# CONSTRUCTING EVOLUTIONARY TREES IN THE PRESENCE OF POLYMORPHIC CHARACTERS

MARIA BONET* CYNTHIA PHILLIPS† TANDY J. WARNOW‡ SHIBU YOOSEPH§

**Abstract.** Most phylogenetics literature and construction methods based upon characters presume monomorphism (one state per character per species), and 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, which was presented by invitation at the National Academy of Sciences in November 1995.

## 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 $c : S \rightarrow Z$, where $Z$ denotes the integers and thus represents the set of possible *states* of $c$. 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 which is permitted to have more than one state on a given object will be called a *polymorphic character*, and one which 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. 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.*[1] Polymorphism also arises in the comparison of different languages. The Indo-Europeanist Donald Ringe writes: *In choosing lexical characters we try to work with 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-labelled tree topologies for polymorphic data have

* Department of Computer Science, Universidad Politecnica de Cataluña. bonet@goliat.upc.es. Research partly supported by NSF grant number CCR-9403447

† Sandia National Labs, Albuquerque, NM, USA. caphill@cs.sandia.gov. This work was performed under U.S. Department of Energy contract number DE-AC04-76AL85000

‡ Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA. tandy@central.cis.upenn.edu. Research partly supported by an NSF National Young Investigator Award under contract ccr-9457800 and from an NSF grant in Linguistics .

§ Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA. yooseph@gradient.cis.upenn.edu. Research partly supported by a Fellowship from the Institute for Research in Cognitive Science at the University of Pennsylvania and also by a Fellowship from the Program in Mathematics and Molecular Biology at the University of California at Berkeley, which is supported by the NSF under grant no. DMS-9406348

[1] From [28], page 254.

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 major contribution of this paper is a methodology for inferring perfect phylogenies from monomorphic and polymorphic characters. Recent work in Historical Linguistics [38] 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 [38] 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.

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 [38]. 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*, where by this we mean that the nodes sharing any state of any character form a connected set in the tree.

DEFINITION 1. *Given a set $S$ of taxa defined by a set $C$ of characters ($|C| = k$), where each $c_j \in C$ is a function $c_j : S \to (2^Z - \{\emptyset\})$, let $T$ be a tree which is leaf-labelled by the taxa in $S$ and with each internal node $v$ labelled with a vector from $(2^Z - \{\emptyset\})^k$ such that the value of $c_j(v)$ is given by the $j^{th}$ component of this vector. A character (polymorphic or monomorphic) $c$ is* **convex** *on $T$ if for all $i \in Z$, the set $X_{c,i} = \{v \in V(T) : i \in c(v)\}$ 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. *A polymorphic character $c$ with $r$ states is* separated *into characters $\alpha_1, \ldots, \alpha_l$ by a function $f : \{1, \ldots, r\} \to \{1, \ldots, l\}$ where $\alpha_j^{-1}(i) = c^{-1}(i)$ if $f(i) = j$. Undetermined values* *of $\alpha_1, \ldots, \alpha_l$ are arbitrary. In particular* singletons *maintain the spirit of character $c$. That is, if $f(i) \neq j$ for any $i \in c(s)$ (species $s$ does not contain any state mapped to character $j$), then let $\alpha_j^{-1}(\alpha_j(s)) = \{s\}$ ($s$ has a unique state for $\alpha_j$).*

*Problem 1: Separation into $l$ convex characters.*

**Input:** Set $S$ of taxa defined by set $C$ of characters.

**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 $\alpha_i$ (hence the use of singletons).

In Biology, polymorphic characters can arise when dealing with allozyme data [26] and morphological data [39]. 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 pre-specified bound of states for each character. This bound may be character dependent.

