimal tree for each root label, we must maintain all trees whose cost (represented as a vector of components for each state) is undominated. This number can be exponential in r, the number of states, even for bounded-degree input trees. This is a common theme in combinatorial optimization: the more global nature of minimax makes it harder to compute than summation objectives, but also more useful.

Gusfield and Wang [19] take the approach of [13] a step further by proving that the best uniform lifted tree (ULT) is within a factor of 2 of the best arbitrarily-labelled tree. In a uniform lifted tree on each level, all internal nodes are labeled by the same child (e.g. all nodes at level one take the label of their leftmost child). This proof also extends to the ℓ-phylogeny metric. If the input tree is a complete binary tree, then there are only n ULTs, and exhaustive search is efficient, giving an algorithm which is faster than ours and has an equivalent performance bound. However, when the input tree isn't complete (even if it is binary), Gusfield and Wang use dynamic programming to find the minimum-cost ULT, and so their method fails when it is applied to the ℓ-phylogeny problem. Wang, Jiang, and Gusfield recently improved the efficiency of their PTAS for tree alignment [20], but still use dynamic programming.

We give a simple 2-approximation for the fixed-topology ℓ-phylogeny problem that works for arbitrary input topologies. It is based on rounding the linear-programming relaxation of an integer programming formulation for the fixed-topology ℓ-phylogeny problem. The technique of obtaining approximation algorithms for NP-complete problems by rounding the results of the linear-programming relaxation of the original problem has been used successfully for many traditional problems (see [11, 12, 14, 17]). To our knowledge, this is the first application of linear-programming technology to phylogeny problems.

As we described earlier, ℓ-phylogeny is most appropriate for slowly-evolving characters. It is most restrictive (and hence most different from parsimony) when ℓ is small. Therefore, we look more closely at the first NP-hard case: $\ell = 3$. For this case, we give an algorithm based upon the structure of a 3-phylogeny that will construct a 4-phylogeny if the input instance has a 3-phylogeny.

The remainder of our paper is organized as follows: in section 2, we give the 2-approximation algorithm for the ℓ-phylogeny problem. In section 3 we give the optimal approximation algorithm for inputs with 3-phylogenies. In section 4, we extend the linear-programming-based techniques to develop an approximation algorithm for the problem of finding parsimonious low-load labellings for phylogenies with polymorphic characters.

2. A 2-approximation algorithm for the fixed-topology phylogenetic number problem. The interaction between characters in phylogeny problems affects the choice of the topology, but it does not affect the labelling of the internal nodes once the topology is chosen. Thus, for this problem, we can consider each character separately.

Let $c : S \to \{1, \ldots, r\}$ be a character and let T be a tree with root q and leaves labelled by character states $1, \ldots, r$. For each state i, let T_i be the subtree of T consisting of all the leaves labelled i and the minimum set of edges connecting these leaves. Let $L(T_i)$ be the set of leaves of T_i, and let rt_i, the root of T_i, be the node of T_i closest to the root of T. The *important nodes* of T_i are the leaf nodes and the nodes of degree greater than 2. An *i-path* p of T_i is a sequence of edges of T_i that connects two important nodes of T_i, but does not pass through any other important nodes. The two important nodes are referred to as the *endpoints* of p, and the other nodes along the i-path are said to be *on* p (an i-path need not have any nodes on it). Although the edges of the tree T are undirected, we will sometimes use the notation $(v \to w)$ for an edge or i-path with endpoints v and w, to indicate that v is nearer to the root of T than w (v is the *higher* endpoint and w is the *lower* endpoint); otherwise we will write edges and i-paths as (v, w). If the lower endpoint w is labelled i and the label of the upper endpoint v or some node on the i-path $p = (v \to w)$ is not i, then we say that p *breaks* state i. If an i-path goes through the (degree-2) root of T_i, then *both* endpoints are considered lower endpoints.

Given a tree T with each node labeled from the set $\{1, \ldots, r\}$, we need a way to count the number of components induced by the nodes labeled i. Since the tree is rooted, we can assign each connected component a root, namely the node closest to the root of T. We then count the number of roots for components labelled i. A node is the root of its component if its label differs from that of its parent. The root q, which has no parent, is also the root of its component. Therefore we have the following:

OBSERVATION 2.1. *Let T be a tree with its leaves and internal nodes labelled by elements of $\{1, \ldots, r\}$. For each i, let T_i be defined as above, and let q be the root of tree T. Then the number of connected components induced by the nodes labelled i is $|\{e = (v \to w) : \ c(v) \neq i, \ c(w) = i\}| + Y_i$, where $Y_i = 1$ if q is labelled i and 0 otherwise.*

We now define an integer linear program (ILP) which solves the fixed-topology

ℓ-phylogeny problem. The linear-programming relaxation of this ILP is the key to our 2-approximation algorithm. The integer linear program $\mathcal{I}$ uses the variables $X_{v,i}$, for each state $i \in \{1,\ldots,r\}$, and each node v in the tree T, the variables $X_{p,i}$ for each state i, and each i-path p of T_i and the variables $cost_{p,v,i}$ for each state i, i-path p in T_i and each lower endpoint v of path p. Recall that each path has one lower endpoint except when there is an i-path through a degree-2 root, in which case both endpoints are lower endpoints. These variables have the following interpretation:

$$X_{v,i} = \begin{cases} 1 & \text{if node } v \text{ is labelled } i \\ 0 & \text{otherwise} \end{cases}$$

$$X_{p,i} = \begin{cases} 1 & \text{if all nodes on } p \text{ are labelled } i \\ 0 & \text{otherwise} \end{cases}$$

$$cost_{p,v,i} = \begin{cases} 1 & \text{if lower endpoint } v \text{ of } p \text{ is the root of a component of state } i \\ 0 & \text{otherwise} \end{cases}$$

ILP $\mathcal{I}$ is defined as follows:

$$\text{minimize } \ell$$

subject to

$$\begin{array}{llcll}
(1) & X_{v,i} & = & 1 & \text{for each leaf } v \in T_i,\ i = 1,\ldots,r \\
(2) & X_{v,i} & = & 0 & \text{if } v \notin T_i,\ i = 1,\ldots,r \\
(3) & \sum_{i=1}^{r} X_{v,i} & \leq & 1 & \forall v \in T \\
(4) & X_{p,i} & = & X_{v,i} & i = 1,\ldots,r,\ \forall p \in T_i,\ \forall v \in p \\
(5) & X_{p,i} & \leq & X_{v,i} & i = 1,\ldots,r,\ \forall p \in T_i, \text{endpoint } v \in p \\
(6) & cost_{p,v,i} & \geq & X_{v,i} - X_{p,i} & i = 1,\ldots,r,\ \forall p \in T_i, \text{lower endpoint } v \in p \\
(7) & \sum_{p,v} cost_{p,v,i} + X_{rt_i,i} & \leq & \ell & i = 1,\ldots,r \\
(8) & X_{v,i}, X_{p,i}, cost_{p,v,i} & \in & \{0,1\} &
\end{array}$$

