SIAM J. COMPUT.
Vol. 29, No. 1, pp. 103–131
© 1999 Society for Industrial and Applied Mathematics

CONSTRUCTING EVOLUTIONARY TREES IN THE PRESENCE OF POLYMORPHIC CHARACTERS*

MARIA BONET†, CYNTHIA PHILLIPS‡, TANDY WARNOW§, AND SHIBU YOOSEPH¶

Abstract. Most phylogenetics literature and construction methods based upon characters presume monomorphism (one state per character per species), yet polymorphism (multiple states per character per species) is well documented in both biology and historical linguistics. In this paper we consider the problem of inferring evolutionary trees for polymorphic characters. We show efficient algorithms for the construction of perfect phylogenies from polymorphic data. These methods have been used to help construct the evolutionary tree proposed by Warnow, Ringe, and Taylor for the Indo-European family of languages and presented by invitation at the National Academy of Sciences in November 1995.

Key words. algorithms, graphs, evolutionary trees

AMS subject classifications. 05C05, 68Q25, 92-08, 92B05

PII. S0097539796324636

1. Introduction.

Determining the evolutionary history of a set S of objects (taxa or species) is a problem with applications in a number of domains such as biology, comparative linguistics, and literature. Primary data used to compare different taxa (whether biological species, populations, or languages) can be described using *characters*, where a character is a function $\alpha : S \to Z$, where Z denotes the integers and thus represents the set of possible *states* of α. In this paper we consider tree construction when characters are permitted to have more than one state on a given object. We call this the *polymorphism problem*. A character that is permitted to have more than one state on a given object will be called a *polymorphic character*, and one that can have only one state for every object is referred to as a *monomorphic* character.

Polymorphism is well documented in both the molecular genetics and comparative linguistics domains [8, 33]. For example, the population geneticist Masatoshi Nei writes: "The study of protein polymorphism has indicated that the extent of genetic variation in natural populations is enormous. However, the total amount of genetic variation cannot be known unless it is studied at the DNA level. The study of DNA polymorphism is still in its infancy, but the results so far obtained indicate that the extent of DNA polymorphism is far greater than that of protein polymorphism" [28, p. 254]. Polymorphism also arises in the comparison of languages. The Indo-Europeanist Donald Ringe writes: "In choosing lexical characters we try to work with

*Received by the editors August 21, 1996; accepted for publication (in revised form) February 20, 1998; published electronically September 14, 1999.

http://www.siam.org/journals/sicomp/29-1/32463.html

†Department of Computer Science, Universidad Politecnica de Cataluña, Barcelona, Spain (bonet@goliat.upc.es). The research of this author was partly supported by NSF grant CCR-9403447.

‡Sandia National Laboratories, Albuquerque, NM (caphill@cs.sandia.gov). The work of this author was performed under U.S. Department of Energy contract DE-AC04-76AL85000.

§Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA (tandy@central.cis.upenn.edu). The research of this author was partly supported by an NSF National Young Investigator Award under contract ccr-9457800 and an NSF grant in linguistics.

¶Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA (yooseph@gradient.cis.upenn.edu). The research of this author was partly supported by a fellowship from the University of Pennsylvania and by a fellowship from the University of California at Berkeley supported by NSF grant DMS-9406348.

basic meanings (semantic slots), choosing from each language the word that most usually expresses each basic meaning. Languages typically have one word for each basic semantic slot, but instances of two (or even more) words apparently filling the same basic slot are not rare" [30].

Thus polymorphic data is a reality when working with evolutionary tree construction for both linguistic analysis and biological taxa, and methods appropriate for such construction must be devised. In the phylogenetics literature and programs (such as Phylip, PAUP, and MacClade), algorithms and software to *evaluate* fixed leaf-labeled tree topologies for polymorphic data have explicitly required that the number of states be kept quite small because the evaluation requires time exponential in the number of states. This is the first algorithmic study of this problem to go beyond fixed topology problems for bounded number of states.

The concept of an idealized evolutionary tree was introduced by LeQuesne in a series of articles (see, for example, [24, 25]), and later termed "Perfect Phylogenies" by Gusfield in [18], and studied in several papers (see, for example, [13, 14]). The major contribution of this paper is a methodology for inferring perfect phylogenies from monomorphic and polymorphic characters. Recent work in historical linguistics [39] has shown that perfect phylogenies should be obtainable from properly selected and encoded linguistic characters. Algorithms for constructing perfect phylogenies from monomorphic characters were used in [39] to analyze the Indo-European family of languages, whose first-order subgrouping had been argued for decades without resolution. The methodology we propose here significantly extends the range of the data that can be analyzed in historical linguistics. We have applied this methodology to the data set studied by Warnow, Ringe, and Taylor. Detection and resolution of polymorphism led to a modification of their initially proposed phylogeny, which was based only on monomorphic characters. Our methodology and its results were presented at the Symposium on the Frontiers of Science at the National Academy of Sciences in November 1995 [38].

The structure of the rest of the paper is as follows. In section 2 we discuss the causes of polymorphism in linguistics and biology and define the problem of inferring trees from polymorphic characters in these two domains. We show that a perfect phylogeny is an appropriate objective when working with linguistic data as well as some biological data. In section 3 we present two algorithms, one graph theoretic and one combinatorial, for the problem of inferring perfect phylogenies from polymorphic data. In section 4.3 we present a methodology for inferring perfect phylogenies from data which combine monomorphic and polymorphic data. In section 5 we present our analysis of the Indo-European data studied by Warnow, Ringe, and Taylor [39]. In section 6 we consider the problem of inferring evolutionary trees from polymorphic data when a perfect phylogeny is an unlikely outcome. We conclude in section 7.

2. Foundations. The causes of polymorphism in biology and linguistics differ, and within biology, polymorphism has more than one cause as well. In linguistics, convergence of meanings over time, borrowing of synonyms from other languages, and the inability of modern-day linguists to detect subtle differences of meaning in words from ancient languages can all produce polymorphic characters. Some such cases, like English *little* and *small,* arise by the convergence of meanings over time; others, like American English *stone* and *rock* (to describe a small chunk of the substance that can be thrown), are instances of replacement in progress (*rock* is replacing *stone* in that basic meaning in America). It can be shown that the different manifestations of polymorphism in linguistics each can be described by the conflation of two or more

distinct linguistic characters. Often we are able to determine the precise number of monomorphic characters that have merged into the polymorphic character. In linguistics it has been observed that monomorphic characters are *convex*, by which we mean that the nodes sharing any state of any character form a connected set in the tree.

DEFINITION 2.1. *Given a set S of taxa defined by a set C of characters ($|C| = k$), where each $\alpha_j \in C$ is a function $\alpha_j : S \to (2^Z - \{\emptyset\})$, let T be a tree that is leaf-labeled by the taxa in S and with each internal node v labeled with a vector from $(2^Z - \{\emptyset\})^k$ such that the value of $\alpha_j(v)$ is given by the jth component of this vector. A character (polymorphic or monomorphic) $\alpha_j \in C$ is* convex *on T if for all $i \in Z$ the set $X_{i,\alpha_j} = \{v \in V(T) : i \in \alpha_j(v)\}$ is connected. T is a* perfect phylogeny *if every character is convex.*

For polymorphism caused by convergence of convex monomorphic characters, polymorphism can be considered a *separation* problem.

DEFINITION 2.2. *Let α be a character defined on a species set S and let i be a character state of α. Then we define $\alpha^{-1}(i) = \{s \in S : i \in \alpha(s)\}$.*

DEFINITION 2.3. *Let β be a polymorphic character with character states $0, \ldots, r-1$, and for each $s \in S$ suppose $|\beta(s)| \leq l'$. Then β is* separated *into l monomorphic characters $\alpha_1, \ldots, \alpha_l$, where $l \geq l'$, if there is a function $f : \{0, \ldots, r-1\} \to \{1, \ldots, l\}$, such that if $f(i) = j$ then i is a character state of α_j and in addition, for every $s \in \alpha_j^{-1}(i)$, we have $i \in \beta(s)$.*

Example. Let $S = \{A, B, C, D\}$ and let $\beta(A) = \{0, 1\}, \beta(B) = \{2, 1\}, \beta(C) = \{0\}$, and $\beta(D) = \{2\}$. Then β is separated into two monomorphic characters α_1 and α_2 by setting $f(0) = 1, f(1) = 2, f(2) = 1$. Thus $\alpha_1(A) = \{0\}, \alpha_1(B) = \{2\}, \alpha_1(C) = \{0\}, \alpha_1(D) = \{2\}, \alpha_2(A) = \{1\}$, and $\alpha_2(B) = \{1\}$. In addition, we set $\alpha_2(C) = \{4\}$ and $\alpha_2(D) = \{5\}$.

In the example note that $\alpha_2(C)$ and $\alpha_2(D)$ are set to previously unused values. Thus *if for some $j' \in \{1, 2, \ldots, l\}$ $\alpha_{j'}(s)$ is undetermined, then $\alpha_{j'}(s)$ can be set to a previously unused state.*

Problem 1 (separation into l convex characters).

Input. Set S of taxa (defined by set C of characters) and an integer l.

Question. Can we separate each character into at most l monomorphic characters, so that a perfect phylogeny exists for the derived set of monomorphic characters?

Due to inadequate historical evidence, input data may not reflect the actual degree of polymorphism. Separation may be necessary to obtain convexity even if all input characters appear monomorphic. For example, consider four languages with three characters: $A = (1, 2, 1), B = (1, 2, 2), C = (1, 2, 1), D = (1, 2, 2)$. Suppose the first two characters convolve (meanings merge) and linguists detect only one of these characters for each language. This polymorphic character appears monomorphic: $A = (1, 1), B = (1, 2), C = (2, 1), D = (2, 2)$. There is no perfect phylogeny for this set, but we can separate the first character into two such that there is a perfect phylogeny: $A = (1, a, 1), B = (1, b, 2), C = (c, 2, 1)$, and $D = (d, 2, 2)$. Because of lost information, we cannot completely determine the *inferred* characters α_i (hence the use of singletons or previously unused states).

We note that an r-state character can always be separated into r monomorphic *single state* characters which are each convex on all phylogenies. Thus $l = r$ is an easy instance of Problem 1.

In biology, polymorphic characters can arise when dealing with allozyme data [26] and morphological data [40]. In coding allozyme data, each locus is assumed to be a

character (as opposed to a character being defined as the presence or absence of individual alleles) and the set of character states can then be defined by the combination of alleles present at the locus. When dealing with sequence data, alternative encodings of the same amino acid sequence can also lead to the presence of polymorphic characters. In each of these cases, the number of different forms that the character can take on a given taxon may be bounded, in which case we may reasonably seek a tree in which every node has no more than some prespecified bound of states for each character. This bound may be character dependent.

DEFINITION 2.4. *Let T be a phylogeny. A character α is said to have* load *l if, for every $v \in V(T)$, $|\alpha(v)| \leq l$. The load of T is defined to be the maximum load on any character.*

Problem 2 (l-load perfect phylogeny).

Input. Set S of taxa (defined by set C of polymorphic characters) and an integer l.

Question. Does an l-load perfect phylogeny exist?

For many morphological characters in biology, convexity is a reasonable assumption (e.g., consider *vertebrate-invertebrate*). Although the causes of polymorphism in biology and linguistics differ, when convexity can be assumed the different problem formulations are equivalent.

THEOREM 2.5. *Given a set of taxa defined by a set C of polymorphic characters, T is an l-load perfect phylogeny for C iff we can separate each polymorphic character into at most l monomorphic characters such that T is also a perfect phylogeny for the derived set C'.*

Proof. One direction is easy. For the converse, let T be a perfect phylogeny with load l, let $\alpha \in C$ be given, and assume α has r states present on S. Let T_i be the subgraph of T induced by the vertices labeled i by α. Since T is a perfect phylogeny, each T_i is a subtree. Define G_α to be the graph whose vertices are in one-to-one correspondence with the subtrees $T_i, i = 1, 2, \ldots, r$, and where $(T_i, T_j) \in E$ iff $T_i \cap T_j \neq \emptyset$. Note that since T has load l, G_α has max clique size at most l. G_α is triangulated since it is the intersection graph of subtrees of a tree [7], and hence G_α is perfect [17]. Since G_α is perfect, the chromatic number of G_α equals the max clique size and hence is bounded by l. Hence we can partition the nodes of G_α into at most l independent sets, $V_1, V_2, \ldots, V_l$. Each V_i thus defines a monomorphic character (filled in with singletons), and hence T is a perfect phylogeny for each of these monomorphic characters. □

Polymorphism in characters that are based upon columns of molecular sequences behaves differently than polymorphism in morphological characters; for these characters, variations on the parsimony criterion are more appropriate optimization criteria. We discuss the computational complexity of these problems in section 6.

3. Inferring perfect phylogenies from polymorphic characters. When the maximum permissible load for each character is not given, the problem of inferring perfect phylogenies is best stated as a *minimum load* problem. This is addressed in section 3.1. When the maximum permissible load for each character is given, we have two algorithms which can construct perfect phylogenies; both are efficient when the number of characters is small. These algorithms are presented in section 4. When the character set includes a sufficient number of monomorphic characters, we have a third algorithm which combines techniques for monomorphic and polymorphic characters. This algorithm is presented in section 4.3.