Definition 3. *A tree $T$ which has polymorphic characters is said to have* **load** *$l$ if for every character $c \in C$ and every $v \in V(T)$, $|c(v)| \leq l$.*

*Problem 2: $l$-load perfect phylogeny.*

**Input:** Set $S$ of taxa defined by set $C$ of (possibly) polymorphic characters.

**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 2.1. *Given a set of taxa defined by a set $C$ of polymorphic characters, $T$ is an $l$-load perfect phylogeny for $C$ if and only if 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 $\alpha$ has $r$ states present on $S$. Let $T_i$ be the subgraph of $T$ induced by the vertices labelled $i$ by $\alpha$. Since $T$ is a perfect phylogeny, each $T_i$ is a subtree. Define $G_\alpha$ to be the graph whose vertices are in one-to-one correspondence with the subtrees $T_i, i = 1, 2, \ldots, r$, and where $(T_i, T_j) \in E$ if and only if $T_i \cap T_j \neq \emptyset$. Note that since $T$ has load $l$, $G_\alpha$ has max clique size at most $l$. $G_\alpha$ is triangulated since it is the intersection graph of subtrees of a tree [7], and hence $G_\alpha$ is perfect [17]. Since $G_\alpha$ is perfect, the chromatic number of $G_\alpha$ equals the max clique size, and hence is bounded by $l$. Hence we can partition the nodes of $G_\alpha$ 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. $\square$

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 4.3.

**3.1. Min Load Problems.** When convexity of the monomorphic constituents of the polymorphic characters is a reasonable request, we may seek a tree with a pre-specified load bound, or else we may seek a tree with a minimum possible load bound. We call the latter problem the Minimum Load Problem.

We note that the Minimum Load Problem is NP-hard, since the question of whether a 1-load Perfect Phylogeny exists is NP-Complete [4, 36]. The 2-load Perfect Phylogeny Problem is the next question to consider. The various parameters to the problem are $n$, the number of species; $k$, the number of (polymorphic) characters; and $r$, the maximum number of states per character.

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) & (v) :** When $n$ is fixed, the number of possible leaf-labelled topologies is bounded, so we need only consider the Min Load problem on a fixed topology. Determining the minimum load on a fixed leaf-labelled 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) & (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. 2 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.

*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 $E(G) \subseteq E(G')$ 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 $\alpha$, 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 4.4 and 4.5.

**Parts (iv) & (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:

*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 load problem is constructed as follows. Replace each $*$ entry in the matrix defined by $I$, by 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. Initialize $S' = S$ and $C' = C$. Now, for each $\alpha \in C$, define two new characters $\alpha^1$ and $\alpha^2$, and nine new species $s^1_\alpha, \ldots, s^9_\alpha$ as follows
For each $\beta \in C'$ (where $\beta \neq \alpha$), set $\beta(s^i_\alpha) = 2$ , where $1 \leq i \leq 9$.
For each $s \in S'$, set $\alpha^1(s) = 2$ and $\alpha^2(s) = 2$.
Also set $s^1_\alpha = (0, 0, 2)$,$s^2_\alpha = (0, 1, 2)$, $s^3_\alpha = (0, 2, 2)$, $s^4_\alpha = (2, 0, 0)$, $s^5_\alpha = (2, 1, 1)$, $s^6_\alpha = (2, 2, 2)$, $s^7_\alpha = (1, 2, 0)$, $s^8_\alpha = (1, 2, 1)$, $s^9_\alpha = (1, 2, 2)$.
*(Notation : $s^i_\alpha = (x, y, z)$ indicates that $\alpha(s^i_\alpha) = x, \alpha^1(s^i_\alpha) = y, \alpha^2(s^i_\alpha) = z$).*
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 $S$. We will define $\alpha.i = \{s | \alpha(s) = i\}$. For each $\alpha \in C$, note that there is an edge in $T$ which partitions $\alpha.0$ from $\alpha.1$. 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 relabelling 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 characters set $C$ to $C'$ by adding the new characters, which consists 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. For each character $\alpha \in C$, identify the edge $e$ which partitions $\alpha.0$ from $\alpha.1$. Attach the 9 new species, associated with $\alpha$, as shown in Figure 1.