Constraint (8) assures that the cost ($cost_{p,v,i}$), i-path ($X_{p,i}$), and vertex ($X_{v,i}$) variables serve as indicator variables in accordance with their interpretation. Constraint (1) labels the leaves in accordance with the input. Constraint (2) prohibits labelling a node v with a state i when v is not in T_i (the number of components labelled i could not possibly be reduced by this labelling). Constraint (3) ensures that each internal node will have no more than one label. Constraints (4) and (5) ensure that for each tree T_i, nodes on paths are taken all-or-none; if any node on an i-path p (including endpoints) is lost to a state i, then it does no good to have any of the other nodes on the path (though it may be beneficial to maintain one or both endpoints). Constraint (6) computes the path costs (counts roots) and constraint (7) ensures that each state has no more than ℓ connected components. This is an implementation of Observation 2.1. Since there is no i-path in T_i with rt_i as its lower endpoint, we must explicitly check the root of each tree T_i, just as we checked the global root in Observation 2.1.

Integer program $\mathcal{I}$ solves the fixed-topology ℓ-phylogeny problem. We will now show that the optimal value of ℓ given by $\mathcal{I}$ is a lower bound on the phylogenetic number of tree T with the given leaf labelling.

PROPOSITION 2.2. *If there exists an ℓ-phylogeny for tree T with a given leaf labelling, then there is a feasible solution for the integer linear program for this value of ℓ.*

Proof. Suppose there exists an ℓ-phylogeny on the tree T with leaves and internal nodes labelled from $\{1, \ldots, r\}$. Consider one particular ℓ-phylogeny, and assume without loss of generality that all node labels are useful for connectivity (i.e. changing the label of node v from i to something else will increase the number of components labelled i). This may require some nodes to be unlabelled. We obtain a feasible solution to $\mathcal{I}$ as follows. Set variable $X_{v,i}$ to 1 if node v is labelled i in this phylogeny and 0 otherwise. Set $X_{p,i}$ to 1 if both endpoints and all internal nodes of i-path p are labelled i and 0 otherwise. Set $cost_{p,v,i} = 1$ if lower endpoint v of p is labelled i and the i-path is not, and set $cost_{p,v,i} = 0$ otherwise. We now show this assignment is a solution to $\mathcal{I}$.

The $X_{v,i}$, $X_{p,i}$, and $cost_{p,v,i}$ variables are binary by construction, thus satisfying Constraint (8). By construction, Constraint (1) will be satisfied by our assignment. Constraint (2) will also be satisfied, because it is never useful to label nodes outside T_i with i, and we have assumed all the labels on nodes are useful for connectivity. Constraint (3) is also satisfied because each node of the phylogeny will be labelled with at most one state. Constraints (4) and (5) are satisfied because the condition that all labelled nodes are necessary for connectivity ensures that a node on an i-path will only be labelled i if all the nodes and endpoints of the i-path are labelled i. Constraint (6) is satisfied by construction.

To show that constraint (7) is satisfied, consider the connected components for i; by our assumption, these all lie in T_i. Let $\gamma = \{e = (v \to w) : c(v) \neq i,\ c(w) = i\}$. By Observation 2.1 we have $|\gamma| + X_{q,i} \leq \ell$, where q is the root of T. To calculate $(\sum_{p,v} cost_{p,v,i}) + X_{rt_i,i}$, note that $cost_{p,w,i} = 1$ if and only if $X_{w,i} = 1$ for the lower endpoint w and $X_{p,i} = 0$ and otherwise $cost_{p,w,i}$ is 0. By our definitions above, $X_{p,i} = 0$ and $X_{w,i} = 1$ if and only if the edge $(v, w) \in T$ from w's parent (on the i-path p or its upper endpoint) into w has $c(v) \neq i$ and $c(w) = i$. Furthermore, this is the only edge on the i-path with this property (the cost of each other edge is 0) unless path p passes through a degree-2 root and both its endpoints have breaks. In the latter case there is a second endpoint w' such that $cost_{p,w',i} = 1$. Since the i-paths partition T_i, each i-path p and lower endpoint v with $cost_{p,v,i} = 1$ contains one element of γ which is unique to that i-path and lower endpoint. Thus $(\sum_{p,v} cost_{p,v,i}) \leq |\gamma| \leq \ell$. If rt_i is the node q, then $X_{rt_i,i} = X_{q,i}$ and $(\sum_{p,v} cost_{p,v,i}) + X_{rt_i,i} \leq |\gamma| + X_{q,i} \leq \ell$. Otherwise, if rt_i is not the global root q, by our assumption that only useful nodes of T are labelled with i, the ancestor node a_i of rt_i is not labelled i. Then, if $X_{rt_i,i} = 1$ the edge $e = (a_i \to rt_i)$ contributes 1 to $|\gamma|$, and therefore $(\sum_{p,v} cost_{p,v,i}) + X_{rt_i,i} \leq |\gamma| + X_{q,i} \leq \ell$. Hence constraints (7) are satisfied and we have a solution for the integer program $\mathcal{I}$. ☐

Integer linear programming in $\mathcal{NP}$-hard in general [3, 10, 15], so we cannot solve it directly in polynomial time. (In fact, doing so would solve the fixed-topology ℓ-phylogeny problem, which we know to be $\mathcal{NP}$-hard for $\ell \geq 3$ from [9].) However, we can solve the linear-programming relaxation $\mathcal{L}$ of $\mathcal{I}$, which consists of all the constraints of $\mathcal{I}$ except that Constraint (8) is replaced by the constraint $0 \leq X_{v,i}, X_{p,i}, cost_{p,v,i} \leq 1$ (8′).

THEOREM 2.3. *If there is a solution for the linear program $\mathcal{L}$ for a fixed topology T with leaves labelled with states from $\{1,\ldots,r\}$, then we can assign states to the internal nodes of T such that no state $i \in \{1,\ldots,r\}$ has more than 2ℓ components.*

Proof. The 2ℓ phylogeny for the character $c : S \rightarrow \{1,\ldots,r\}$ on T is constructed by assigning states to the nodes of each tree T_i based on the $X_{v,i}$ values. For each state $i \in \{1,\ldots,r\}$, consider each internal node v of T_i. A node v is labelled i if and only if $X_{v,i} > 1/2$, and there is a path $v, w_1, w_2, \ldots, w_k, v^*$ through tree T_i to a leaf v^* of T_i where $X_{w_j,i} > 1/2$ for all $j = 1,\ldots,k$. If $X_{v,i} > 1/2$, but there is no such path, then node v is *isolated*, and by our procedure remains unlabelled. A node v also remains unlabelled if $X_{v,i} \leq 1/2$ for all states i.