The various parameters to the problem are n (the number of species), k (the number of characters), r (the maximum number of states per character), and l (maximum

load for each character).

3.1. Minimum load problems. When convexity of the monomorphic constituents of the polymorphic characters is a reasonable request, we may seek a tree with a prespecified load bound, or else we may seek a tree with a minimum possible load bound. We call the latter problem the *minimum* (or *min*) *load problem.*

We note that the min load problem is NP-hard since the question of whether a 1-load perfect phylogeny exists is NP-complete [4, 36]. However, although the 1-load perfect phylogony problem is NP-complete, the natural fixed parameter versions of the problem are solvable in polynomial time; see [1, 2, 5, 12, 19, 21, 22, 23, 37]. The 2-load perfect phylogeny problem is the next question to consider.

THEOREM 3.1.

(i) *The min load problem can be solved in polynomial time for all fixed n.*
(ii) *The min load problem can be solved in polynomial time when $r = 2$.*
(iii) *The min load problem is NP-hard for all fixed k.*
(iv) *The min load problem is NP-hard for all fixed $r \geq 3$.*
(v) *Determining whether a 2-load perfect phylogeny exists is solvable in polynomial time for all fixed n.*
(vi) *Determining whether a 2-load perfect phylogeny exists is solvable in polynomial time for $r = 2$.*
(vii) *Determining whether a 2-load perfect phylogeny exists is solvable in polynomial time for all fixed k.*
(viii) *Determining whether a 2-load perfect phylogeny exists is NP-complete for all fixed $r \geq 3$.*

Proof. Parts (i) and (v): When n is fixed, the number of possible leaf-labeled topologies is bounded, so we need only consider the min load problem on a fixed topology. Determining the minimum load on a fixed leaf-labeled topology is trivial, since for each internal node $v \in V(T)$ and each character $\alpha \in C$ we simply set $\alpha(v) = \{i : \exists x, y$ leaves of T with v on the path from x to y, and $i \in \alpha(x) \cap \alpha(y)\}$. This determines the minimum load for the topology. The same argument can be used to show that 2-load perfect phylogeny is solvable in polynomial time when n is fixed.

Parts (ii) and (vi): If $r = 2$, then clearly the min load problem and thus the 2-load perfect phylogeny problem can be solved in polynomial time by observing that 1-load perfect phylogeny on binary characters is solvable in polynomial time [18] and that there is always an r-load perfect phylogeny on any input set containing characters with at most r states.

Part (iii): We now show that the min load problem is NP-hard for all fixed k by showing that the l-load perfect phylogeny problem with fixed number of characters $k \geq 1$, where each character has input load 2 (i.e., two states for every species), is NP-complete. The reduction is from the following problem involving partial t-tree recognition. See section 4.2.3 for definitions of t-trees and partition intersection graphs.

Problem 3 (partial t-tree recognition).

Input. A graph $G = (V, E)$ and an integer $t \leq (n - 1)$, where $|V| = n$.

Question. Is G a partial t-tree, i.e., does there exist $G' = (V, E')$ such that $E \subseteq E'$ and G' is a t-tree?

The above problem was shown to be NP-complete by Arnborg, Corneil, and Proskurowski [3].

The reduction is as follows. Let $(G = (V, E), t)$ be an instance of the partial t-tree problem. The corresponding instance of the load problem consists of the species set

$S = \{s_e | e \in E\}$ and one character α, with $\alpha(s_e) = \{i, j\}$, where $e = (i, j)$. Also, set $l = t + 1$. We claim that the instance to the partial t-tree problem has a solution iff the corresponding instance to the load problem has a solution. This can be seen by observing that G is the partition intersection graph of the instance of the load problem and thus we can use Theorems 4.8 and 4.9.

Parts (iv) and (viii): Next we show that the 2-load perfect phylogeny problem, where each of the input characters is monomorphic, is NP-complete for fixed $r \geq 3$. This will also imply that the min load problem is NP-hard for fixed $r \geq 3$. The reduction is from the partial binary characters problem (PBCP), which is defined as follows.

Problem 4 (partial binary character perfect phylogeny).

Input. An $n \times k$ matrix M, of n species and k characters, in which each entry of M is an element of the set $\{0, 1, *\}$.

Question. Can each $*$ entry be set to 0 or 1 so that there exists a 1-load perfect phylogeny with the new matrix ?

The above problem is just a reformulation of the quartet consistency problem, which was shown to be NP-complete by [36].

Given an instance I of PBCP, the instance of the 2-load problem is constructed as follows. Replace each $*$ entry in the matrix defined by I with a 2. Let C be the set of k characters and let S be the set of n species defined by this new matrix. We will add $2k$ new characters and $9k$ new species as follows ($s^i_\alpha = (x, y, z)$ indicates that $\alpha(s^i_\alpha) = x, \alpha^1(s^i_\alpha) = y, \alpha^2(s^i_\alpha) = z$):

1. Initialize $S' = S$ and $C' = C$.
2. For each $\alpha \in C$, do the following:
   a. Define two new characters α^1 and α^2 and nine new species $s^1_\alpha, \ldots, s^9_\alpha$ as follows:
      i. For each $\beta \in C'$ (where $\beta \neq \alpha$) set $\beta(s^i_\alpha) = 2$, where $1 \leq i \leq 9$.
      ii. For each $s \in S'$ set $\alpha^1(s) = 2$ and $\alpha^2(s) = 2$.
      iii. Set $s^1_\alpha = (0, 0, 2)$, $s^2_\alpha = (0, 1, 2)$, $s^3_\alpha = (0, 2, 2)$, $s^4_\alpha = (2, 0, 0)$, $s^5_\alpha = (2, 1, 1)$, $s^6_\alpha = (2, 2, 2)$, $s^7_\alpha = (1, 2, 0)$, $s^8_\alpha = (1, 2, 1)$, $s^9_\alpha = (1, 2, 2)$.
   b. Update $S' = S' \cup \{s^1_\alpha, \ldots, s^9_\alpha\}$ and $C' = C' \cup \{\alpha^1, \alpha^2\}$.

$I' = (S', C')$ is the instance of the load problem. We claim that I has a solution iff I' has a perfect phylogeny with load 2. The proof follows. Let T be a perfect phylogeny which is a solution to instance I of PBCP. Each vertex in T is a k-tuple binary vector. We will first construct the solution for the load problem when restricted to the initial species set S. Identify the species which initially had a $*$ entry for that character and replace the state for that character by a 2. It can be verified that by doing this for every character in C and then relabeling the internal vertices so that the convexity property still holds, we get a solution to the load problem for the initial species S and character set C. Extend the character set C to C' by adding the new characters, which consist entirely of character state 2. This is still a solution to the load problem for S with C'. Let T' be the tree obtained as a result of the above modifications. We will now show how to add the additional $9k$ species. We define $\alpha^{-1}(i) = \{s : \alpha(s) = i\}$. For each $\alpha \in C$ note that there is some edge in T' which separates $\alpha^{-1}(0)$ from $\alpha^{-1}(1)$. For each character $\alpha \in C$, identify an edge e which separates $\alpha^{-1}(0)$ from $\alpha^{-1}(1)$. Attach the 9 new species, associated with α, as shown in Figure 3.1. We note that $\alpha^{-1}(0) \subseteq A$ and $\alpha^{-1}(1) \subseteq B$.

Let T'' be the tree finally obtained after the addition of the $9k$ new species as described above. It can be easily verified that T'' is a solution to instance I' of the

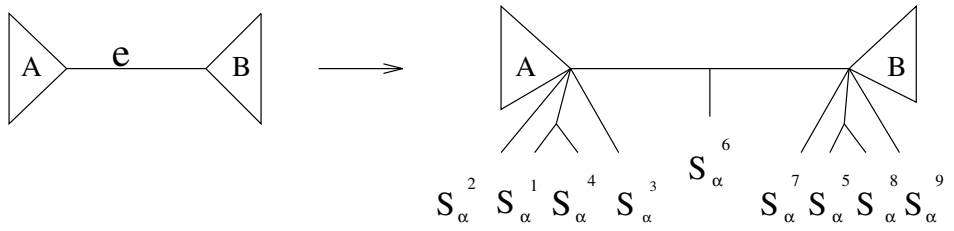

FIG. 3.1. *Adding the* 9 *new species associated with* α.

2-load problem.

For the other direction of the proof, let T be a solution to instance I' of the load problem. We first observe that in any solution to an instance of the 2-load problem involving 3-state monomorphic characters in the input, every character α has associated with it an edge, which splits $\alpha^{-1}(0)$ from $\alpha^{-1}(1)$, or $\alpha^{-1}(0)$ from $\alpha^{-1}(2)$, or $\alpha^{-1}(1)$ from $\alpha^{-1}(2)$. Observe that in I', for each $\alpha \in C$, the only partition possible is $\alpha^{-1}(0)$ from $\alpha^{-1}(1)$. This follows as a result of the constraints imposed by α^1 and α^2. Thus, to get a solution to the PBCP for instance I, we restrict T to the original set of species S and characters C, and then for each character in C we replace the 2's that appear on the 0's side of the partition by 0's and the 2's that appear on the 1's side of the partition by 1's.

This completes the proof.

Part (vii): If k is fixed, then the 2-load perfect phylogeny problem can be solved in polynomial time using the algorithms from section 4. □

This theorem shows that *any* polynomial time algorithm *requires* both k and l bounded (under $P \neq NP$ assumption).

4. Algorithms for perfect phylogenies from polymorphic characters. In this section we present the two algorithms for inferring perfect phylogenies from polymorphic data when we know the load bound. Although the algorithms we will present assume a universal load bound, these algorithms can be easily modified to allow individual load bounds for each character and will achieve comparable running times. For the sake of clarity, we will present these algorithms as though the load bound is the same for each character; the run times of these algorithms when implemented to handle variable constraints are given within their respective sections.

4.1. A combinatorial algorithm for fixed k and l. The algorithm we present is an extension and simplification of the algorithm of Agarwala and Fernández-Baca [2]. For the remainder of this section the term *perfect phylogeny* refers to an l-load perfect phylogeny.

Because each character has only r states and each node can choose at most l of these in an l-load perfect phylogeny, the number of possible labels for nodes in the tree is $O(r^{lk})$. Let us call this set S^* and note that $S \subseteq S^*$ (since otherwise some node in S has load greater than l). In contrast to the algorithm in [2], we do not require that the internal nodes be labeled distinctly from the species in S, and instead we will permit species in S to be internal nodes because we can transform any perfect phylogeny in which some species in S label internal nodes into a perfect phylogeny in which all species label leaves by attaching a leaf for s to the internal node labeled by s.

We need some preliminary definitions and facts.

DEFINITION 4.1. $\sum_{\alpha \in C} |\alpha(x) \triangle \alpha(y)|$ *is the* extended Hamming distance *of* $e = (x, y)$*, where* $\triangle$ *denotes the symmetric difference. However, we will call this the* Hamming distance*, understanding this to refer to the extended Hamming distance.*

We note that if a perfect phylogeny exists for S, then one exists where the Hamming distance on any edge is exactly one. We will seek a perfect phylogeny with this property. Working with such perfect phylogenies allows us to quickly solve subproblems because it limits the number of ways a (maximally refined) perfect phylogeny can be constructed.

DEFINITION 4.2 (see [23]). *Given* $x \in S^*$*, the equivalence relation* E_x *is the transitive closure of the following relation* E'_x *on* $S - \{x\}$*:* $aE'_x b$ *if there exists character* α *such that* $(\alpha(a) \cap \alpha(b)) - \alpha(x) \neq \emptyset$*. We denote this set of equivalence classes by* $(S - \{x\})/x$*.*

An observation that follows immediately from this definition is that if T is a perfect phylogeny on S and x an internal node in T, then two species in S which are in the same equivalence class of $(S - \{x\})/x$ must be in the same component of $T - \{x\}$. We also make the following observation.

LEMMA 4.3. *Let* T *be a perfect phylogeny on* S *and* x *be an internal node in* T*. Consider* T *as rooted at* x*. Let* G *be an equivalence class of* $(S - \{x\})/x$ *and let* $y = lca_T(G)$*. Let* v *be a node of* T *on the path from* x *and* y *(thus* $v = x$ *or* $v = y$ *is also possible). Then there exist* $H_1, \ldots, H_t$ *in* $(S - \{v\})/v$ *such that* $H_1 \cup \cdots \cup H_t = G$*.*

Proof. Let T be a perfect phylogeny for S, and x, G, y, and v are as stated. Let $H_1, \ldots, H_t$ be equivalence classes of $(S - \{v\})/v$ containing species from G. Clearly, to prove the lemma it will suffice to prove that all H_i are either disjoint from G or contained in G.

Suppose, by way of contradiction, that for some $i, 1 \leq i \leq t$, H_i contains species from G and from $S - G$. We will show that this implies the existence of a character $\alpha \in C$ and a state a of α such that $a \notin \alpha(v)$, yet $a \in \alpha(x) \cap \alpha(z)$ for some leaf z below v; such a character is not convex on T, contradicting our assumption that T is a perfect phylogeny. This will show that all equivalence classes H_i are either disjoint from or contained in G and will establish our claim.