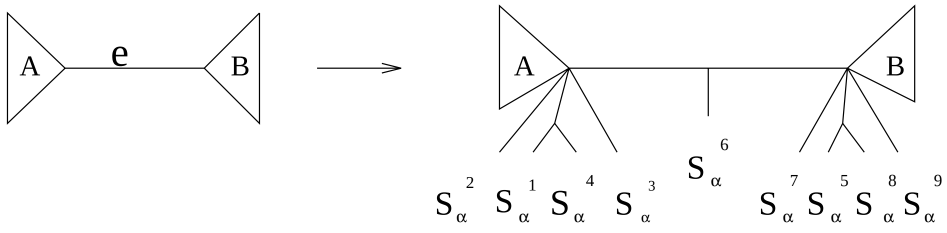

FIG. 1. *Adding the* 9 *new species associated with* $\alpha$

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 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 $\alpha$ has associated with it an edge, which splits $\alpha.0$ from $\alpha.1$, or, $\alpha.0$ from $\alpha.2$, or, $\alpha.1$ from $\alpha.2$. Observe that, in $I'$, for each $\alpha \in C$, the only partition possible is $\alpha.0$ from $\alpha.1$. This follows as a result of the constraints imposed by $\alpha^1$ and $\alpha^2$. 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 $C$, 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.
$\square$

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 runtimes 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 labelled distinctly from the species in $S$, and instead 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 labelled by $s$.

We need some preliminary definitions and facts.

Definition 4. *The* **Extended Hamming distance** *of* $e = (x, y)$ *is* $\sum_{c \in C} |c(x) \triangle c(y)|$, *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 5. *(See [23]) Given* $x \in S^*$*, the equivalence relation* $E_x$ *is the transitive closure of the following relation* $E'_x$ *on* $S - \{x\}$*:* $aE'_xb$ *if there exists character* $c$ *such that* $(c(a) \cap c(b)) - c(x) \neq \emptyset$*. We denote this set of equivalence classes by* $(S - \{x\})/x$*.*

Some facts follow from this definition. Let $x$ be an internal node of a perfect phylogeny $T$.

**Fact 1:** Two species in $S$ which are in the same equivalence class of $(S - \{x\})/x$ must be in the same component of $T - \{x\}$.

**Fact 2:** If a perfect phylogeny exists for $S \cup \{x\}$, then there is a perfect phylogeny $T$ in which the components of $T - \{x\}$ have leaf sets which are the components of $(S - \{x\})/x$.

Fact 2 does not necessarily hold simultaneously for all internal nodes of a perfect phylogeny $T$. Instead the following fact is true for every internal node of $T$.

**Fact 3:** 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 $v$ as stated. Let $H_1, \ldots, H_t$ be equivalence classes of $(S - \{v\})/v$ containing species from $G$. Clearly, to prove Fact 3 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 $c \in C$ and a state $a$ of $c$ such that $a \notin c(v)$, yet $a \in c(x) \cap c(z)$ 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 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 $c$ such that $(c(z_1) \cap c(z_2)) - c(v) \neq \emptyset$ (this follows from $(S - \{v\})/v$ being the transitive closure of $E_v$). Let $a \in c(z_1) \cap c(z_2) - c(v)$. Since $z_1$ and $z_2$ are in different equivalence classes of $(S - \{x\})/x$, $a \in c(x)$. Now let $z = lca_T(z_1, z_2)$. This node $z$ is in the subtree rooted at $v$, and satisfies $a \in c(z)$, because $T$ is a perfect phylogeny and $z$ is on the path between $z_1$ and $z_2$. This is the character $c$ 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 $\sum_{c \in C} |c(x_1) \triangle c(x_2)| = 1$. 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 6. *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 $T_i$ labelled 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.