To show that the labelling is a 2ℓ-phylogeny, we show that each component of state i adds at least $1/2$ to the sum $(\sum_{p,v} cost_{p,v,i}) + X_{rt_i,i}$. From Observation 2.1, the number of connected components for the state i is $|\{e = (v \rightarrow w) : c(v) \neq i,\ c(w) = i\}| + Y_i$, where Y_i is 1 if q has state i (and therefore $q = rt_i$) and 0 otherwise. Constraints (5) and (4) ensure that if the edge $e = (v \rightarrow w)$ has $c(v) \neq i$ and $c(w) = i$ then either w is the root of T_i, or w must be an endpoint node with $X_{w,i} > 1/2$, and that either $X_{v,i} \leq 1/2$ or v is isolated. However, since w is labelled i, w must not be isolated, and therefore v would not be isolated if $X_{v,i}$ was greater than $1/2$. So $X_{v,i} \leq 1/2$, and $X_{p,i} \leq 1/2$ for the i-path p with lower endpoint w. Therefore we need only calculate the number of lower endpoints w from i-paths p such that $X_{p,i} \leq 1/2$, $X_{w,i} > 1/2$, and w is not isolated.

Suppose w is a lower endpoint of i-path p. Since w is not isolated and the node above w is not labelled i, there is a sequence $p_1 = (w \rightarrow v_1),\ p_2 = (v_1 \rightarrow v_2), \ldots, p_j = (v_{j-1} \rightarrow v_j)$ of i-paths of T_i such that $X_{s,i} > 1/2$ for every $s \in \{p_1,\ldots,p_j\}$ and $X_{v,i} > 1/2$ for every $v \in \{v_1,\ldots,v_j\}$, and v_j is a leaf of T_i. Calculating $cost_{p,w,i} + cost_{p_1,v_1,i} + \ldots + cost_{p_j,vj,i} = (X_{w,i} - X_{p,i}) + (X_{v_1,i} - X_{p_1,i}) + \ldots + (X_{v_j,i} - X_{p_j}) = -X_{p,i} + (X_{w,i} - X_{p_1,i}) + (X_{v_1,i} - X_{p_2}) + \ldots + (X_{v_{j-1},i} - X_{p_j,i}) + X_{v_j,i}$, we know by constraints (5) that $X_{w,i} - X_{p_1,i} \geq 0,\ X_{v_1,i} - X_{p_2,i} \geq 0,\ \ldots,\ X_{v_{j-1},i} - X_{p_j,i} \geq 0$. So $cost_{p,w,i} + cost_{p_1,v_1,i} + \ldots + cost_{p_j,vj,i} \geq X_{v_j,i} - X_{p,i} = 1 - X_{p,i} \geq 1/2$.

Note that for any two breaks that appear at lower endpoints w and w' of i-paths p and p' respectively, the i-labelled paths to leaves are disjoint (because they are in separate components of i). Therefore each break of i at a lower endpoint w contributes at least $1/2$ to the sum $(\sum_{p,v} cost_{p,v,i})$. If rt_i is labelled i (and hence the root of a component of i), then $X_{rt_i,i} > 1/2$ (corresponding to an edge $(v \rightarrow rt_i)$ in T or to the case $Y_i = 1$). So $2 \times ((\sum_{p,v} cost_{p,v,i}) + X_{rt_i,i}) \geq |\{e = (v \rightarrow w) : c(v) \neq i,\ c(w) = i\}| + Y_i$, and therefore $2\ell \geq |\{e = (v \rightarrow w) : c(v) \neq i,\ c(w) = i\}| + Y_i$. ◻

Theorem 2.3 is tight as shown by the following example: let the input topology be a star graph with $2x$ leaves: x leaves labelled i and x leaves labelled j. The LP solution has the root labelled half i and half j, so that $\ell = (x+1)/2$ by constraint 6. The optimal solution has $\ell = x$, arbitrarily close to twice the LP bound. In this example, however, it is the LP bound which is loose, and therefore our analysis of the approximation quality of the algorithm may not be tight.

Recall that we have considered each character separately in our 2-approximation algorithm. Thus, our work applies to the generalized ℓ-phylogeny problem (and not just to the ordinary ℓ-phylogeny problem). In particular, we have the following theorem.

THEOREM 2.4. *There is a 2-approximation algorithm for the generalized ℓ-phylogeny problem.*

3. 4-phylogeny algorithm. In this section we give an algorithm which takes a fixed-topology phylogeny instance with arbitrary topology and, as long as it has a 3-phylogeny, finds a 4-phylogeny for the instance.

We use the following definitions, in addition to those that we used for the 2-approximation. We will maintain a forest F_i for every state i, which corresponds to the set of nodes that state i is contending for. A **branch point** of F_i is a node in F_i with degree 3. We say that a node $v \in F_i$ is *claimed* by state i if it is not in F_j for any $j \neq i$.

The algorithm generalizes the fixed-topology 2-phylogeny algorithm of [9]. It consists of a *forced phase* and then an *approximation phase.* The forced phase produces a partial labelling (resolution of labels on some subset of the nodes) which can still be extended to a 3-phylogeny; it makes no labelling decisions that are not forced if one is to have a 3-phylogeny. The approximation phase removes all remaining contention for labels, but it can break some states into four pieces. Because finding a fixed-topology 3-phylogeny is $\mathcal{NP}$-complete [9], this is an optimal approximation algorithm for phylogeny instances with 3-phylogenies.

3.1. The Forced Phase of the Algorithm. Initially, for every state i we will have $F_i = T_i$. During the forced phase of the algorithm, nodes will be removed from the forests F_i. The invariant during the forced phase of the algorithm is that there is a 3-phylogeny in which every node v is assigned a state j such that $v \in F_j$. The forced phase applies the following rules in any order until none can be applied. If any forest F_i is broken into more than three components by the application of these rules, then the instance has no 3-phylogeny and the algorithm terminates.

1. For any i-path (v, w), let S be the set containing v and w and the nodes on the i-path. If S contains two or more branch points of F_j (for $i \neq j$) then every node on the i-path is removed from F_i. Note that in the updated copy of F_i (after the rule is applied), v and w will have lower degree than in the original F_i. Furthermore, if v has degree 2 in the updated F_i then the i-path containing it will consist of nodes from two different i-paths in the original F_i. Similarly, i-paths can be merged as a result of the following rules.
2. If F_i has $C(F_i)$ connected components and F_i contains a node v of degree at least $5 - C(F_i)$ then in every forest F_j with $j \neq i$, v and all nodes on j-paths adjacent to v are removed from F_j (i.e., i claims node v).
3. Suppose v is a branch point of F_i but not a branchpoint of F_j, and suppose two i-paths (v, w_1) and (v, w_2) adjacent to v each contain a branch point of F_j. Then in every forest F_k with $k \neq i$, every branch point $w \notin \{v, w_1, w_2\}$ of F_i and every node on every k-path adjacent to w is removed from F_k (i.e., F_i claims all branchpoints except v, w_1, and w_2).