Since H_i contains species in G and in $S - G$, and is an equivalence class of $(S - \{v\})/v$, there are species $z_1 \in H_i \cap G$ and $z_2 \in H_i - G$ and character α such that $(\alpha(z_1) \cap \alpha(z_2)) - \alpha(v) \neq \emptyset$ (this follows from $(S - \{v\})/v$ being the transitive closure of E_v). Let $a \in \alpha(z_1) \cap \alpha(z_2) - \alpha(v)$. Since z_1 and z_2 are in different equivalence classes of $(S - \{x\})/x$, $a \in \alpha(x)$. Now let $z = lca_T(z_1, z_2)$. This node z is in the subtree rooted at v and satisfies $a \in \alpha(z)$ because T is a perfect phylogeny and z is on the path between z_1 and z_2. This is the character α and state a we stated we would demonstrate, proving our claim. □

We now present a dynamic programming algorithm for constructing perfect phylogenies from polymorphic data. We define the *search graph* $SG = (V, E)$ as follows. Each vertex in V is associated with a pair $[G, x]$, where $G = S$ or $G \in (S - \{x\})/x$, and represents the question, Does $G \cup \{x\}$ have a perfect phylogeny? The edges of the search graph are of the form $([G, x][S, x])$, and all pairs of the form $([G_1, x_1], [G_2, x_2])$ where $G_1 \subseteq G_2$ and x_1 and x_2 satisfy $\sum_{\alpha \in C} |\alpha(x_1) \triangle \alpha(x_2)| = 1$. There are $O(r^{lk})$ nodes of type $[S, x]$, and $O(nr^{lk})$ of type $[G, x]$ (because there are at most n equivalence classes in $(S - \{x\})/x$). Also, there are $O(nr^{lk})$ edges of type $([G, x][S, x])$ and $O(nlkr^{lk+1})$ of type $([G_1, x_1], [G_2, x_2])$, since the outdegree of every node is at most lkr.

DEFINITION 4.4. *Given a node* $[G, x]$*, a set of nodes* $[H_1, y], [H_2, y], \ldots, [H_p, y]$

such that (a) Hamming(x,y)= 1 and (b) $\cup_i H_i = G$ is called a bundle.

There can be multiple bundles going into $[G, x]$, corresponding to the maximally refined perfect phylogenies of $G \cup \{x\}$. If $[H_1, y], [H_2, y], \ldots, [H_p, y]$ is a bundle for $[G, x]$, and all the subproblems have perfect phylogenies, then there is a perfect phylogeny for $G \cup \{x\}$ with subtrees T_i labeled by H_i. We can also have a bundle of just one edge (i.e., ([G,y],[G,x])); such a bundle indicates the existence of a perfect phylogeny T for $G \cup \{y\}$ in which the node corresponding to y has only one child. This is necessary if we require all edges to have Hamming distance 1.

ALGORITHM PHYLOGENY(S). *First create the search graph G_S. For each node $[G, x]$, determine its bundles. Note that some incoming edges $([G_1, x_1], [G, x])$ may not correspond to any bundle because $(S - \{x_1\})/x_1$ does not have the proper form (i.e., G may not be the union of a subset of the components of $(S - \{x_1\})/x_1$). Remove such edges. Now for each bundle compute the size of the bundle (number of edges) b_i and set a counter* **count**$_i$ *equal to b_i. Each node $[G_1, x_1]$ that is a predecessor of node $[G_2, x_2]$ is given a pointer to the counter for its bundle. We initialize a queue of "true" nodes as empty.*

We locate each node $[G, x]$ with $|G| = 1$, mark it as true, and place it in the queue. We then pull a node $[G_1, x_1]$ out of the queue and process it as follows. For each edge in the search graph $([G_1, x_1], [G_2, x_2])$ we decrement the counter for the appropriate bundle into $[G_2, x_2]$. If the counter is decremented to 0, then all edges of the bundle have been set to true and node $[G_2, x_2]$ is added to the queue. When we have processed all edges out of node $[G_1, x_1]$ we choose another node from the queue and continue. If we ever try to enqueue a node of the form $[S, x]$, then the instance has a perfect phylogeny. If the queue is emptied without ever labeling a node of this form as true, then there is no perfect phylogeny.

As we enqueue true nodes, we build a topology for a perfect phylogeny for the subproblem represented by that node, ultimately building one for the whole problem if it exists. We denote the topology of the perfect phylogeny for $[G, x]$ by $T[G, x]$. We enqueue $[G, x]$ when a bundle $[H_1, y], [H_2, y], \ldots, [H_p, y]$ is found such that each $[H_i, y]$ has been determined to be true, and hence a topology $T[H_i, y]$ for each subproblem has already been determined. We create a new node v. If $x \in S$, then we label the node x. Otherwise it remains unlabeled for now. A method for labeling these nodes is given in the proof of Theorem 4.6. *We take each of the trees $T[H_1, y], T[H_2, y], \ldots, T[H_p, y]$, merge the roots into a single node, and make this node a child of node v. Once $[G, x]$ has been enqueued, we construct the tree $T[G, x]$ and we do not consider any more edges entering $[G, x]$. Thus we compute only one topology per true subproblem.*

LEMMA 4.5. *If there exists a perfect phylogeny for $S \cup \{x\}$, then the algorithm PHYLOGENY(S) assigns true to $[G, x]$, for each $G \in (S - \{x\})/x$.*

Proof. The proof is by induction on $|G|$. The base case is trivial. Suppose that the claim holds for all nodes $[G', x']$ where $|G'| < k$. Consider now the node $[G, x]$ where $|G| = k$. Let T be a perfect phylogeny for $S \cup \{x\}$. Assume that T is a perfect phylogeny where the Hamming distance between every pair of adjacent nodes in the tree is one. Consider T as rooted at x, and let $y = lca_T(G)$. From Lemma 4.3, for each node v in the path between x and y, there is a set of equivalence classes of $(S - \{v\})/v$ whose union equals G. Because $y = lca_T(G)$, y is the first node below x for which there are classes $H_1, H_2, \ldots, H_p$, $p > 1$, of $(S - \{y\})/y$ such that $\cup H_i = G$. For all i, $H_i \cup \{y\}$ has a perfect phylogeny. Since $|H_i| < |G|$ it follows that the algorithm has already determined (correctly) that $[H_i, y]$ is true for each $i = 1, 2, \ldots, p$.

We now need to show that for every node z on the path from y to x, $[G, z]$ is set

to true. This will prove that $[G, x]$ is true.

Consider the node $z = parent(y)$. We have two cases to consider, depending upon whether z and x are distinct. We consider the first case, where $z = x$. The edges $([H_j, y], [G, x]), j = 1, 2, \ldots, p$ constitute a bundle for $[G, x] = [G, z]$ so that $[G, x] = [G, z]$ is also set to true. We now consider the second case, where $z \neq x$. In this case, G is an equivalence class of $(S - \{z\})/z$, so that $[G, z]$ is also a subproblem, and $([G, y], [G, z])$ is an edge in the search graph. Since $[G, y]$ is set to true (by the above analysis) the algorithm also sets $[G, w]$ to true for all w such that $[G, w]$ is a vertex in the search graph. Thus, for each node w on the path from y to x, $[G, w]$ is set to true; setting $w = x$ yields the result. □

THEOREM 4.6. *The algorithm PHYLOGENY(S) runs in time $O(r^{lk+1}lkn)$ and returns "yes" iff S has a perfect phylogeny.*

Proof. If S has a perfect phylogeny, then there is some species x that can be an internal node of the tree. By Lemma 4.5 the algorithm will return "yes." Suppose now that the algorithm returns the answer yes, and suppose the leaf-labeled tree produced is D. We now show that the internal nodes of this tree can be labeled so as to create a perfect phylogeny T with load l. Given a character α and an unlabeled node v, we assign $\alpha(v)$ to be the states i such that for some pair of leaves x and y in different subtrees of $D - \{v\}$, $i \in \alpha(x) \cap \alpha(y)$. This clearly creates a perfect phylogeny, and we now need to show that the load is bounded by l. Suppose for some node v the load exceeds l, so that (without loss of generality) for each of the first $l + 1$ states, $1, 2, \ldots, l + 1$, of α, there are at least two subtrees of v with that state. The node v represents a node $[G, y]$ in the search graph, and since it appears in D there is a bundle $[G_1, z], [G_2, z], \ldots, [G_t, z]$ such that all nodes in the bundle are set to true and z and y have distance one. By the construction of D, the subtrees of v have leaf sets $G_1, G_2, \ldots, G_t, S - G$, where $G_i \in (S - \{z\})/z$ for $i = 1, 2, \ldots, t$ (from which it also follows that $S - G$ is the union of the remaining equivalence classes of $(S - \{z\})/z$). Also by construction, z had load at most l, so that z is labeled with at most l α-states. Thus at least one of the states $1, 2, 3, \ldots, l + 1$ is missing from z; without loss of generality let it be $l + 1$. It is easy to see that if G_i and G_j (for $i \neq j$) both have leaves with state $l + 1$, then they would not be separate equivalence classes in $(S - \{z\})/z$, and similarly if some G_i and $S - G$ both have leaves with state $l + 1$. Hence, this labeling has load bounded by l.

The search graph can be constructed in time $O(nlkr^{lk+1})$ by noting that there are $O(r^{kl})$ nodes in the graph, and for each node the maximum number of incoming edges is $O(nrlk)$ and the maximum number of outgoing edges is $O(rlk)$. The rest of the algorithm can be made to run in $(O(nlkr^{lk+1}))$ time, which is linear in the size of the graph. This can be done by sorting the nodes $[G, x]$ according to the size of G and then processing the nodes $[G, x]$ in terms of increasing $|G|$ values. □

Comment. When individual load bounds l_α are given, the algorithm can be modified to run in $O(r^{L+1}Ln)$, where $L = \sum_{\alpha \in C} l_\alpha$.

4.2. A graph-theoretic algorithm for fixed k and l. In this section we give a graph-theoretic algorithm for the l-load perfect phylogeny problem. The algorithm we present is based upon a characterization of intersection graphs derived from l-load perfect phylogenies as a particular kind of vertex-colored triangulated (i.e., chordal) graphs. On the basis of this characterization we will derive an efficient algorithm for the l-load perfect phylogeny problem when we can fix both l and k.

4.2.1. Preliminary definitions. Let $G = (V, E)$ be a graph. A *vertex coloring* of G is a function color: $V \to Z$. We do not require that color be a *proper* coloring

(a coloring function is proper iff $\forall(v,u) \in E$, color$(v) \neq$ color(u)).

The *neighbor set* $\Gamma(v)$ of a vertex v is the set of all vertices in the graph adjacent to v. A vertex v is *simplicial* if $\Gamma(v)$ is a clique.

Given a graph $G = (V, E)$ and a vertex coloring $c : V \to Z$, a *monochromatic clique* in G is a clique with vertex set $V_0 \subset V$ such that color(v)=color(w) for all $v, w \in V_0$. A graph $G = (V, E)$ is *triangulated* if it has no induced cycles of size four or greater. Given a vertex-colored graph $G = (V, E)$, we say that G is *l-triangulated* if G is both triangulated and has no monochromatic cliques of size greater than l. We say that G has an *l-triangulation* $G' = (V, E')$ if $E \subseteq E'$ and G' is l-triangulated.

Let $I = (S, C)$ be an input to the phylogeny problem. For $\alpha \in C$ we define $\alpha_i = \{s \in S : i \in \alpha(s)\}$. The *partition intersection graph* of I is the vertex-colored graph $(G_I = (V, E)$, color) defined by $V = \{\alpha_i : \alpha \in C\}, E = \{(\alpha_i, \beta_j) : \alpha_i \cap \beta_j \neq \emptyset$, where $i \neq j$ if $\alpha = \beta\}$ and for $\alpha \neq \beta$, color$(\alpha_i) =$ color$(\alpha_j) \neq$ color(β_s). Note that because the input I can have load greater than one, the coloring function color may not be proper.

The main results leading to the algorithm can be paraphrased as follows:

- Let I be an input to the l-load perfect phylogeny problem. Then there is an l-load perfect phylogeny for I iff the partition intersection graph G_I has an l-triangulation.
- Given a graph G which is vertex-colored using k colors (not necessarily properly colored) we can determine in time polynomial in fixed k and l whether G has an l-triangulation and construct the l-triangulation when it does.
- Given an l-triangulation G' of G_I we can construct an l-load perfect phylogeny in polynomial time.

As a consequence, we will provide an algorithm for determining if an l-load perfect phylogeny exists for k polymorphic characters defined on n species in $O((rk^3l^2)^{kl+1} + n(kl)^2)$ time.

4.2.2. Characterization of l-triangulated graphs. There is a well-known characterization of triangulated graphs as intersection graphs of subtrees of a tree [7]. In this section we will look at an extension of this particular characterization for l-triangulated graphs.

The following lemma will be useful in the proof of the characterization and also in later theorems. It describes the number of simplicial vertices in a triangulated graph. The proof is simple and is discussed in [17].

LEMMA 4.7. *Let G be a triangulated graph which is not a clique. Then G has at least two nonadjacent simplicial vertices.*

We can make a similar statement about l-triangulated graphs since these graphs are, by definition, also triangulated.

We now present the characterization of l-triangulated graphs.