*The 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 $\mathbf{count}_i$ 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 labelling 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 unlabelled for now. A method for labelling these nodes is given in the proof of Theorem 4.2. 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 only compute one topology per "true" subproblem.

LEMMA 4.1. *If there exists a perfect phylogeny for $S \cup \{x\}$, then the algorithm PHYLOGENY 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 $y = lca_T(G)$. By Fact 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 $x$, that $[G, z]$ 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. $\square$

THEOREM 4.2. *The algorithm PHYLOGENY(S) runs in time $O(r^{lk+1}lkn)$, and returns "yes" if and only if $S$ 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 4.1, the algorithm will return "yes". Suppose now that the algorithm returns the answer "yes", and suppose the leaf-labelled tree produced is $D$. We now show that the internal nodes of this tree can be labelled so as to create a perfect phylogeny $T$ with load $l$. Given a character $\alpha$ and an unlabelled 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 $\alpha$, 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 labelled with at most $l$ $\alpha$-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 labelling has load bounded by $l$.

The search graph can be constructed in time $O(nlkr^{lk+1})$ (see [2]). The rest of the algorithm will run in linear time in the size of the graph $(O(nlkr^{lk+1}))$. $\square$

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

### 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. Based upon 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 if and only if $\forall (v, u) \in E$, $color(v) \neq color(u)$).

The **neighbour 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 $\alpha \neq \beta, color(\alpha_i) = color(\alpha_j) \neq color(\beta_s)$. 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 $I$ if and only if 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 4.3. *Let $G$ be a triangulated graph. 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 4.4. *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 $\varphi : V(G) \rightarrow \{subtrees\ of\ T\}$ and $\phi : V(T) \overset{\text{bijection}}{\rightarrow} \{maximal\ cliques\ of\ G\}$ 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 $\varphi$ and $\phi$. We will first show that this, together with conditions 1 and 2, imply that $G$ is triangulated. Let $\Lambda = a_1a_2 \ldots a_ia_1, i \geq 4$ be a simple cycle in $G$. We will show that $\Lambda$ 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 $\Lambda$ 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 4.3, 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 $\varphi'$ and $\phi'$ 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 $T$, $\phi$ and $\varphi$ from $T'$, $\phi'$ and $\varphi'$ 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 $T$, $\phi$ and $\varphi$ satisfy the stated conditions. □

THEOREM 4.5. *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 4.4, 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 4.4. 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 ∪ 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 4.6. *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 4.3, 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 neighbour 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 $\sigma$ be a perfect elimination scheme for $G''$. Look at $x$ which is the first vertex in $\Gamma_{G'}(v)$ to appear in $\sigma$. There are two cases to handle here. **Case 1** is when $x$ is within the sequence of last $kl$ vertices appearing in $\sigma$. In this case make $v$ adjacent to all vertices in the last $kl$ positions of $\sigma$, 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 $\sigma$. Let $A$ be the set of vertices following $x$ which are neighbours 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 4.7. *Let $G$ be a $(k,l)$-partition intersection graph. Then $G$ can be $l$-triangulated if and only if 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 4.6, 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 7. *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. As our input graphs may not be $(k,l)$-partition intersection graphs, we need the following result:

Lemma 4.8. *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 if and only if $G'$ can be $l$-triangulated, and*

- *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 super-graph 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.

**4.2.3.1. 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 et. al [32] proved the following lemma about triangulated graphs.

Lemma 4.9. *Let $G$ be a triangulated graph, $\sigma$ a perfect elimination scheme for $G$, and let $a, b$ be 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 $\sigma$, then $(a, b)$ is an edge in $G$.*

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

Lemma 4.10. *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 4.11. *Let $G = (V, E)$ be a $(k, l)$-partition intersection graph containing at least $kl + 1$ vertices, $S_0$ pb-set of $G$, and let $C$ be a component of $G - S_0$. Then $C \cup cl(S_0)$ can be $l$-triangulated if and only if 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 $S = S_0 \cup \{v\} - \{x\}$, $C'$ is both a component of $G - S$ and of $C \cup cl(S_0) - S$,*
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 only intersect on $S_0 \cup \{v\}$. Thus, we need only show the converse.