Rule 1 is justified by observing that in any 3-phylogeny, each forest F_i gives up at most two disjoint i-paths, or a single branchpoint with the i-paths adjacent to it. In the setting in which rule 1 is applied, if F_i were to claim the path in question, then F_j would lose two branchpoints and necessarily be in at least four components. Therefore, in any 3-phylogeny for the input, F_i cannot have that i-path. Note that once any node on an i-path is lost to F_i, then F_i has no reason to claim any other nodes on the i-path.

Rule 2 is justified by the following observations. If there is a node of degree at least 4 in tree T_i, then it must be labelled i in any 3-phylogeny (losing it will break state i

into at least 4 pieces). Once F_i has been forced to give up an i-path ($C(F_i) = 2$), it cannot give up another branchpoint. Finally, if $C(F_i) = 3$ then i must claim all remaining nodes in F_i.

Rule 3 is applied when we isolate a region where a break in F_i must occur, but do not yet know exactly where the break will occur. If two paths adjacent to a branchpoint of F_i contain branchpoints of F_j, then by the previous argument for rule 1, F_i cannot keep both of those paths. Therefore, outside of the affected region (those two i-paths), F_i can act as though the forest has been cut into at least two pieces, and can claim all branchpoints.

3.2. The Approximation Phase of the Algorithm. In the following, *releasing* a degree-2 node $v \in F_i$ removes all nodes on its i-path from F_i. Releasing a higher-degree node $v \in F_i$ removes v and all nodes on i-paths adjacent to v from F_i. The approximation phase consists of the following steps.

1. For each connected component C of F_i, if the root of C is unclaimed then F_i releases the root of C. Also, if this root has degree 2, F_i releases any unclaimed branch points at the ends of the i-path through this root.
2. If, after the forced phase, F_i is in a single component with exactly one unclaimed branch point, w, then it releases w.
3. If, after the forced phase, F_i is in a single component with exactly two unclaimed branch points, w_1 and w_2 which are the two endpoints of an i-path, and the path from the root to w_2 passes through w_1, then F_i releases w_2.
4. If, after the forced phase, F_i is in a single component with exactly three unclaimed branch points, w_1, w_2 and w_3 where there is an i-path from w_1 to w_2 and an i-path from w_2 to w_3, then F_i releases w_2.

3.3. The Proof of Correctness. The proof of correctness of the algorithm requires the following observation, and follows from Lemma 3.2 and Lemma 3.7.

OBSERVATION 3.1. *If F_i is in one component and it releases two branchpoints w_1 and w_2 which share an i-path, then the resulting forest F_i has at most 4 components.*

Proof. Suppose without loss of generality that branchpoint w_1 is removed first. This leaves F_i in three pieces. Because w_2 shares an i-path with w_1, this operation reduces the degree of w_2 to two, so the two remaining i-paths adjacent to w_2 are merged. Subsequently removing the i-path through w_2 adds only one more component. □

LEMMA 3.2. *At the end of the approximation phase, every forest F_i has at most 4 connected components.*

Proof. The forest F_i can be in at most three components at the end of the forced phase. If F_i is in three components at the end of the forced phase, then, by Rule 2 of the forced phase, every remaining node in F_i is claimed during the forced phase, so nothing is removed from F_i during the approximation phase. If F_i is in two components after the forced phase, then, again by Rule 2, all branch points of F_i are claimed during the forced phase, so no branch points are removed from F_i during the approximation phase. Step 1 of the approximation phase, therefore, will remove at most one path from each component (when the root has degree 2, since degree-3 roots are claimed) and therefore breaks F_i into at most four components. In this case Steps 2–4 of the approximation phase do not apply, and at most one of Steps 2–4 can apply to each of the remaining cases.

If F_i is in one component after the forced phase, and Steps 2–4 do not apply then we have two cases. If the root is degree three, Step 1 results in at most 3 components. If the root is degree two, then F_i could release the two branchpoints on either end of this i-path, resulting in at most four components by Observation 3.1.

Suppose F_i is in one component with exactly one unclaimed branchpoint after the forced phase. If that branchpoint is released by Step 1, then F_i is in at most 3 components after that step (only that branchpoint and its adjacent i-paths are removed from F_i), and Step 2 is redundant. Otherwise, Step 1 only has an effect if the root has degree 2. In this case, Step 1 releases only the i-path through the root, since its endpoints are claimed, resulting in two components, and the subsequent application of Step 2 adds at most two more for a total of four.

Suppose Step 3 can be applied to F_i. If w_1 is not an endpoint of the i-path through the root of F_i (or the root itself), then, as in the previous case, Step 1 results in at most two components. Subsequently removing w_2 by Step 3 results in at most two more components for a total of four. If w_1 is an endpoint of the i-path containing the root of F_i (or the root itself), then both w_1 and w_2 are released (and nothing more). Since they share an i-path, this results in at most four components by Observation 3.1.

Finally, suppose Step 4 can be applied to F_i. If none of w_1, w_2 or w_3 is the root or is an endpoint of the i-path adjacent to the root, then Step 1 will result in additional components only if the root has degree two. Since both of the endpoints of this i-path are claimed in this case, removing this i-path and w_2 (by Step 4) results in at most four components. Otherwise, the combined application of Steps 1 and 4 requires the release of w_2, and possibly one of w_1 and w_3 as well (but not both, since if w_2 is the root, neither of the other branchpoints will be released). By Observation 3.1 this will result in at most four components. □

The following lemmas use this fact:

FACT 3.3. *([9]) The intersection of two subtrees of a tree is connected and contains the root of at least one of the subtrees, as long as the intersection is non-empty.*

LEMMA 3.4. *If F_i and F_j are each in two components after the forced phase, then after the approximation phase, there is no node that is in F_i and in F_j.*

Proof. This proof is similar to the correctness proof of the fixed-topology 2-phylogeny algorithm in [9]. Let C_i be a component of F_i after the forced phase, and let C_j be a component of F_j after the forced phase. Since F_i and F_j are both split in two components during the forced phase, all branch points in C_i and C_j are claimed during the forced phase (by Rule 2), and their intersection is a path in the fixed topology (i.e., all nodes are degree 2). Furthermore, the root of C_i or C_j is in the intersection. Therefore, the contention is cleared in Step 1 of the approximation phase of the algorithm. □