THEOREM 4.8. *Let $G = (V(G), E(G))$ be a vertex-colored graph. Then G is l-triangulated iff $\exists$ a tree $T = (V(T), E(T))$ together with functions $\varphi : V(G) \to \{$subtrees of $T\}$ and $\phi : V(T) \overset{\text{bijection}}{\to} \{$maximal cliques of $G\}$ such that*

1. *$(v, w) \in E(G)$ iff $\varphi(v) \cap \varphi(w) \neq \emptyset$,*
2. *$\varphi(v) = \{u \in V(T) : v \in \phi(u)\}$,*
3. *$\forall v \in V(T), \phi(v)$ has at most l vertices of the same color.*

Proof. Suppose a tree T exists together with the functions φ and ϕ. We will first show that this, together with conditions 1 and 2, implies that G is triangulated. Let $\Lambda = a_1 a_2 \cdots a_i a_1, i \geq 4$, be a simple cycle in G. We will show that Λ has a chord.

Working in *mod* i arithmetic, it can be seen that $\varphi(a_j) \cap \varphi(a_{j+1}) \neq \emptyset \ \forall 1 \leq j \leq i$. Let $\varphi(a_j) = T_j$. Thus $V(T_j) \cap V(T_{j+1}) \neq \emptyset \ \forall 1 \leq j \leq i$. It can be seen that $\exists j$ such that $V(T_{j-1}) \cap V(T_j) \cap V(T_{j+1}) \neq \emptyset$, as otherwise T will contain a cycle. Let $u \in V(T_{j-1}) \cap V(T_j) \cap V(T_{j+1})$. Thus $\phi(u)$ contains a_{j-1}, a_j and a_{j+1} and so $(a_{j-1}, a_{j+1}) \in E(G)$. Hence Λ contains a chord and so G is triangulated.

From condition 3, G can have maximum monochromatic clique size l. Thus G is l-triangulated.

We now prove the converse by induction on $|V(G)|$. Suppose the statement is true for all graphs having less than n vertices. Let G be a connected graph with n vertices and suppose G is l-triangulated. Now if G is complete, then T is a single vertex and the result is trivial. Assume that G is connected but not complete. Since G is l-triangulated, from Lemma 4.7, it contains a simplicial vertex v. Let $A = \{v\} \cup \Gamma(v)$. Note that A is a maximal clique of G and contains monochromatic cliques of size at most l. Let $B = \{u \in A : \Gamma(u) \subset A\}$ and let $X = A - B$. Note that B, X, and $V(G) - A$ are nonempty since G is connected but not complete. Observe that $G' = G|(V(G) - B)$ is l-triangulated and has fewer vertices than G. Applying the induction hypothesis, let T' be the tree and φ' and ϕ' be the functions satisfying the conditions of the theorem for G'. There are two cases to handle here. Case 1 is when X is a maximal clique in G' and Case 2 is when it is not (note that X is a clique in both G and G'):

Case 1. We can obtain T, ϕ, and φ from T', ϕ', and φ' as follows: Identify that vertex $v' \in V(T')$ such that $\phi'(v') = X$. Define $\phi(w) = \phi'(w) \ \forall w \neq v'$ and $\phi(v') = A$. Define $\varphi(y) = \varphi'(y) \ \forall y \notin B$ and $\varphi(y) = \{v'\} \ \forall y \in B$.

Case 2. Identify that vertex $v' \in V(T')$ such that $\phi'(v') \supset X$. Create a new vertex v and connect it to v'. Define $\phi(w) = \phi'(w) \ \forall w \neq v$ and $\phi(v) = A$. Define $\varphi(y) = \{v\} \cup \varphi'(y) \ \forall y \in X$ and $\varphi(y) = \{v\} \ \forall y \in B$.

Note that in both cases A contains at most l vertices from the same color class and that T, ϕ, and φ satisfy the stated conditions. □

THEOREM 4.9. *Given an instance I of the l-load perfect phylogeny problem, let G_I be the corresponding partition intersection graph. Then I has a solution iff G_I has an l-triangulation.*

Proof. Let T be the solution to the instance I of the l-load perfect phylogeny problem. By Theorem 4.8 there is a graph G which is l-triangulated and is related to T as mentioned in that theorem. It can be seen that G is a supergraph of G_I. Thus G_I can be l-triangulated.

Suppose G_I can be l-triangulated. Let G be the l-triangulation of G_I. Then there is a tree T associated with G satisfying the conditions of Theorem 4.8. It can be seen that in T all the character states are convex and each vertex in T has a label set containing at most l vertices of the same color. Thus T is a solution to the instance I of the l-load perfect phylogeny problem. □

4.2.3. *l*-triangulating a vertex-colored graph. In this section we turn to the problem of l-triangulating a vertex-colored graph. The solution to this problem makes use of several properties of triangulated graphs and also of a particular class of triangulated graphs called k-trees.

Further definitions. Triangulated graphs admit orderings $v_1, v_2, \ldots, v_n$ on the vertex set such that for each i, $N_i = \Gamma(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is a clique [17]. These orderings are called *perfect elimination schemes*.

Consider a graph $G = (V, E)$ with $|V| = n \geq k$ that contains at least one k-clique. Such a graph G is a *k-tree* if the nodes of G can be ordered $v_1, v_2, \ldots, v_n$, whereby $\Gamma_G(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is a k-clique for all i with $1 \leq i \leq n - k$. A k-tree also

has the following recursive definition: The complete graph on k vertices is a k-tree; if $G = (V, E)$ is a k-tree, and $S \subset V$ is a k-clique, then the graph formed by adding a new vertex v and attaching it to each vertex in S is also a k-tree. Each k-tree may be constructed using several different sequences of these operations. The initial set $S \subset V$ is called a *basis* for the k-tree.

For a graph $G = (V, E)$ and vertex-separator $S \subset V$ with C a component of $G - S$, we define C $\cup$ cl(S) to be the graph formed by adding to the subgraph of G induced by $C \cup S$ sufficient edges to make S into a clique. Let $G = (V, E)$ be a k-colored graph. We say that G is a *(k,l)-partition intersection graph* if (a) the maximum monochromatic clique size is l, and (b) G is edge covered by kl-cliques. Note that the maximum clique size in a (k, l)-partition intersection graph is kl.

The algorithm we present for l-triangulating a vertex-colored graph is based on dynamic programming. We will need the following lemmas in our algorithm.

LEMMA 4.10. *Let $G = (V, E)$ be a connected graph which is vertex-colored (not necessarily properly colored) using k colors with $|V| \geq kl$, where l is the maximum monochromatic clique size in G. Let the maximum clique size in G be kl. Then G has an l-triangulation iff it has an l-triangulation that is a $(kl-1)$-tree.*

Proof. Clearly, if G has an l-triangulation that is a $(kl-1)$-tree, then G has an l-triangulation.

Now suppose that G has an l-triangulation. We will use induction to show that G has an l-triangulation that is a $(kl-1)$-tree. Base case is when $|V(G)| = kl$, i.e., G is a clique. This is already a $(kl-1)$-tree and it is l-triangulated.

Suppose the statement is true for all graphs with less than n vertices ($n > kl$) and containing maximum monochromatic clique size l and maximum clique size kl.

Let G be a graph with $|V(G)| = n$. Since G can be l-triangulated, let G' be the l-triangulation of G. From Lemma 4.7 there are at least two nonadjacent simplicial vertices in G'. Pick that simplicial vertex $v \in V(G')$ such that $G' - \{v\}$ still has maximum clique size kl. Let $\Gamma_{G'}(v)$ denote the neighbor set of v in G'. Observe that $G' - \{v\}$ is an l-triangulation of $G - \{v\}$. Thus, by the induction hypothesis, $G' - \{v\}$ can be l-triangulated into a $(kl-1)$-tree. Let G'' be the $(kl-1)$-tree. Let σ be a perfect elimination scheme for G''. Look at x which is the first vertex in $\Gamma_{G'}(v)$ to appear in σ. There are two cases to handle here. Case 1 is when x is within the sequence of last kl vertices appearing in σ. In this case make v adjacent to all vertices in the last kl positions of σ, except with some vertex $u \notin \Gamma_{G'}(v)$ and color(u) = color(v). The resulting graph is an l-triangulated $(kl-1)$-tree. Case 2 is when x is not within the sequence of the last kl vertices appearing in σ. Let A be the set of vertices following x which are neighbors of x. Clearly, $(\Gamma_{G'}(v) - x) \subset A$. Make v adjacent to all vertices in $\Gamma_{G'}(v)$ and also to all except the one vertex u appearing in $A - \Gamma_{G'}(v)$ such that color(v) = color(u). The resulting graph is an l-triangulated $(kl-1)$-tree.

Thus we have that if G has an l-triangulation then it has an l-triangulation which is a $(kl-1)$-tree. □

LEMMA 4.11. *Let G be a (k, l)-partition intersection graph. Then G can be l-triangulated iff there exists a set $K \subseteq V$ of size $(kl-1)$ which is a separator for G such that for all components C of $G - K$, $C \cup cl(K)$ can be l-triangulated.*

Proof. In Lemma 4.10 it was shown that G has an l-triangulation iff it has an l-triangulation G' which is a $(kl-1)$-tree. If such a G' exists, then G' has a separator of size $kl-1$ which is a clique by [31]. The converse is straightforward. □

We are thus motivated to make the following definition.

DEFINITION 4.12. *Let $G = (V, E)$ be a vertex-colored graph with k colors and with*

maximum monochromatic clique size l. A potential basis for G', the l-triangulation of G, is a subset $V_0 \subseteq V$ such that (a) $|V_0| = kl - 1$ *and* (b) V_0 *is a vertex separator for G. If $V_0 \subset V$ satisfies both these conditions then we say that V_0 is a* potential basis *for G, and call V_0 a pb-set.*

Our dynamic programming algorithm will solve the l-load problem when the input is a (k, l)-partition intersection graph. Because our input graphs may not be (k, l)-partition intersection graphs, we need the following result.

LEMMA 4.13. *Let $G = (V, E)$ be vertex-colored with a coloring function color (using k colors) and assume that the maximum monochromatic clique size is l. Then there exists a (k, l)-partition intersection graph $G' = (V', E')$ such that the following is true:*

- *For every pb-set $S \subseteq V'$ containing $(k-1)$ colors and every component C of $G' - S$, $C \cup S$ has all k colors present.*
- *G can be l-triangulated iff G' can be l-triangulated.*
- *The number of vertices in G' is $|V| + |E|(kl - 2)$.*

Proof. For each edge $e = (v, w)$ in E, add $kl - 2$ vertices and sufficient edges so that the kl vertices together form a clique with k color classes of size l. Call the resultant graph G'.

Clearly, $|V(G')| = |V| + |E|(kl - 2)$. Also, since G' is now a (k, l)-partition intersection graph, every edge in G' is part of some kl-clique. Thus, for every *pb*-set S of G' containing $(k-1)$ colors and for every component C of $G' - S$, $C \cup S$ will have all k colors present.

Finally, suppose G' has an l-triangulation G'_1. Then the subgraph of G'_1 induced by the vertex set $V(G)$ is also l-triangulated [17]. Thus G can be l-triangulated. For the other direction, suppose G has an l-triangulation G_1. Identify the edges in G_1 which were present in G and make each of the edges a part of a new kl-clique. This defines a graph G_1^* which can be verified to be a supergraph of G^* and is also l-triangulated. Thus G can be l-triangulated iff G' can be l-triangulated. □

We now have the basis for an algorithm for computing l-triangulations of vertex-colored graphs:

ALGORITHM B (l-triangulating k-colored graphs).

Step 1. Embed G in a (k, l)-partition intersection graph, G'.

Step 2. Compute all *pb*-sets $V_0 \subseteq V(G')$ and all components C of $G' - V_0$. The subproblems $C \cup cl(V_0)$ are then bucket sorted by size.

Step 3. Use dynamic programming to determine the answers for each subproblem in turn.

Step 4. If there is a *pb*-set V_0 such that for all components C of $G' - V_0$, $C \cup cl(V_0)$ is has an l-triangulation, then return (Yes), else return (No).

It is clear that we need to indicate how we implement Step 3.

Solving subproblems using dynamic programming. We have thus reduced the problem of determining whether the graph G can be l-triangulated to looking at graphs of the form $C \cup cl(S)$, where S is a *pb*-set, C is one of the components of $G' - S$, and we presume G' to be a (k, l)-partition intersection graph.

Rose, Tarjan, and Lueker [32] proved the following lemma about triangulated graphs.

LEMMA 4.14. *Let G be a triangulated graph, σ a perfect elimination scheme for G, and a, b vertices in G. If there is a path P from a to b in G such that every vertex in $P - \{a, b\}$ comes before a and b in the ordering σ, then (a, b) is an edge in G.*

We also observe the following lemma about $(kl - 1)$-trees.

LEMMA 4.15. *If G can be l-triangulated into a $(kl-1)$-tree G', then any $(kl-1)$-clique in G' can be a basis for G'.*

We now prove the following theorem. The proof for this theorem is along the same lines as the proof for Theorem 1 appearing in [27].