So 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 4.10, 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 $V_K = \{v : L(v) = K \textit{ or } v \in K\}$, 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 unlabelled vertices (these will be in $S'$) and at least one labelled 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, 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 unlabelled, and thus $X = C$, or that the labelled vertices were all labelled $X$. Since we have assumed $v_k$ was labelled, 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 $V_K = \{v : L(v) = K \textit{ or } v \in K\}$) is an $l$-triangulated $(kl-1)$-tree, and that $T_K$ is based upon the set $K$.

By our construction of the labelling 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 $\sigma'$ be a perfect elimination scheme for $T_K \cup \{x\}$. Clearly, we can assume that $x$ is the last vertex in $\sigma'$ 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 4.9. 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 labelled the vertex $z$ we labelled 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 4.12. *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 if and only if 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 a $M_x \in \mathcal{M}$ 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 4.11, 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 4.11, 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. □

We can now conclude with our final theorem.

THEOREM 4.13. *Let $G = (V, E)$ be a $(k, l)$-partition intersection graph and let $S \subset V$ 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 $G - S'$.*

#### 4.2.3.2. Implementation details of the dynamic programming algorithm (i.e. 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 $M_i^j$ can either 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 4.12 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 4.12
    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 4.12 *)
            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 4.7 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
```

Note that the above algorithm can be easily modified to give back the $l$-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 4.12 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 4.14. *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 4.4.

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

Let $A_p = \Gamma(v_p) \cap \{v_{p+1}, v_{p+2}, .., v_{|V|}\}$. Inductively, assume we are at vertex $v_j$ in $\sigma$ and assume we have the tree $T_{j+1}$. Let $v_i$ be the first vertex following $v_j$ (i.e. $j < i$) in $\sigma$ 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 $G|\{v_j, v_{j+1}, v_{j+2}, ..v_{|V|}\}$ and $G|\{v_i, v_{i+1}, v_{i+2}, ..v_{|V|}\}$ respectively, it follows that $|\{v_i\} \cup A_i| > |A_j|$ iff, in $G|\{v_j, v_{j+1}, v_{j+2}, ..v_{|V|}\}$, 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 $G|\{v_j, v_{j+1}, v_{j+2}, ..v_{|V|}\}$. Thus, from Theorem 4.4, 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 $G|\{v_j, v_{j+1}, v_{j+2}, ..v_{|V|}\}$ is not a maximal clique. Let $v_q$ $(j \leq q)$ be the first vertex to the left of $v_i$ in $\sigma$ 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 with every vertex $v_r \in \sigma$ $(j < r)$, two variables, one which corresponds to the vertex $u \in V(T_r)$ such that $u$ represents the maximal clique $\{v_r\} \cup A_r$ in $G|\{v_r, v_{r+1}, .., v_{|V|}\}$ and another variable which corresponds to a vertex $v_s$ in $\sigma$ 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 that $T$ can then be constructed in $O(|V| + |E|)$. Hence we have the following theorem.

THEOREM 4.15. *Let $G = (V, E)$ be a vertex colored graph which is $l$-triangulated. Then the tree $T$ which satisfies the conditions in Theorem 4.4 can be constructed in $O(|V| + |E|)$.*