LEMMA 3.5. *If F_i is in one component after the forced phase, and F_j is in two components after the forced phase, then there is no node that is in F_i and in F_j after the approximation phase.*

Proof. First note that no branch point of T_j is part of T_i. (Since F_j is in only 2 pieces, it gave up only degree-2 nodes in the forced phase, and subsequently claimed all branch points. None of these are in F_i, since F_i was unbroken in the forced phase). Thus, the intersection of T_i and T_j is a path in T_j. We conclude that the intersection of F_i and F_j is a path in F_j and contains at most one branch point of F_i (otherwise, the path would be removed from F_j during the forced phase by Rule 1). If the intersection contains the root of F_j, then the contention will be removed during Step 1 of the approximation phase. Otherwise, the intersection contains the root of F_i. Thus, the single branch point of F_i that is contained in the intersection of F_i and F_j is either the root of F_i or it is an endpoint of the i-path containing the root of F_i. In either case, the contention will be removed in Step 1 of the approximation phase. □

LEMMA 3.6. *If F_i and F_j are each in one component after the forced phase, then there is no node that is in F_i and in F_j after the approximation phase.*

Proof. We will consider various cases. Case (α, β, γ) will represent the situation in which the intersection of F_i and F_j after the forced phase contains α branch points of F_i and β branch points of F_j, γ of which are shared. Recall that when a branchpoint v of F_i is released, v and all nodes on i-paths adjacent to v are removed from F_i.

Case $(0,0,0)$**:** As in the proof of Lemma 3.4, the intersection of F_i and F_j is a path containing the root of one of F_i and F_j, so the contention is cleared in Step 1 of the approximation phase.

Case $(1,0,0)$**:** As in the proof of Lemma 3.5, the intersection of F_i and F_j is a path of F_j containing one branch point of F_i so the contention is cleared in Step 1 of the approximation phase.

Case $(1,1,\gamma)$**:** Suppose without loss of generality that the root of F_i is in the intersection of F_i and F_j after the forced phase. Then the branch point of F_i is either the root of F_i or it is the endpoint of an i-path containing the root of F_i. In either case, it is released by F_i during Step 1 of the approximation phase.

Case $(2,0,0)$**:** This case cannot arise after the forced phase, because it requires two branch points of F_i on a single path of F_j, which is forbidden by Rule 1 of the forced phase.

Case $(2,1,0)$**:** By Rule 1, after the forced phase, the intersection of F_i and F_j has the branchpoint of F_j (w_2), between the two branchpoints for F_i (w_1 and w_3) as illustrated in Figure 2(a). Let w_4 be the other endpoint of the j-path adjacent to w_2 that contains w_1, and let w_5 be the other endpoint of the j-path adjacent to w_2 that contains w_3. By Rule 3 of the forced phase, all branch points of F_j except w_2, w_4 and w_5 are claimed. If node w_2 is released by F_j during the approximation phase, then the contention is cleared. Otherwise, by Rules 2 and 4 of the approximation phase, exactly one of $\{w_4, w_5\}$ is unclaimed after the forced phase. Suppose without loss of generality this is w_4. Because w_2 is not released by F_j in the approximation phase, the root of F_j is not w_2 or on any j-path adjacent to it. Therefore the root of F_i is in the intersection. If the root of F_i is on the i-path between w_1 and w_3 then by Step 1 of the approximation phase, F_i will release both w_1 and w_3, and the contention will be cleared. If the root of F_i was on the other i-path adjacent to w_1 in the intersection, then w_4 would be the closest to the root of F_j among the three unreleased F_j branchpoints, and F_j would have released w_2 by step 3 of the approximation phase. Finally, if the root is on an i-path adjacent to w_3 (but not w_1), F_i will release w_3 by Step 1 of the approximation phase, clearing the i-path from w_3 to w_1 (not including w_1). By Step 3 of the approximation phase F_j will release w_4, clearing the j-path from w_2 to w_4 (not including w_2). Therefore the contention is removed.

Case $(2,1,1)$**:** This case cannot arise after the forced phase, because it requires two branch points of F_i on a single path (including endpoints) of F_j, which is forbidden by Rule 1 of the forced phase.

Case $(2,2,0)$**:** By Rule 1 of the forced phase, the branch points of F_i and F_j are interleaved and lie on a path, as illustrated in Figure 2(b). Label the four relevant branch points w_1, w_2, w_3, w_4 such that for $k \in \{1,2,3\}$, there is a path between w_k and w_{k+1} which does not pass through any $w_\ell \notin \{w_k, w_{k+1}\}$. Without loss of generality,

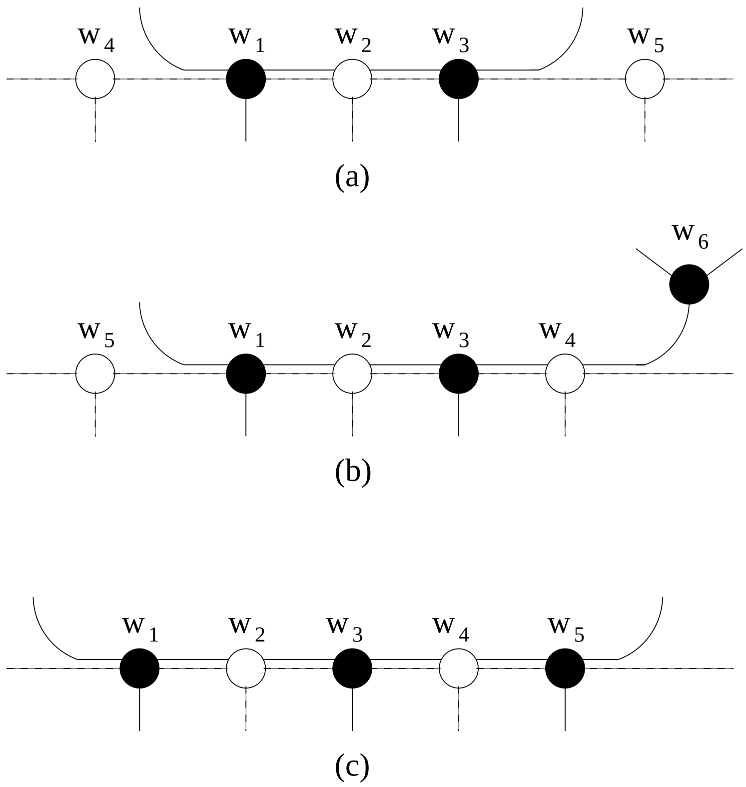

FIG. 2. *Cases from Lemma 10. Branchpoints of forest F_i are represented by solid circles, and i-paths are solid lines. Branchpoints of forest F_j are represented as empty circles with j-paths as dashed lines. Dashed and solid together represent shared paths. (a) case* $(2,1,0)$*, (b) case* $(2,2,0)$*, and (c) case* $(3,2,0)$.

assume that w_1 is a branch point of F_i. Let w_5 be the other endpoint of the j-path adjacent to w_2 that contains w_1, and let w_6 be the other end of the i-path adjacent to w_3 that contains w_4. By repeated applications of Rule 3 of the forced phase, all branch points of F_i and F_j other than w_1–w_6 are claimed.