THEOREM 4.16. *Let $G=(V,E)$ be a (k,l)-partition intersection graph containing at least $kl+1$ vertices, S_0 pb-set of G, and C a component of $G-S_0$. Then $C\cup cl(S_0)$ can be l-triangulated iff there exists a family $\mathcal{F}$ of l-triangulated $(kl-1)$-trees and a vertex $v\in C$ such that*

1. *For every $F\in\mathcal{F}$ there exists a vertex $x\in S_0$ such that $V(F)=C'\cup cl(S)$, where $S=S_0\cup\{v\}-\{x\}$ and C' is a component of both $G-S$ and $C\cup cl(S_0)-S$.*
2. *$|V(F)|<|V(C\cup cl(S_0))|$, for every $F\in\mathcal{F}$.*
3. *Every two graphs in $\mathcal{F}$ intersect only on $S_0\cup\{v\}$.*
4. *$G|(C\cup S_0)$ is contained in $\bigcup_{F\in\mathcal{F}}F$.*

Proof. It is easy to see that if these conditions hold we can combine the l-triangulated $(kl-1)$-trees in $\mathcal{F}$ into one l-triangulated $(kl-1)$-tree covering $C\cup cl(S_0)$ since they intersect only on $S_0\cup\{v\}$.

For the converse, suppose that $G_1=C\cup cl(S_0)$ can be l-triangulated. Let G' be an l-triangulation of $C\cup cl(S_0)$. By Lemma 4.15 the $(kl-1)$-clique S_0 can be a basis for G'. Let v be the vertex added to the basis S_0 in the construction of G' and let $S'=S_0\cup\{v\}$. Thus there is a perfect elimination scheme for G' in which the vertices of S' occur at the end. We will show that we can decompose $C\cup cl(S_0)$ into the union of l-triangulated $(kl-1)$-trees, T_K, each of which is based upon a $(kl-1)$-clique subset $K\subset S'$. We will then show that each such K forming the basis of one of these l-triangulated $(kl-1)$-trees will be a separator for G, so that T_K-K has components $C_1,\ldots,C_r$. We can then in turn write each T_K as the union of possibly smaller $(kl-1)$-trees, $T_K^i=T_K|(C_i\cup K)$. These l-triangulated $(kl-1)$-trees are the ones of interest.

G' is built by adding vertices, one at a time, and making each new vertex adjacent to every vertex in some $(kl-1)$-clique. We will define G_i to be the subgraph of G' induced by the vertex set $\{v_i,v_{i+1},\ldots,v_{|V|}\}$. Thus $G_{|V|-kl+2}$ is a $(kl-1)$-clique, and to form G_i we make vertex v_i adjacent to every vertex in some $(kl-1)$-clique in G_{i+1}. We will show that we can assign to each added vertex v_i (with $i<(|V|-kl+1)$) a label $L(v_i)$ the *name* of a $(kl-1)$-clique $K\subset S'$, so that for each $K\subset S'$ the subgraph $T_K=G|V_K$, where $V_K=\{v:L(v)=K \text{ or } v\in K\}$, is an l-triangulated $(kl-1)$-tree. We will also show that every edge e in $G|(C-\{v\})$ is in one of these $(kl-1)$-trees and that the $(kl-1)$-cliques K forming the basis of the $(kl-1)$-trees T_K are separators of G. We will also need to show that the component C' of $C\cup cl(S_0)-L(v)$ containing v is a component of $G-L(v)$. This will prove our assertions.

We first need to show how we assign vertices to $(kl-1)$-clique subsets of S'. Let L be the assignment function we wish to define for every vertex not in S'. Suppose we have constructed the graph G_{i+1} and are now adding v_i to the graph and making it adjacent to every vertex in some $(kl-1)$-clique, R. If $R\subset S'$, then we set $L(v_i)=R$. Otherwise, the vertices in R will consist of (perhaps) some unlabeled vertices (these will be in S') and at least one labeled vertex. If all of the labels in R agree, then this is the label that we will assign to v_i. On the other hand, suppose for our construction that when we make v_i adjacent to every vertex in the $(kl-1)$-clique R not all the labels are the same and that this is the first vertex in this construction for which this happens. In this case, for some vertices v_j and v_k in R, $L(v_j)=X$

and $L(v_k) = Y$, for distinct subsets $X, Y \subset S'$. Without loss of generality we can assume that $i < j < k$. In constructing G_j we made v_j adjacent to every vertex in some $(kl-1)$-clique $C \subset G_{j+1}$. Note that $v_k \in C$ since v_j and v_k are adjacent and $k > j$. Since we were able to set $L(v_j) = X$ unambiguously, this means that either every vertex in C was unlabeled, and thus $X = C$, or that the labeled vertices were all labeled X. Since we have assumed v_k was labeled, we can infer that $L(v_k) = X$ and hence $X = Y$. Thus this assignment of vertices to $(kl-1)$-clique bases is well defined, and each label denotes a subset K of S'. It is easy to see that the subgraph $T_K = G'|V_K$ (for $V_K = \{v : L(v) = K \text{ or } v \in K\}$) is an l-triangulated $(kl-1)$-tree and that T_K is based upon the set K.

By our construction of the labeling function, it is also clear that no edge in G has different labels at its endpoints, so that every edge in $G|(C - \{v\})$ is in exactly one l-triangulated $(kl-1)$-tree, T_K.

We now show that each $(kl-1)$-clique $K \subset S'$ forming the basis of an l-triangulated $(kl-1)$-tree in $\mathcal{F}$ is a separator for $C \cup cl(S_0)$ and for G. We first show that K is a separator for $C \cup cl(S_0)$. Suppose to the contrary, so that for some set $K \subset S$ forming the basis of an l-triangulated $(kl-1)$-tree T_K, $C \cup cl(S_0) - K$ is connected. Let $K = S - \{x\}$. We will show that there is no path from x to any vertex in $C \cup cl(S_0) - K$. Let σ' be a perfect elimination scheme for $T_K \cup \{x\}$. Clearly, we can assume that x is the last vertex in σ' to occur before the vertices of K. Let a be the vertex immediately preceding x. If there is a path from x to a in $C \cup cl(S) - K$, then the edge (a, x) is in G by Lemma 4.14. But then $S \cup \{a\}$ is a $(kl+1)$-clique, contradicting that G has a supergraph which is a $(kl-1)$-tree. The proof can be modified to show that K is a separator for G as well. Hence the $(kl-1)$-trees T_K^i each contain fewer vertices than G.

We now complete our proof by showing that the components of $C \cup cl(S_0) - K$ are also components of $G - K$, where K is the basis of a $(kl-1)$-tree $F \in \mathcal{F}$. Recall that by our construction each such basis K is a set $L(a)$ for some $a \in V(F) - K$. So let C' be a component of $C \cup cl(S_0) - L(a)$, for some $a \in C$. It is easy to see that $L(a) = S_0 \cup \{v\} - \{x\}$ is a separator for $C \cup cl(S_0)$ and that every component X of $C \cup cl(S_0) - L(a)$, such that $x \notin X$, is also a component of $G - L(a)$. Thus we will show that $x \notin C'$, so that C' is a component of $G - L(a)$.

Suppose $x \in C'$. Then x is adjacent to at least one vertex z of $C' - \{x\}$. When we labeled the vertex z we labeled it with $L(a)$ implying that $x \in L(a)$, and yet, by our construction, $x \notin L(a)$. Hence the component C' of $C \cup cl(S_0) - L(a)$ containing a is a component of $G - L(a)$. This completes our proof. □

We can now state the following theorem.

THEOREM 4.17. *Let $G = (V, E)$ be a (k, l)-partition intersection graph with $|V| \geq kl+1$. Let S_0 be a pb-set and let C be a component of $G - S_0$. Then $C \cup cl(S_0)$ can be l-triangulated iff there exists some vertex v in C and a family of pb-sets $\mathcal{M}$ such that the following is true:*

1. *For each $M \in \mathcal{M}$, $M \subset S_0 \cup \{v\}$ and M is a separator for $C \cup cl(S_0)$ and for G.*
2. *For each vertex $x \in S_0$ there is an $M_x \in \mathcal{M}$ and a component C_x of $G - M_x$ and of $C \cup cl(S_0) - M_x$ such that $|C_x| < |C|$ and $C_x \cup cl(M_x)$ can be l-triangulated.*
3. *Every edge in C is in exactly one C_x given above.*

Proof. Suppose that $C \cup cl(S_0)$ can be l-triangulated and let G' be a l-triangulation of $C \cup cl(S_0)$. From Theorem 4.16 we infer that there is a vertex $v \in C$ such that the

subgraph of G' induced by the vertices of $C \cup cl(S_0)$ can be written as the union of the l-triangulated $(kl-1)$-trees T_K based upon *pb*-sets $K \subset S' = S_0 \cup \{v\}$. We will let $\mathcal{M}$ consist of these subsets K, which form the bases of the $(kl-1)$-trees T_K. From Theorem 4.16 it can be seen that $\mathcal{M}$ satisfies the conditions above.

For the converse, if such a family $\mathcal{M} = \{M_i : i \in I\}$ of *pb*-sets exists, then there exists $v \in C$ such that the graph $C \cup cl(S_0)$ is contained in the union of l-triangulatable graphs of the form $C_x \cup cl(M)$, where each $M \in \mathcal{M}$ is a *pb*-set and a subset of $S_0 \cup \{v\}$ and C_x is a component of $G - M$ and a proper subset of C. Since G is a (k,l)-partition intersection graph, these graphs each have all k colors and have monochromatic cliques of maximum size l and also have cliques of maximum size kl. Hence they can be completed to l-triangulated $(kl-1)$-trees T_x, where $V(T_x) = V(C_x \cup M)$. This family of $(kl-1)$-trees $\mathcal{F} = \{T_x : x \in C - \{v\}\}$ shows that $C \cup cl(S_0)$ can be l-triangulated. □

THEOREM 4.18. *Let $G = (V, E)$ be a (k,l)-partition intersection graph, $S \subset V$ be a pb-set, and C be a component of $G - S$. Then we can determine whether $C \cup cl(S)$ can be l-triangulated simply by knowing the "answer" for each smaller graph of the form $C' \cup cl(S')$, where S' is a pb-set and C' is a component of $G - S'$.*

Implementation details of the dynamic programming algorithm (Step 3 of Algorithm B). Data structure: A family $\mathcal{X} = \{M_i\}$ of *pb*-sets. For each set M_i in $\mathcal{X}$, and for each of the r_i components C_j of $G - M_i$, we denote by M_i^j, $j = 1, 2, \ldots, r_i$, the subgraph of G induced by $C_j \cup M_i$ with the addition of edges required to make M_i into a clique. Each such M_i^j either can be l-triangulated or cannot be. This will be determined during the algorithm, in order of increasing size of the M_i^j's, and an appropriate answer (yes or no) will be stored for each.

Recall that Step 2 of Algorithm B sorts the subproblems $C_j \cup M_i$ using bucket sort.

ALGORITHM. (the statements in italics denote comments).

(Examine the M_i^j in turn by order of number of vertices,*
and determine whether each can be l-triangulated.
Any graph containing all k colors with
*l vertices per color class can be l-triangulated *)*

IF M_i^j has kl vertices with l vertices per color class, **THEN** set its answer to Yes.

IF M_i^j has kl vertices such that there is one color class with more than l vertices, **THEN** set its answer to No.

(We will now apply Theorem 4.17 to each graph M_i^j*
and search for a vertex $v \in M_i^j - M_i$
and family $\mathcal{M}$ satisfying the conditions of Theorem 4.17
*to determine whether M_i^j can be l-triangulated *)*

FOR EACH graph M_i^j in order of size $h > kl$ **DO**

FOR EACH $v \in M_i^j - M_i$ such that $M_i \cup \{v\}$ has no color class containing more than l vertices, **DO**

(We now check whether for vertex v there is a*
*family $\mathcal{M}$ satisfying the conditions of Theorem 4.17 *)*

Examine all sets M_m of vertices in $M_i \cup \{v\}$ which are *pb*-sets for G

FOR EACH such M_m, let L_m be the union of the M_m^j which can be

l-triangulated
IF the union of the L_m (for each
M_m above) contains $M_i^j - M_i - \{v\}$
THEN set the answer of M_i^j to
Yes and **EXIT-DO**
END-DO
IF no answer was set for M_i^j
THEN set the answer for M_i^j to No
*(*Applying Lemma* 4.11 *now*)*
IF G has a vertex-separator M_i such that
all M_i^j graphs have the answer Yes,
THEN (G can be l-triangulated)
RETURN (Yes)
ELSE RETURN (No)
END-DO
end of algorithm.

The above algorithm can be modified easily to give back an l-triangulation, if it exists.

Run time analysis of Algorithm B. Let $G = (V, E)$ be the (k, l)-partition intersection graph which is given as input to Step 2 of Algorithm B. Then, in Step 2, in the worst case the algorithm checks all subsets of size $kl - 1$ of which there are $O(|V|^{kl-1})$. Each of these is checked for being a pb-set, which involves checking the set to see if it is a vertex separator. This takes $O(|V|^2)$ for each subset. Bucket sorting the subproblems takes a total of $O(|V|^{kl})$. Step 3, which involves checking to see if a subgraph satisfies the conditions of Theorem 4.17, takes time linear in the number of vertices in the subgraph. Thus the overall complexity is $O(|V|^{kl+1})$.

We summarize with the following.

THEOREM 4.19. *Let $G = (V, E)$ be a (k, l)-partition intersection graph. We can in $O(|V|^{kl+1})$ time determine whether G can be l-triangulated and produce the l-triangulation when it exists.*

4.2.4. Summary of the algorithm to solve the l-load perfect phylogeny problem. Given I, compute the partition intersection graph, G_I, and embed G_I in a (k, l)-partition intersection graph G'_I. Use Algorithm B to determine if G'_I can be l-triangulated, and compute the triangulation $G = (V, E)$ if it exists. If there is no l-triangulation, return No. Else, use G to compute the l-load perfect phylogeny T.

We now briefly discuss how T can be constructed from the l-triangulated graph $G = (V, E)$. Recall that T is related to G by Theorem 4.8.

Let $\sigma = v_1 v_2 \dots v_{|V|}$ be a perfect elimination scheme for G. We will construct the tree inductively where T_i is the tree corresponding to $G|\{v_i, v_{i+1}, \dots, v_{|V|}\}$. Thus $T_1 = T$ is the tree we seek.

Let $A_p = \Gamma(v_p) \cap \{v_{p+1}, v_{p+2}, \dots, v_{|V|}\}$. Inductively, assume we are at vertex v_j in σ and assume we have the tree T_{j+1}. Let v_i be the first vertex following v_j (i.e., $j < i$) in σ which is in $\Gamma(v_j)$. Note that $(A_i \cup \{v_i\}) \supseteq A_j$. Since v_j and v_i are simplicial in $G|\{v_j, v_{j+1}, v_{j+2}, \dots, v_{|V|}\}$ and $G|\{v_i, v_{i+1}, v_{i+2}, \dots, v_{|V|}\}$, respectively, it follows that $|\{v_i\} \cup A_i| > |A_j|$ iff, in $G|\{v_j, v_{j+1}, v_{j+2}, \dots, v_{|V|}\}$, the subgraph induced by $\{v_i\} \cup A_i$ is a maximal clique.

Case 1. If the subgraph induced by $\{v_i\} \cup A_i$ is a maximal clique then it follows that $\{v_j\} \cup A_j$ also induces a maximal clique in $G|\{v_j, v_{j+1}, v_{j+2}, \dots, v_{|V|}\}$. Thus from Theorem 4.8 in T_j there will be a vertex which corresponds to $\{v_j\} \cup A_j$. To

get T_j from T_{j+1}, we add a new vertex u to $V(T_{j+1})$ and add an edge from u to the vertex $u' \in V(T_{j+1})$ which corresponds to the maximal clique $\{v_i\} \cup A_i$.

Case 2. If $|\{v_i\} \cup A_i| = |A_j|$, then the subgraph induced by $\{v_i\} \cup A_i$ in $G|\{v_j, v_{j+1}, v_{j+2}, \ldots, v_{|V|}\}$ is not a maximal clique. Let v_q $(j \leq q)$ be the first vertex to the left of v_i in σ such that $v_i \in \Gamma(v_q)$ and $|\{v_i\} \cup A_i| = |A_q|$. If $q = j$, then to get T_j we relabel the vertex $u \in V(T_{j+1})$ corresponding to the maximal clique $\{v_i\} \cup A_i$ to now correspond to $\{v_j\} \cup A_j$. If $q \neq j$, then to get T_j from T_{j+1} we create a new vertex corresponding to $\{v_j\} \cup A_j$ and connect it to the vertex $u' \in V(T_{j+1})$ which corresponds to $\{v_q\} \cup A_q$.

It can be seen that the above operations of obtaining T_j from T_{j+1} can be implemented in $O(deg(v_j))$, where $deg(v)$ is the degree of vertex v. This can be achieved by associating two variables with every vertex $v_r \in \sigma$ $(j < r)$, one that corresponds to the vertex $u \in V(T_r)$ such that u represents the maximal clique $\{v_r\} \cup A_r$ in $G|\{v_r, v_{r+1}, \ldots, v_{|V|}\}$ and one that corresponds to a vertex v_s in σ such that v_s is the first vertex to the left of v_r for which $v_s \in \Gamma(v_r)$ and $|\{v_r\} \cup A_r| = |A_s|$.

The time taken for producing a perfect elimination scheme for G is $O(|V| + |E|)$ [17]. From the discussion above, it can be seen that T can then be constructed in $O(|V| + |E|)$. Hence we have the following theorem.

THEOREM 4.20. *Let $G = (V, E)$ be a vertex-colored graph which is l-triangulated. Then the tree T which satisfies the conditions in Theorem* 4.8 *can be constructed in $O(|V| + |E|)$.*

THEOREM 4.21. *The l-load perfect phylogeny problem for n species and k polymorphic characters can be solved and the l-load perfect phylogeny constructed (when it exists) in $O(nk^2l^2 + (rk^3l^2)^{kl+1})$ time.*

Proof. Let I be the input to the l-load perfect phylogeny problem and $G_I = (V, E)$ be the partition intersection graph. Then $|V| = rk$, and it can be shown that if $|E| > kl|V|$ then there is no l-triangulation [27]. Hence $|E| \leq k^2lr$. Let $G'_I = (V', E')$ be the (k, l)-partition intersection graph embedding of G_I, and note that $|V'| = |V| + |E|(kl - 2) \leq rk + k^3l^2r$. The rest follows. □

Comment. In the case where individual load bounds l_α are given, the algorithm can be modified to run in $O(nL^2 + (rkL^2)^{L+1})$, where $L = \sum_{\alpha \in C} l_\alpha$.

4.3. Inferring perfect phylogenies from mixed data. In the previous section we presented two algorithms for inferring perfect phylogenies from polymorphic character data; these algorithms had running times which were exponential in L, where $L = \sum_{\alpha \in C} l_\alpha$ and l_α is the load bound for the character α. We can use these algorithms directly for sets of characters when some of the characters are monomorphic and some are polymorphic, but the expense would be too large. This follows since in typical data sets the number of characters k is the largest parameter, often in the hundreds or thousands; since $L > k$, algorithms that are exponential in L are prohibitively costly. Instead, we propose a method that should be efficient when the number of monomorphic characters is sufficient to reduce the number of minimal perfect phylogenies to a small number. In practice, as the majority of the characters will be monomorphic, this is likely to be very efficient. The method we propose involves two steps and is efficient when the number of minimal perfect phylogenies generated from the monomorphic characters is small.

ALGORITHM C.

Step 1. Infer all minimal perfect phylogenies from the monomorphic characters, using [23].

Step 2. Determine whether any of the minimal perfect phylogenies obtained in Step 1 can be refined so that each polymorphic character is convex on it within the specified load bound.

Discussion of Step 1. The algorithm by Kannan and Warnow [23] has running time of $O(2^{2r+r^2}k_m^{r+3}+Mk_mn)$, where M is the number of minimal perfect phylogenies and k_m is the number of monomorphic characters. This is theoretically expensive if r, the number of states, is too large; however, in practice the algorithm works quickly as long as not too many of the characters have a large number of states. Also, in practice, as long as the monomorphic characters are independent of each other and comprise a suitably large set, there will be very few perfect phylogenies. Thus we expect Step 1 to be very fast and to produce very few minimal perfect phylogenies.

Discussion of Step 2. We consider the following problem.

Problem. Refining a tree.

Input. Leaf-labeled tree T and set C of polymorphic characters, each with an individual load bound.

Question. Does a perfect phylogeny T' exist for the polymorphic characters, subject to the constraint that T' is a refinement of T?

ALGORITHM D.

For each internal $v \in T$ which has degree greater than 3, do

1. Let $\Gamma(v) = \Gamma_1(v) \cup \Gamma_2(v)$, where $\Gamma_1(v)$ consists of all the neighbors of v which are leaves and $\Gamma_2(v)$ consists of all the nonleaf neighbors of v. For each $u_j \in \Gamma_2(v)$ add a new node w_j on the edge (v, u_j). Compute the labeling of w_j so as to make every character convex (each character must contain every state that appears on both sides of w_j).
2. If some new node has a load for a character that exceeds the stated load bound for that character, RETURN(NO). Let $S_v = \Gamma_1(v) \cup \{w_j | w_j$ is a new node and w_j is a neighbor of $v\}$. Use any of the algorithms from section 3 to determine if there is a perfect phylogeny for (S_v, C) satisfying the load bounds. If any (S_v, C) fails to have a perfect phylogeny meeting the load bounds then RETURN(NO), else RETURN(YES).

Example. Consider $S = \{a, b, c, d, e, f, g\}$ and $C = \{\alpha, \beta\}$. Let α be a monomorphic character and β be a polymorphic character with load bounded by three. Also, let $\alpha(a) = \{0\}, \alpha(b) = \{0\}, \alpha(c) = \{1\}, \alpha(d) = \{1\}, \alpha(e) = \{1\}, \alpha(f) = \{1\}$, and $\alpha(g) = \{1\}$. In addition, let $\beta(a) = \{0\}, \beta(b) = \{3\}, \beta(c) = \{2, 4\}, \beta(d) = \{2, 3\}, \beta(e) = \{1, 4\}, \beta(f) = \{1, 3\}$, and $\beta(g) = \{0, 4\}$. Figure 4.1(i) shows the phylogeny T obtained by applying the perfect phylogeny algorithm to the monomorphic character α. Let $S' = \{h, c, d, e, f, g\}$ be the species set obtained at the end of step 1 of Algorithm D. Then $\beta(h) = \{0, 3\}$. The instance $(S', \{\beta\})$ has a perfect phylogeny with load bounded by three. This is shown in Figure 4.1. From this phylogeny it is possible to obtain the solution to the instance (S, C), and this is shown in Figure 4.1(iii).

THEOREM 4.22. *Algorithm D correctly determines whether a perfect phylogeny T' exists refining T within the stated load bounds, and it can be modified to produce the perfect phylogeny T' in time $min\{O(r^{L+1}Ln^2), O(n^2L^2 + n(rkL^2)^{L+1})\}$.*

Proof. If the algorithm returns NO, it is clear that no perfect phylogeny within the constraints of the problem exists. If it returns YES, then the perfect phylogenies refining each of the stars can be hooked up via the new nodes. The refinement can be done by using the algorithms in section 4. It can be shown that the algorithm takes $min\{O(r^{L+1}Ln^2), O(n^2L^2 + n(rkL^2)^{L+1})\}$. □

In Algorithm D, if $|S_v|$ is small then it may be cheaper in practice to look at all

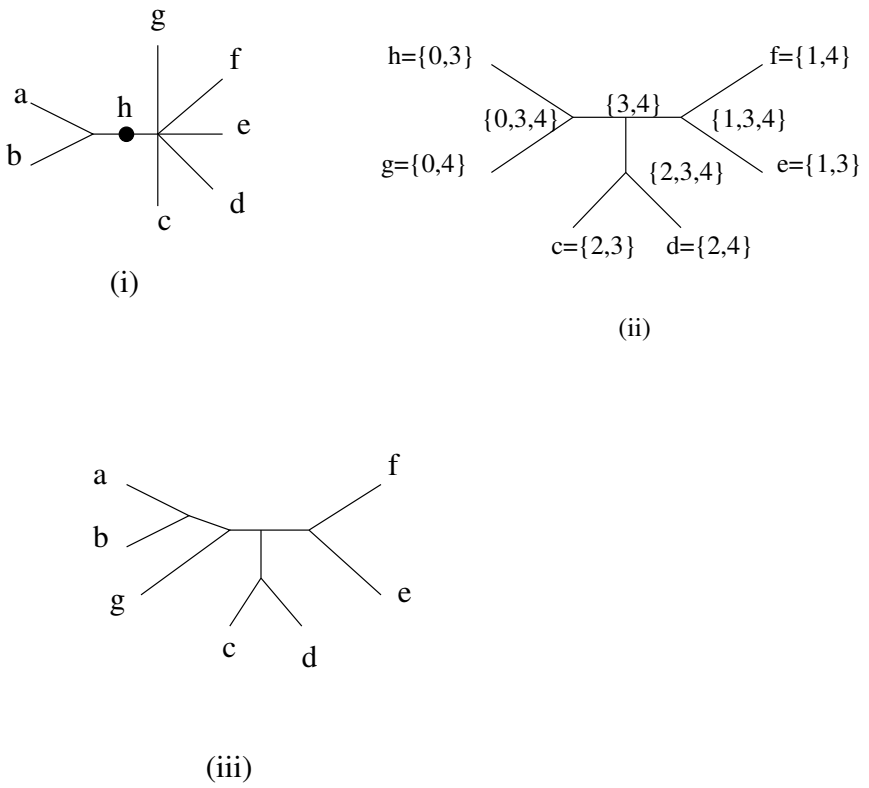

FIG. 4.1. *Example for* Algorithm D.

possible leaf-labeled topologies on S_v rather than use the algorithms of section 4 to determine the existence of perfect phylogenies on S_v.

5. Polymorphism in linguistics. Properly chosen and encoded characters in linguistics have been shown to be convex on the true tree, so that with proper scholarship we should be able to infer a *perfect phylogeny*. In recent work on an Indo-European data set, Warnow, Ringe, and Taylor [39] found that there was extensive presence of polymorphic characters. The degree of polymorphism for each polymorphic character could be determined from the data with high confidence, so that the question of inferring the correct tree amounted to determining if a perfect phylogeny existed in which each character was permitted a maximum degree of polymorphism (i.e., load) on the tree. Figure 5.1 shows the tree that they now posit. This was obtained using Algorithm D. This tree is in fact different from their earlier hypothesis and from the tree that was presented in [6]. This tree has been obtained as a result of using more data. This tree shows a limited support for Indo-Hittite, moderate support for the Italo-Celtic hypothesis, and significant support for a subgroup of Greek and Armenian.