If F_j does not claim w_5 in the forced phase and F_i does not claim w_6, then by Rule 4 of the approximation phase, w_3 is released by F_i and w_2 is released by F_j, so the contention is cleared.

So, suppose without loss of generality that w_6 is claimed by F_i. Consider the location of the root of F_i relative to w_1 and w_3. If the root of F_i is on the far side of w_6 (down one of the i-paths not adjacent to w_3), or on the (w_3, w_6) i-path between w_4 and w_6, then w_4 is on a j-path adjacent to the root of F_j. Therefore, by Step 1 of the approximation phase, F_j will release w_4, clearing contention up to, but not including w_2, and by Step 3 of the approximation phase, F_i will release w_1, clearing the remaining contention. A similar argument holds when the root is on the far side of w_1. If the root is on the (w_1, w_3) i-path, then by Step 1 of the approximation phase, F_i releases both w_1 and w_3, removing contention. Similarly, if the root is on the (w_2, w_4) j-path, (included in F_i), then F_j releases both w_2 and w_4.

Case $(3,2,0)$**:** By Rule 1 of the forced phase, the branch points of F_i and F_j are interleaved and lie on a path as illustrated in Figure 2(c). Label the five relevant branch points w_1, w_2, w_3, w_4, w_5 such that for $k \in \{1,2,3,4\}$, there is a path between w_k and w_{k+1} which does not pass through any $w_\ell \notin \{w_k, w_{k+1}\}$. By repeated application of

Rule 3 of the forced phase, all branch points of F_i and F_j other than w_1–w_5 are claimed. Node w_3 is released by F_i by Rule 4 of the approximation phase. Contention remains at w_1 and w_5. Consider where the global root is with respect to this intersection. If the global root is located down one of the paths adjacent to w_1 (but not adjacent to w_3), then the root of F_j is on an j-path adjacent to w_2. F_j releases w_2 by Step 1 of the approximation phase and w_4 by Step 3 of the approximation phase, removing the remaining contention. A similar argument holds if the root is down a i-path adjacent to w_5 (other than (w_3, w_5)). If the global root is down a j-path adjacent to w_2 (other than (w_2, w_4)), then the root of F_i is on an i-path adjacent to w_1 (or w_1 itself), and therefore F_i will release it by Step 1 of the approximation phase. As before, F_j will release w_4 by Step 3, removing the rest of the contention. A similar argument holds when the global root is down a j-path adjacent to w_4 (other than (w_2, w_4)). If the global root is on j-path (w_2, w_4), or down the i-path adjacent to w_3 which does not intersect this j-path, then by Step 1, F_j releases both w_2 and w_4, removing all contention.

Case $(3, 3, 0)$**:** This case cannot arise. By Rule 1 of the forced phase, if it did exist, the branch points of F_i and F_j would be interleaved and lie on a path. But then, by applying Rule 3 of the forced phase, we find that at least one of the relevant branch points of F_i and F_j would have been claimed during the forced phase.

Case $(\alpha > 1, \beta > 1, \gamma > 0)$**:** This case cannot arise after the forced phase, because it requires two branch points of F_i on a single path of F_j, which is forbidden by Rule 1 of the forced phase.

Case $(3, \beta \leq 1, \gamma)$**:** This case cannot arise after the forced phase, because it requires two branch points of F_i on a single path of F_j, which is forbidden by Rule 1 of the forced phase. ☐

LEMMA 3.7. *After the approximation phase, every node is in at most one forest F_i.*

Proof. The lemma follows from Lemmas 3.4, 3.5 and 3.6. ☐

4. Approximating Polymorphism. A **polymorphic character** (see [16]) allows more than one state per character per species. This type of character has strong application in linguistics [2, 22]. If there are r states, a polymorphic character is a function $c : S \rightarrow (2^{\{1\ ...,\ r\}} - \emptyset)$, where $2^{\{1,\ ...,\ r\}}$ denotes the power set (set of all subsets) of $\{1,\ ...,\ r\}$. For a given set of species, the *load* is the maximum number of states for any character for any species.

Often the evolution of biological polymorphic characters from parent to child is modelled by mutations, losses and duplications of states between species (see [16]). A **mutation** changes one state into another; a **loss** drops a state from a polymorphic character from parent to child; and a **duplication** replicates a state which subsequently mutates. We associate a cost with each mutation, duplication and loss between a pair of species. In the state-independent model, which we will consider, a loss costs c_l, a mutation costs c_m and a duplication costs c_d, regardless of which states are involved. Following the justification in [2], we insist $c_l \leq c_m \leq c_d$. Let $s_1, s_2 \in S$ and assume s_1 is the parent of s_2. In the state-independent model, we first look at the differences in cardinality of the parent and child sets. If the parent has fewer states, then we pay the appropriate duplication costs to account for the increased size of the child. If the parent is bigger, then we pay the loss cost. Then we match up as many elements as possible, and pay for the remaining changes as mutations. More specifically, as given in [2], we define $X = c(s_1) - c(s_2)$, and $Y = c(s_2) - c(s_1)$. Then the cost for the character c from s_1 to s_2 is $c_m * |X|$ if $|X| = |Y|$, and is $c_l * [|X| - |Y|] + c_m * |Y|$ if $|X| > |Y|$ and is $c_d * [|Y| - |X|] + c_m * |X|$ if $|Y| > |X|$.

As input we are given a fixed-topology T which has a unique species from S associated with each of its leaves, and label the leaf associated with $s \in S$ with the set of states $c(s)$. The **parsimony problem** is the problem of extending the function c to the internal nodes of T so that the sum of the costs over all edges of T is minimized. In the monomorphic case (one state per character per species), as discussed earlier, this problem can be solved in polynomial time [7], though the problem of finding a minimum cost labelling is NP-hard if the input does not include a topology [4, 5, 8, 21]. We will consider the **load problem**, introduced in [2]; calculate a labelling of the internal nodes of a fixed topology T with load at most ℓ and cost at most p. This problem was shown to be NP-hard in [2], even when $c_l = 0$ and the topology T is a binary tree. Note that the dynamic programming techniques presented in Jiang, Lawler and Wang's [13] and Gusfield and Wang's [19] papers do not appear to generalize for the polymorphic load problem. An (α, β)-approximation algorithm for the load problem computes a phylogeny with load at most $\alpha\ell$ and cost at most βp provided there is a load-ℓ cost-p phylogeny. Note that this is a pseudo-approximation algorithm, since the cost of the best $\alpha\ell$-load phylogeny may be significantly lower than the cost of the best ℓ-load phylogeny. In this section of the paper, we consider the load problem when $c_l = 0$ and the topology is arbitrary. We extend the results of section 2 to obtain, for any $\alpha > 1$, an $(\alpha, \frac{\alpha}{\alpha-1})$-approximation algorithm for the problem. (Note that taking $\alpha = 2$ gives a $(2, 2)$-approximation algorithm.)

We first quote the following observation, which was first noted in [2]:

OBSERVATION 4.1. *If $c_l = 0$, then if there is a labelling c for the topology T which has load ℓ and cost p, then there is also a labelling for T with load ℓ and cost p such that each internal node contains all the states in the subtree rooted at it or else has load ℓ.*

Therefore to approximate the load we only need to consider the labellings where each internal node contains all the states in the subtree rooted at it or else has load ℓ. For such a labelling c, any edge $e = (u \to v)$ such that $c(v) \not\subseteq c(u)$ satisfies $|c(u)| = \ell$. If $c(v) \subseteq c(u)$, then there is only loss from parent to child and the cost is 0. Otherwise, $|c(v) - c(u)| = |c(u) - c(v)|$ (since both parent and child have ℓ states), so the cost is $m * |c(v) - c(u)|$. We use this simpler cost function in an ILP which provides an exact solution for the load problem when $c_l = 0$. We then use the solution to the linear-programming relaxation of this ILP to compute an $(\alpha, \frac{\alpha}{\alpha-1})$-approximation for the problem. The integer program $\mathcal{P}$ uses the variables $X_{v,i}$, for each node v of the fixed-topology T and each state $i \in \{1, \ldots, r\}$, cost variables $cost_{e,i}$ for each edge $e \in E(T)$ and each state $i \in \{1, \ldots, r\}$ and the total cost variable $cost_e$ for each edge e. These variables have the following interpretation:

$$\begin{aligned}
X_{v,i} &= \begin{cases} 1 & \text{if state } i \text{ is in } c(v) \\ 0 & \text{otherwise} \end{cases} \\
cost_{e,i} &= \begin{cases} 1 & \text{if } i \in c(v) \text{ and } i \notin c(u), \text{ for } e = (u \to v) \\ 0 & \text{otherwise} \end{cases} \\
cost_e &= \textstyle\sum_{i=1}^{r} cost_{e,i}
\end{aligned}$$

The ILP $\mathcal{P}$ is then defined as:

$$\text{minimize } p$$

subject to

$$
\begin{array}{lrcll}
(9) & X_{v,i} & = & 1 & \text{for each leaf } v \in V(T),\ \forall i \in c(v) \\
(10) & \sum_{i=1}^{r} X_{v,i} & \leq & \ell & \forall v \in V(T) \\
(11) & cost_{e,i} & \geq & 0 & \forall e \in E(T),\ i = 1, \ldots, r \\
(12) & cost_{e,i} & \geq & X_{v,i} - X_{u,i} & \forall e = (u \to v) \in E(T),\ i = 1, \ldots, r \\
(13) & cost_e & = & \sum_{i=1}^{r} cost_{e,i} & \forall e = (u \to v) \in E(T) \\
(14) & \sum_{e \in E(T)} cost_e & \leq & p/c_m & \\
(15) & X_{v,i}, cost_{e,i} & \in & \{0,1\} &
\end{array}
$$

Constraints (11), (12) and (13) calculates $cost_e$ as $|c(v) - c(u)|$ for $e = (u \to v)$. Since the parsimony cost of a labelling is the sum of the costs on each edge, the cost of the labelling is $c_m * (\sum_{e \in E(T)} cost_e)$. Constraint (14) ensures that the cost of the labelling is no more than p. The integer program $\mathcal{P}$ solves the load problem exactly when $c_l = 0$. However, we require only that it provides a lower bound on the best cost. We now show that when we solve $\mathcal{P}$ with parameter ℓ, the optimal value of p is a lower bound on the cost of the best load-ℓ solution to the fixed topology problem.

LEMMA 4.2. *Let S be a species set, T be a fixed topology and $c : S \to (2^{\{1,\ldots,r\}} - \emptyset)$ be a polymorphic character on S. If the internal nodes v of T can be labelled with subsets of $\{1, \ldots, r\}$ to create a phylogeny for c with load ℓ and cost p, then there is a feasible solution for the linear program for this value of ℓ and p.*

Proof. Because of Observation 4.1, we can assume that in the load-ℓ, cost-p phylogeny, for each internal node v in $V(T)$, either $c(v') \subset c(v)$ for every child v' of v, or else $|c(v)| = \ell$. Therefore the cost of this phylogeny is $\sum_{(u \to v) \in E(T)} (c_m * |c(v) - c(u)|) = p$. Assign values to the $X_{v,i}$ variable for each internal node v and to the $cost_{e,i}$ variable for each edge $e = (u \to v)$ as follows:

$$
\begin{aligned}
X_{v,i} &= \begin{cases} 1 & \text{if } i \in c(v) \\ 0 & \text{otherwise} \end{cases} \\
cost_{e,i} &= \begin{cases} 1 & \text{if } i \in c(v) - c(u) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

This assignment satisfies constraints (15), (10), (11) and (12) of $\mathcal{P}$. Constraint (9) is automatically satisfied, and constraint (13) is definitional. Also, $c_m * cost_e = c_m * |c(v) - c(u)|$ for every $e = (u \to v)$ by definition of the $cost_{e,i}$, and therefore $\sum_{e \in E(T)} c_m * cost_e = p$, and constraint (14) is satisfied. ☐

Once again, since integer linear programming is $\mathcal{NP}$-hard, we solve the linear-programming relaxation $\mathcal{LP}$ of $\mathcal{P}$, which consists of all the constraints of $\mathcal{P}$ except that Constraint 15 is replaced with the constraint $0 \leq X_{v,i}, cost_{e,i} \leq 1$ (15′).

THEOREM 4.3. *Suppose there is a solution for the linear program $\mathcal{LP}$. Then we can assign states to the internal nodes of input tree T such that the resulting phylogeny for c has load $\alpha\ell$ and cost no more than $\left(\frac{\alpha}{\alpha-1}\right) p$.*

Proof. We assign states to the internal nodes of the fixed topology from the leaves upwards. For each internal node $v \in V(T) - L(T)$, consider the set $R(v) = \cup_{(v \to v') \in E(T)} c(v')$. If $|R(v)| \leq \alpha\ell$, then define $c(v) = R(v)$. If $|R(v)| > \alpha\ell$ then choose the $\alpha\ell$ states i of $R(v)$ which have the greatest $X_{v,i}$ values. By definition, this assignment of states to the internal nodes of T has load at most $\alpha\ell$.