6. Polymorphism in biology. The evolution of biological polymorphic characters can be modeled using the following operations [28]. A *mutation* changes one state into another. A *loss* drops a state from a polymorphic character from parent to child. A *duplication* replicates a state which subsequently mutates. This allows children to have higher load on a polymorphic character than their parents. We con-

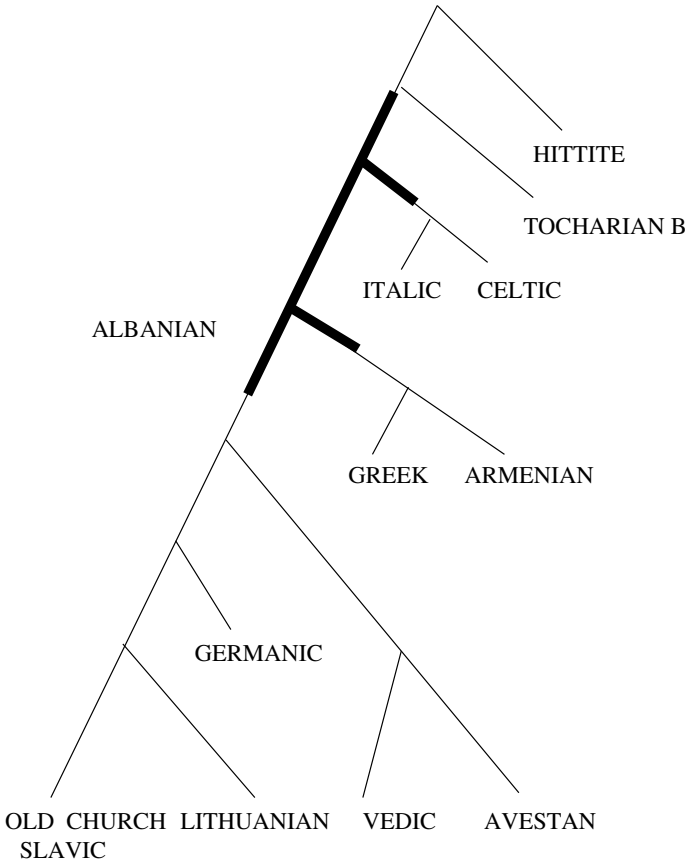

FIG. 5.1. *The tree on the Indo-European data set. Albanian can be on any of the thick edges. The tree indicates only a rooted topology without any edge lengths.*

sider two types of costs: (a) state-independent costs, in which any loss costs $cost_\ell$, any mutation costs $cost_m$, any duplication costs $cost_d$, and any match costs 0; and (b) state-dependent costs, in which the costs are dependent on the states involved.

Parsimony is a popular criterion for evaluating evolutionary trees from biomolecular data. A most parsimonious tree T minimizes $\sum_{e \in E(T)} cost(e)$. Traditionally, for monomorphic characters $cost(e)$ is the Hamming distance of the labels at the two endpoints of e. For unknown topology, the traditional parsimony problem is *NP-hard* [9, 10], but for fixed topology it is in P [16].

Consider the case where costs $cost_\ell, cost_m$, and $cost_d$ are not state-dependent. Let (u, v) be an edge in T with u above v. We define the cost $cost(\alpha, (u, v))$ of $\alpha \in C$ on (u, v) as follows: Let $X = \alpha(u) - \alpha(v), Y = \alpha(v) - \alpha(u)$, and $Z = \alpha(u) \cap \alpha(v)$.

- If $|X| = |Y|$ then $cost(\alpha, (u, v)) = cost_m|X|$ (all events are mutations, but shared states do not change).
- If $|X| > |Y|$ then $cost(\alpha, (u, v)) = cost_\ell[|X| - |Y|] + cost_m|Y|$.
- If $|X| < |Y|$ then $cost(\alpha, (u, v)) = cost_d[|Y| - |X|] + cost_m|X|$.

The cost of the edge (u, v) is then $\sum_{\alpha \in C} cost(\alpha, (u, v))$. For state-dependent costs we must also match states in the parents to states in the child for mutation and duplication events.

We consider the following problem: Given a fixed leaf-labeled topology and a maximum load l, what is the most parsimonious labeling of the internal nodes?

The problem is NP-complete for arbitrary loss, mutation, and duplication cost functions. If $cost_\ell = 0$, such as when we wish to maximize convexity, the problem becomes even harder.

THEOREM 6.1. *The following problems are NP-complete:*

- *Given a tree with leaves labeled by species each with load at most l and a value P, determine if the internal nodes can be labeled to create a phylogeny with load at most l and parsimony cost at most P for arbitrary $cost_\ell < cost_m < cost_d$.*
- *If $cost_\ell = 0$ and $cost_m \leq cost_d$ are arbitrary then given a tree with leaves labeled by species and values l and P, determine if the internal nodes can be labeled to create a phylogeny with load at most l and parsimony cost at most P. This problem remains NP-complete even if the tree is binary, no edges of weight 0 are allowed, and the input load is $1 \leq l_i \leq l$.*

Proof. In the fixed-topology setting, characters are independent. Therefore we consider only the case of a single character with r states.

Clearly the problem is in NP. We now show it is NP-hard. Our reduction is from the *three-dimensional matching problem* (3DM), known to be NP-complete [20], which is defined as follows. We are given three disjoint sets, A, B, and C, each with n elements, and a set X of m *triples*, $X = \{(a_i, b_j, c_k) : a_i \in A, b_j \in B, \text{and } c_k \in C\}$. We say that triple (a_i, b_j, c_k) *covers* a_i, b_j, and c_k. We wish to find a set of n triples that covers every element of A, B, and C exactly once. This set of n triples is called a *perfect matching*.

Given an instance of 3DM, we construct a tree T with leaves labeled by species each with load at most $m-n$. The internal nodes of T can be labeled with load $m-n$ and parsimony $(3mn - 3n^2)cost_m$ iff the instance of 3DM has a perfect matching.

We construct the tree T as follows. We begin by creating an internal *root* node. This root has $3n$ children, $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, and $c_1, \ldots, c_n$, which are all internal nodes. Let $n(a_i)$ for $1 \leq i \leq n$ be the number of triples that contain a_i. We have the following states for our character: m states $x_1, x_2, \ldots, x_m$ corresponding to the m triples $x_j \in X$, and $d(a_i) \equiv m - n - n(a_i) + 1$ *dummy states* associated with each a_i (similarly we have $d(b_j) \equiv m - n - n(b_j) + 1$ dummy states for each b_j and $d(c_k) \equiv m - n - n(c_k) + 1$ dummy states for each c_k). Let $D(a_i)$ be the set of dummy states associated with a_i ($|D(a_i)| = d(a_i)$). Let $X(a_i)$ be the set of triples that contain a_i ($|X(a_i)| = n(a_i)$). For the remainder of this discussion we will concentrate on nodes a_i. The nodes b_j and c_k are treated symmetrically.

Node a_i has $n(a_i)$ leaf children. Let $x_1, x_2, \ldots, x_{n(a_i)}$ be the states associated with the triples that contain a_i. The ith leaf under node a_i has all the dummy states $D(a_i)$ associated with a_i and all of $x_1, x_2, \ldots, x_{n(a_i)}$ *except for* state x_i. Each child thus has load $m - n$.

It can be shown that we can label the internal nodes of this tree with load at most $m - n$ and cost at most $(3mn - 3n^2)cost_m$ iff the instance of 3DM has a perfect matching.

First suppose that $X_s \subset X$ is a set of n triples that forms a perfect matching. Label the root with states from the set $X - X_s$ and label a_i with all the leaves below it except x_q, where $x_q \in X_s$ and $a_i \in x_q$. Each internal node has load $m - n$ as required. The edge from the root to node a_i has cost $d(a_i)cost_m$ since none of the dummy states in $D(a_i)$ are in the root, all of the remaining states in a_i's label are

in the root and the root has the same load as a_i (thus we have mutation rather than duplication or loss). Since $\sum_{i=1}^{n} d(a_i) = mn - n^2 + n - m$, we have (since the edges from the root to the b_j and c_k nodes are of similar cost) that the total cost of the edges from the root to its $3n$ children is $3mn - 3n^2 + 3n - 3m$. Now consider the cost of the edges from node a_i to its $n(a_i)$ children. If x_q is the triple in X_s that contains a_i, then the edge to the child missing x_q will have cost 0. All other edges have cost $cost_m$. Summing over all edges from a_i, b_j, or c_k nodes to their children, we have a cost of $(3m - 3n)cost_m$. Thus the total cost is $(3mn - 3n^2)$ and the parsimony bound is met.

Suppose instead that there is a labeling of the internal nodes of T so that the maximum load is $m - n$ and the total cost is at most $3mn - 3n^2$.

We need the following lemma.

LEMMA 6.2. *If $cost_m < cost_d$, then in any optimal solution an internal node will always have load at least as high as the minimum load of any of its children.*

Proof. Consider a node p that is the parent of k children. Suppose there is a labeling of the nodes such that node p has a load smaller than all of its children. Thus in the cost of the tree there are at least k duplications associated with edges from p to its children. If we add to the label of p another state found in at least one of its children (such a state always exists since all children have more labels than p), then regardless of the labeling of p's parent we will decrease the cost of the tree. Adding the label costs at most $cost_d$ along the edge from node p to its parent. However, it saves at least $cost_d + (k-1)(cost_d - cost_m)$. Since $cost_d > cost_m$, adding the label always results in a net savings. Therefore we can assume that in the labeling we are given all internal nodes have load $m - n$, or we can add labels to these internal nodes and only reduce the cost. □

Thus from Lemma 6.2 all internal nodes of the input tree will have load $m - n$. Looking at the root, each state x_i is contained in the leaves of exactly three children. Each dummy state is contained in the leaves of only one child. Therefore a lowest-cost labeling of the root will have $m - n$ states x_i. If all the a_i, b_j, and c_k are also labeled by the states chosen by the root, then the minimum possible cost of the edges from the root to its children is $3mn - 3n^2 - 3m + 3n$. This is because each child a_i must mismatch on at least $m - n - n(a_i)$ labels. Summing over all children of the root, this give $3mn - 3n^2 - 3m$, but since n triples could not be in the label of the root, there is an additional cost of $3n$. Looking at the parsimony bound, even if the cost of the edges from the root to its children is minimum, the total cost of the a_i, b_j, and c_k nodes to their children can be at most $3m - 3n$. Consider a node a_i and its children. The minimum possible cost of the edges between these nodes is $n(a_i) - 1$, which comes from labeling a_i with all its dummy states and all but one of the triple states associated with it. If a_i is instead labeled with all of its triple states and all but one of its dummy states, the cost is $n(a_i)$. To achieve a total remaining cost of $3m - 3n$, however, the cost of each a_i to its children must be the minimum and therefore one of the triple states is missing from each a_i, b_j, and c_k. These must be the n triple states missing in the root or there will be higher cost associated with the edges from the root to its children and the parsimony bound will be exceeded. Thus the n triple states missing from the label of the root cover each of the a_i, b_j, and c_k and correspond to a perfect matching for the 3DM instance.

We now prove the second part of Theorem 6.1. Clearly the problem is in NP. We now show it is NP-hard. We again use a reduction from 3DM as in the proof of the first part of Theorem 6.1. We construct the tree as above with the following

modifications. Each node a_i now has two children. For the case of load-1 input, each child is the root of a binary tree. Each of these trees has all the dummies in $D(a_i)$ represented in the leaf set and the states of $X(a_i)$ are arbitrarily divided among the children, appearing as a leaf just once in the subtree rooted at a_i. For other input loads, the labels of the leaves vary. For instance, for load L there are only two leaf children of a_i, one labeled with all the dummies in $D(a_i)$ and all but one state in $X(a_i)$, the other labeled with all the dummy states and the single state $x_q \in X(a_i)$ missing in the label of its sibling. For other loads, the children of a_i can also be made into binary trees where the input load is met by at least one leaf, all dummy states are represented in each child of a_i, and each state in $X(a_i)$ is represented exactly once. To make the whole tree binary, we form an arbitrary binary tree with the a_i as "leaves." (The two children of a_i will be attached.) We call this tree (without the children of a_i) the *A tree*. We make the root of the A tree a child of the global root. Similarly we form a B tree and a C tree and make them children of the global root.

Again, it can be shown that we can find labels for the internal nodes of this tree with load at most $m-n$ and cost at most $(3mn+6n-3n^2-3m)cost_m$ iff the instance of 3DM has a perfect matching.

First suppose that $X_s \subset X$ is a set of n triples that forms a perfect matching. Label the root with states from the set $X - X_s$ and label a_i with all the leaves below it except x_q where $x_q \in X_s$ and $a_i \in x_q$. Each internal node in the A tree is labeled with all of the x_q states appearing in the labels of its two children. It then picks an arbitrary set of the dummy states appearing in its two children so that its final load is $m - n$. Thus each internal node has load at most $m - n$. We now calculate the cost of this labeling. Each dummy state associated with a_i arises once in the A tree by mutation since all internal nodes in the A tree have load $m-n$, as does the global root, and the global root is not labeled with any dummy states. The total cost of all dummies is $(3mn + 3n - 3n^2 - 3m)cost_m$. In addition each of the n triple states not represented in the root arise three times in the tree by mutation for a total cost of $3ncost_m$. Thus the tree costs $(3mn+6n-3n^2-3m)cost_m$ and the parsimony bound is met.

Suppose instead that there is a labeling of the internal nodes of the tree so that the maximum load is $m-n$ and the total cost is at most $(3mn--+6n-3n^2-3m)cost_m$.

We need the following lemma.

LEMMA 6.3. *If $cost_\ell = 0$, then there exists an optimal solution where each internal node contains all the states in the subtree rooted at it or has maximum load.*

Proof. Suppose we are given a labeling where a node p has load l_p which is not the maximum load but its label does not contain a state r represented in the label of one of its children. Let P be the set of all nodes on the path from p to its first ancestor with load at least $l_p + 1$ or up to the root if no such ancestor exists. Add state r to the label of p. Choose some state that is in node p that is not in its parent (one always exists if the parent has load at most l_p) and add it to the label of its parent, and so on so that each node in set P has its load increased by 1. This increase in labeling costs at most $cost_m$ on the label from the highest node in P to its parent (nothing if the highest node in P is the root), and it saves $cost_m$ on the edge (p, c). Thus we have not increased the cost of the tree. Starting this process with internal nodes lowest in the tree gives us the above claim. □

Therefore we can assume that every internal node is labeled with all states in its subtree or has maximum load. In particular, we can assume that the root has maximum load $m - n$. Since each triple state x_q is represented in the leaves of

all three children and the dummy states in only one child each, then the minimum possible cost associated with dropping states at the root is $(3mn+6n-3n^2-3m)cost_m$ (all the dummy states must arise somewhere in the tree and the n triple states not present in the root label must arise three times in the tree). Therefore, to meet the parsimony bounds, there can be no additional cost throughout the remainder of the tree. Considering a single node a_i, there are $m-n+1$ states in the subtree rooted at a_i. Since it can have load at most $m-n$, one of these states must be missing from the label at a_i. If one of the dummy states is dropped, then there will be an extra cost of $cost_m$ beyond what is forced by the root (the root allowed each dummy state to arise once and each dummy state appears in the subtrees rooted in both children of node a_i). Therefore node a_i must be labeled with all its dummy states and all but one of the $x_q \in X(a_i)$. These triple states must be passed up the A tree to the root, but there is sufficient capacity to do so (the states that are dropped at internal nodes of the A tree of dummy states, which ultimately had to be dropped anyway). To achieve the parsimony bound, this x_q missing from the label of node a_i must be one of the triples not represented at the root. Therefore the n triple states missing from the root label correspond to the perfect matching in the 3DM instance. □

We now consider algorithms for fixed load l. Since the topology is given, characters can be solved independently. We first give the algorithm for the most general possible cost function and then consider special cases which can be solved more efficiently. All the algorithms are standard bottom-up dynamic programming. A final pass downward from the root produces an optimal labeling of the tree in time $O(nlk)$.

THEOREM 6.4. *Given a tree on n species with k characters where r is the maximum number of states for any character,*

1. *there exists an $O(nkr^{2l})$-time algorithm to compute the most parsimonious load-l labeling for the tree for arbitrary state-dependent costs;*
2. *there exists an $O(nkl(2r)^l)$-time algorithm to compute the most parsimonious load-l labeling for the tree for arbitrary fixed costs $cost_\ell \le cost_m \le cost_d$;*
3. *there exists an $O(nk(2r)^l)$-time algorithm to compute the most parsimonious load-l labeling for the given tree when $cost_\ell = 0$.*

Proof. When the cost function is state-dependent, we convert our input to a weighted monomorphic parsimony problem. We define a new set of $O(r^l)$ states, one for each possible label of a node. Given two labels l_p and l_c, we can determine the cost of a parent-child edge with labels l_p and l_c. We must match states for mutations and duplications. We thus compute a matrix of edge costs. Because loss and duplication costs are not the same, this matrix is not symmetric in general. We then use the algorithm of Sankoff and Cedergren [34] for weighted parsimony which runs in time $O(nkj^2)$ for n species, k characters, and j states/characters. In our case we have r^l states, where r was the original number of states in the polymorphic character. Thus this algorithm has time $O(nkr^{2l})$.

The bottom-up dynamic programming algorithm for weighted parsimony proceeds as follows. For an internal node v, let $c(v, l_v)$ be cost of the best labeling of the subtree rooted at v provided that node v is labeled l_v. Then we have $c(v, l_v) = \sum_{v' \text{child of } v}(\min_{l_{v'}} c(v', l_{v'}) + w(l_v, l_{v'}))$, where $w(l_v, l_{v'})$ is the cost of the edge with parent label l_v and child label $l_{v'}$. Thus we consider every possible label for an internal node and compare it against every possible label for its children. For arbitrary weight function w, this will cost r^{2l} for each parent-child interaction.

For the case of arbitrary $cost_\ell \le cost_m \le cost_d$ (not state-dependent), we can reduce the overall time to $O(nkl(2r)^l)$. Again, we wish to consider every possible label

for node v but we need not consider every possible label for its children. Suppose that for each child we know the best choice of label for each of load $1, 2, \ldots, l$, where some specific subset (possibly empty) of the label is specified. For example, we know the best load-3 labeling of the child where a and b are two of the three states. This is $O(lr^l)$ information. To find the best labeling of the subtree rooted at v provided v is labeled by l_v, the only labels we need to consider for the children of v are the best ones for each possible subset of l_v and each possible load. For example, if $l_v = \{a, b\}$, $l = 3$, and $*$ can be any state, then the only labels that must be considered for a child are $*$ (best tree with load-1 label), $**$, $***$, a, $a*$, $a**$, b, $b*$, $b**$, ab, and $ab*$. More formally, let $c(v, L, x)$ be the cost of the best subtree rooted at v where the label of v contains state set L and x other states. Then the cost of label l_v and node v is

$$c(v, l_v) = \sum_{\text{children } v'} \min_{L \subseteq l_v} \min_{0 \le l' \le l - |L|} (c(v', L, l') + w(l_v, L, l')),$$

where

$$w(l_v, L, l') = \begin{cases} l' cost_m + (|l_v| - |L| - l') cost_\ell & \text{if } |L| + l' \le |l_v| \\ (|l_v| - L) cost_m + (|L| + l' - |l_v|) cost_d & \text{otherwise.} \end{cases}$$

Thus to compute the cost of a label, each parent must check $O(l2^l)$ labels in each child. Once the label l_v is computed, it contributes to $O(2^l)$ minimizations used by its parent (each subset of l_v with load $|l_v|$). Since each of the $O(n)$ edges is checked $O(l2^l)$ times for each of the r^l possible parent labels, the overall cost is $O(nkl(2r)^l)$.

To prove the final part of the theorem, when $cost_\ell = 0$ (for example when we wish to maximize convexity), we note that whenever we have $cost_\ell = 0$ there exists an optimal solution where each internal node contains all the states in the subtree rooted at it or has maximum load. We begin by locating the highest internal nodes v with at most l states in the subtree rooted at them. We label node v by these states and make it a leaf by removing all its children. Now we can assume all internal nodes have load l. This saves a factor of l using the preceding algorithm since there is now only one value of l'. □

7. Discussion. In this paper we introduced an algorithmic study of the problem of inferring the evolutionary tree in the presence of polymorphic data. We considered parsimony analysis for polymorphic data on fixed topologies and presented algorithms as well as hardness results. We also presented algorithms for inferring perfect phylogenies from such data, and we note that it is reasonable to seek perfect phylogenies for certain types of data. The results of our analysis of the an expanded Indo-European data set studied by Warnow, Ringe, and Taylor has led to a new hypothesis for the evolution of Indo-European languages.

REFERENCES

[1] R. Agarwala and D. Fernández-Baca, *A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed*, SIAM J. Comput., 23 (1994), pp. 1216–1224.

[2] R. Agarwala and D. Fernández-Baca, *Fast and Simple Algorithms for Perfect Phylogeny and Triangulating Colored Graphs*, Technical report TR94-51, DIMACS, to appear in special issue on algorithmic aspects of computational biology, Internat. J. Found. Comput. Sci.

[3] S. Arnborg, D. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-Tree*, SIAM J. Alg. Discrete Methods, 8 (1987), pp. 277–284.
[4] H. Bodlaender, M. Fellows, and T. Warnow, *Two strikes against perfect phylogeny*, in Proc. 19th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1992, pp. 273–283.
[5] H. Bodlaender and T. Kloks, *A simple linear time algorithm for triangulating three-colored graphs*, in Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, 1992, pp. 415–423, to appear in J. Algorithms.
[6] M. Bonet, C. Phillips, T. Warnow, and Shibu Yooseph, *Constructing evolutionary trees in the presence of polymorphic characters*, in Proc. Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, 1996, pp. 220–229.
[7] P. Buneman, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205–212.
[8] L. L. Cavalli-Sforza, P. Menozzi, and A. Piazza, *The History and Geography of Human Genes*, Princeton University Press, Princeton, NJ, 1994.
[9] W. H. E. Day, *Computationally difficult parsimony problems in phylogenetic systematics,* J. Theoret. Biol., 103 (1983), pp. 429–438.
[10] W. H. E. Day, D. S. Johnson, and D. Sankoff, *The computational complexity of inferring phylogenies by parsimony,* Math. Biosci., 81 (1986), pp. 33–42.
[11] W. H. E. Day and D. Sankoff, *Computational complexity of inferring phylogenies by compatibility*, Systematic Zool., 35 (1986), pp. 224–229.
[12] A. Dress and M. Steel, *Convex tree realizations of partitions*, Appl. Math. Lett., 5 (1992), pp. 3–6.
[13] G. F. Estabrook, *Cladistic methodology: A discussion of the theoretical basis for the induction of evolutionary history*, Annu. Rev. Ecol. Syst., 3 (1972), pp. 427–456.
[14] G. F. Estabrook, C. S. Johnson, Jr., and F. R. McMorris, *An idealized concept of the true cladistic character*, Math. Biosci., 23 (1975), pp. 263–272.
[15] J. Felsenstein, *Alternative methods of phylogenetic inference and their interrelationships,* Systematic Zool., 28 (1979), pp. 49–62.
[16] W. Fitch, *Towards defining the course of evolution: Minimum change for a specified tree topology*, Systematic Zool., 20 (1971), pp. 406–416.
[17] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[18] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.
[19] R. M. Idury and A. A. Schäffer, *Triangulating three-colored graphs in linear time and linear space*, SIAM J. Discrete Math., 6 (1993), pp. 289–293.
[20] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[21] S. Kannan and T. Warnow, *Inferring evolutionary history from DNA sequences*, SIAM J. Comput., 23 (1994), pp. 713–737.
[22] S. Kannan and T. Warnow, *Triangulating 3-colored graphs*, SIAM J. Discrete Math., 5 (1992), pp. 249–258.
[23] S. Kannan and T. Warnow, *A fast algorithm for the computation and enumeration of perfect phylogenies*, SIAM J. Comput., 26 (1997), pp. 1749–1763.
[24] W. J. Le Quesne, *A method of selection of characters in numerical taxonomy*, Systematic Zool., 18 (1969), pp. 201–205.
[25] W. J. Le Quesne, *Further studies based on the uniquely derived character concept*, Systematic Zool., 21 (1972), pp. 281–288.
[26] M. F. Mickevich and C. Mitter, *Treating polymorphic characters in systematics: A phylogenetic treatment of electrophoretic data*, in Advances in Cladistics, V. A. Funk and D. R. Brooks, eds., New York Botanical Garden, New York, 1981, pp. 45–58.
[27] F. R. McMorris, T. Warnow, and T. Wimer, *Triangulating vertex-colored graphs*, SIAM J. Discrete Math., 7 (1994), pp. 296–306.
[28] M. Nei, *Molecular Evolutionary Genetics*, Columbia University Press, New York, 1987.
[29] A. Proskurowski, *Separating subgraphs in k-trees: Cables and caterpillars*, Discrete Math., 49 (1984), pp. 275–285.
[30] D. Ringe, *personal communication*, 1995.
[31] D. J. Rose, *On simple characterization of k-trees*, Discrete Math., 7 (1974), pp. 317–322.
[32] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs,* SIAM J. Comput., 5 (1976), pp. 266–283.
[33] A. K. Roychoudhury and M. Nei, *Human Polymorphic Genes: World Distribution*, Oxford University Press, London, UK, 1988.

[34] D. Sankoff and R. J. Cedergren, *Simultaneous comparison of three or more sequences related by a tree*, in Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, D. Sankoff and J.B. Kruskal, eds., Addison–Wesley, Reading MA, 1983, pp. 253–263

[35] D. Sankoff and P. Rousseau, *Locating the vertices of a Steiner tree in arbitrary space,* Math. Programming, 9 (1975), pp. 240–246.

[36] M. A. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.

[37] T. Warnow, *Constructing phylogenetic trees efficiently using compatibility criteria,* New Zealand J. Botany, 31 (1993), pp. 239–248.

[38] T. Warnow, *Mathematical approaches to comparative linguistics,* in Proceedings of the National Academy of Sciences, vol. 94, 1997, pp. 6585–6590.

[39] T. Warnow, D. Ringe, and A. Taylor, *A character based method for reconstructing evolutionary history for natural languages*, Technical report, Institute for Research in Cognitive Science, 1995, and Proc. ACM/SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996.

[40] J. J. Weins, *Polymorphic characters in phylogenetic systematics*, Systematic Biology, 44 (1995), pp. 482–500.