To show that the cost of this assignment is no more than $\left(\frac{\alpha}{\alpha-1}\right) p$, note that the cost on an edge $e = (u \to v) \in E(T)$ is $c_m * |c(v) - c(u)|$, as $|c(v) - c(u)|$ is the number of mutations on e. Our assignment guarantees that if $|c(u)| < \alpha\ell$ then $c(u) \supset c(v)$, which implies $c_m * |c(v) - c(u)| = 0$, so we need only consider edges whose upper endpoint has full load. Suppose $|c(u)| = \alpha\ell$ and $i \in c(v) - c(u)$. Then, by construction of the phylogeny, there is a downwards path from v to some leaf w which has $i \in c(v')$ at every node along the path, including w. Suppose this path is $e_1 = (v \to v_1)$, $e_2 = (v_1 \to v_2)$, ..., $e_j = (v_{j-1} \to w)$. By the constraints of the linear program, $cost_{e,i} + cost_{e_1,i} + \ldots + cost_{e_j,i} \geq (X_{v,i} - X_{u,i}) + (X_{v_1,i} - X_{v,i}) + \ldots + (X_{w,i} - X_{v_{j-1},i}) = X_{w,i} - X_{u,i}$, and as w is a leaf and $i \in c(w)$, this is $1 - X_{u,i}$. Then, since $i \notin c(u)$, and the $\alpha\ell$ states in $c(u)$ were chosen to have the greatest $X_{u,j}$ values, we know $X_{u,i} \leq \ell/(\alpha\ell+1)$. The worst case is achieved when there are $\alpha\ell+1$ positive $X_{u,j}$ values all equal. They sum to at most ℓ from constraint 10, and therefore the smallest one, which cannot be included in the set, has value at most $\ell/(\alpha\ell+1)$. Therefore $cost_{e,i} + cost_{e_1,i} + \ldots + cost_{e_j,i} \geq ((\alpha-1)\ell+1)/(\alpha\ell+1)$. Furthermore, the costs $cost_{e,i}, cost_{e_1,i}, \ldots cost_{e_j,i}$ will not be allocated to any other mutation to i, because any mutation occurring above u will not have an unbroken path in i intersecting with any of the edges $e, e_1, \ldots, e_j$. So every mutation along an edge $e = (u \to v) \in T$ with $|c(u)| = \alpha\ell$ contributes at least $((\alpha-1)\ell+1)/(\alpha\ell+1)$ to the sum $\sum_{e \in E(T)} cost_e$ in our linear program. Hence $p/c_m \geq \sum_{e \in E(T)} cost_e \geq \sum_{e=(u \to v) \in E(T)} |c(v) - c(u)| \left(\frac{(\alpha-1)\ell+1}{\alpha\ell+1}\right)$, so the cost $\sum_{e=(u \to v) \in E(T)} c_m * |c(v) - c(u)| \leq (\alpha/(\alpha-1)) * p$. ☐

REFERENCES

[1] H. Bodlaender, M. Fellows, T. Warnow, "Two Strikes Against Perfect Phylogeny", *Proceedings of the 19th International Congress on Automata, Languages and Programming (ICALP)*, Springer-Verlag Lecture Notes in Computer Science, pp. 273-287 (1992).

[2] M. Bonet, C. Phillips, T.J. Warnow and S. Yooseph, Constructing Evolutionary Trees in the Presence of Polymorphic Characters, *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (1996).

[3] I. Borosh and L.B. Treybig, Bounds on positive integral solutions of linear Diophantine equations, *Proceedings of the American Mathematical Society*, Vol 55 (1976).

[4] W.H.E. Day, Computationally difficult parsimony problems in phylogenetic systematics, *Journal of Theoretical Biology*, Vol 103 (1983).

[5] W.H.E. Day, D.S. Johnson and D. Sankoff, The computational complexity of inferring phylogenies by parsimony, *Mathematical biosciences*, Vol 81 (1986).

[6] W.H.E. Day and D. Sankoff, "Computational complexity of inferring phylogenies by compatibility", *Systematic Zoology*, Vol 35(2):224-229 (1986).

[7] W. Fitch, Towards defining the course of evolution: minimum change for a specified tree topology, *Systematic Zoology*, Vol 20 (1971).

[8] L.R. Foulds and R.L. Graham, "The Steiner Problem in Phylogeny is NP-complete", *Advances in Applied Mathematics*, Vol 3:43-49 (1982).

[9] L.A. Goldberg, P.W. Goldberg, C.A. Phillips, E. Sweedyk and T. Warnow, "Minimizing phylogenetic number to find good evolutionary trees", *Discrete Applied Mathematics*, to appear.

[10] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company (1979).

[11] D. Hochbaum, "Approximation algorithms for set covering and vertex cover problems", *SIAM Journal on Computing*, Vol 11:555-556 (1982).

[12] D. Hochbaum (Ed.), *Approximation algorithms for NP-hard problems*, PWS Pub. Co. (1997).

[13] T. Jiang, E.L. Lawler and L. Wang, "Aligning Sequences via an Evolutionary Tree: Complexity and Approximation", *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing* (1994).

[14] D. Karger, C. Stein, and J. Wein, "Scheduling algorithms". To appear in *Handbook of Algorithms and Theory of Computation*, Mikhail J. Atallah, Editor., CRC Press.

[15] R.M. Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations*, eds. R.E. Miller and J.W. Thatcher, Plenum Press (1972).

[16] M. Nei, *Molecular Evolutionary genetics*, Columbia University Press, New York (1987).

[17] P. Raghavan and C.D. Thompson, "Randomized Rounding", *Combinatorica*, Vol 7:265-274 (1987).

[18] M.A. Steel, "The complexity of reconstructing trees from qualitative characters and subtrees", *Journal of Classification*, Vol 9:91-116 (1992).

[19] L. Wang and D. Gusfield, "Improved Approximation Algorithms for Tree Alignment", *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, 220-233 (1996).

[20] L. Wang, T. Jiang, and D. Gusfield, "A more efficient approximation scheme for tree alignment", *Proceedings of the First Annual International Conference on Computational Molecular Biology* (1997).

[21] T.H. Wareham, "On the Computational Complexity of Inferring Evolutionary Trees", M.Sc. thesis, Technical Report No. 9301, Department of Computer Science, Memorial University of Newfoundland, Canada, (1993).

[22] T. Warnow, D. Ringe and A. Taylor, "A character based method for reconstructing evolutionary history for natural languages", Tech Report, Institute for Research in Cognitive Science, University of Pennsylvania, (1995), and *Proceedings of the 7th Annual ACM/SIAM Symposium on Discrete Algorithms* (1996).