THEOREM 4.16. *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 $l_c$ are given, the algorithm can be modified to run in $O(nL^2 + (rkL^2)^{L+1})$, where $L = \sum_{c \in C} l_c$.

## 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 $L = \sum_{c \in C} l_c$, and $l_c$ is the load bound for the character $c$. 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 which 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 in [23] has running time which is $O(2^{2r+r^2}k_m^{r+3} + Mk_mn)$, 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 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-labelled 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 neighbours of $v$ which are leaves and $\Gamma_2(v)$ consists of all the non-leaf neighbours of $v$. For each $u_j \in \Gamma_2(v)$ add a new node $w_j$ on the edge $(v, u_j)$. Compute the labelling 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 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 $(S_v, C)$. If any $(S_v, C)$ fails to have a perfect phylogeny then RETURN(No), else RETURN(yes).

THEOREM 4.17. *Algorithm D correctly determines whether a perfect phylogeny $T'$ exists refining $T$ within the stated load bounds, and 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 possible leaf-labelled 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, [38] 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 2 shows the tree that they now posit. This was obtained using Algorithm D, which we described earlier. This tree is infact different from their earlier hypothesis (which was presented at the NAS Symposium on the Frontiers of Science in November 1995) and also 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 a significant support for a subgroup of Greek and Armenian.

**6. Polymorphism in Biology.** The evolution of biological polymorphic characters can be modelled 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 consider two types of costs : 1. 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. 2. 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 $cost(\alpha, (u, v)) = cost_\ell[|X| - |Y|] + cost_m|Y|$.

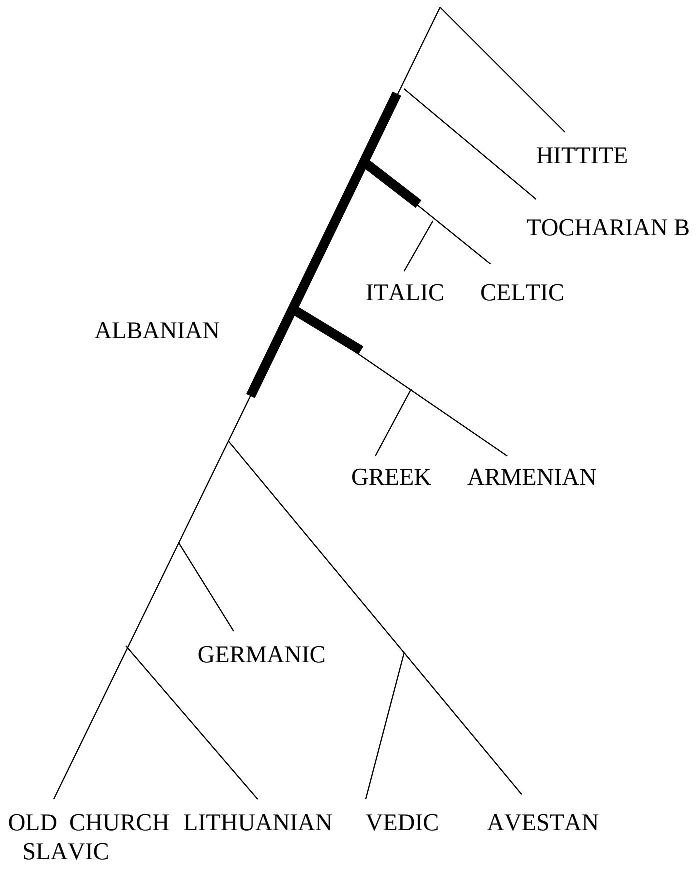

FIG. 2. *The tree on the Indo-European data set. Albanian can be on any of the thick edges. The tree just indicates a rooted topology without any edge lengths.*

- 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-labelled topology and a maximum load, $l$, what is the most parsimonious labelling 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 *3-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 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 $(3mn - 3n^2)cost_m$ if and only if 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 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 $(3mn - 3n^2)cost_m$ if and only if the instance of 3DM has a perfect matching.

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 2 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 $(3mn + 6n - 3n^2 - 3m)cost_m$ if and only if the instance of 3DM has a perfect matching. $\square$

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 $O(nlk)$. We can also randomly sample optimal solutions.

THEOREM 6.2. *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-$l$ la-*

*belling 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 labelling for the tree for arbitrary fixed costs $cost_\ell \leq cost_m \leq cost_d$.*
3. *There exists an $O(nk(2r)^l)$-time algorithm to compute the most parsimonious load-l labelling 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 an 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/character. 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 labelling of the subtree rooted at $v$ provided that node $v$ is labelled $l_v$. 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 $cost_\ell \leq cost_m \leq cost_d$ (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 2 of the 3 states. This is $O(lr^l)$ information. To find the best labelling of the subtree rooted at $v$ provided $v$ is labelled 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 is $*$ (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:

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

where $w(l_v, L, l') =$

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

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 $cost_\ell = 0$, then 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 save 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 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. on Computing, Vol. 23, No. 6, pp. 1216-1224.

[2] R. Agarwala and D. Fernández-Baca, *Fast and Simple Algorithms for Perfect Phylogeny and Triangulating Colored Graphs.* To appear in the special issue on *Algorithmic Aspects of Computational Biology* of International Journal of Foundations of Computer Science. Available as DIMACS technical report TR94-51.

[3] S. Arnborg, D.Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-Tree*, SIAM J. of Algebraic and Discrete Methods, Vol. 8, No. 2, April 1987, pp. 277-284.

[4] H. Bodlaender, M. Fellows, and T. Warnow, *Two strikes against perfect phylogeny*, In Proceedings of the 19th International Colloquium on Automata, Languages, and Programming, Springer Verlag, Lecture Notes in Computer Science (1992), pp. 273–283.

[5] H. Bodlaender and T. Kloks, *A simple linear time algorithm for triangulating three-colored graphs*, In Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (1992), pp. 415–423. To appear, Journal of Algorithms.

[6] M. Bonet, C. Phillips, T. Warnow and Shibu Yooseph, *Constructing evolutionary trees in the presence of polymorphic characters*, In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pages 220-229, Philadelphia, Pennsylvania, 22-24 May 1996.

[7] P. Buneman, *A characterization of rigid circuit graphs*, Discrete Math 9 (1974), pp. 205-212.

[8] L. Luca Cavalli-Sforza, P. Menozzi and A. Piazza, *The History and Geography of Human Genes*, Princeton University Press, 1994.

[9] W. H. E. Day, *Computationally difficult parsimony problems in phylogenetic systematics,* Journal of Theoretical Biology, 103: 429-438, 1983.

[10] W.H.E. Day, D.S. Johnson, and D. Sankoff, *The computational complexity of inferring phylogenies by parsimony,* Mathematical biosciences, 81:33-42, 1986.

[11] W. H. E. Day and D. Sankoff, *Computational complexity of inferring phylogenies by compatibility*, Syst. Zool., Vol. 35, No. 2 (1986), pp. 224–229.

[12] A. Dress and M. Steel, *Convex tree realizations of partitions*, Appl. Math. Letters, Vol. 5, No. 3 (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*, Mathematical Biosciences, 23 (1975), pp. 263–272.

[15] J. Felsenstein, *Alternative methods of phylogenetic inference and their interrelationships,* Systematic Zoology, 28: 49-62, 1979.

[16] W. Fitch, *Towards defining the course of evolution: minimum change for a specified tree topology*, Syst. Zool., 20:406-416 (1971).

[17] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 1980 Academic Press Inc.

[18] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.

[19] R. Idury and A. Schaffer, *Triangulating three-colored graphs in linear time and linear space*, to appear, SIAM J. Discrete Mathematics.

[20] R. Karp, *Reducibility among combinatorial problems*, In R.E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.

[21] S. Kannan and T. Warnow, *Inferring evolutionary history from DNA sequences*, SIAM J. on Computing, Volume 23, No. 3, (1994) pp. 713-737. A preliminary version of this paper appeared in the Proceedings of the Symposium on the Foundations of Computer Science, St. Louis, Missouri, 1990.

[22] S. Kannan and T. Warnow, *Triangulating three-colored graphs*, SIAM J. on Disc. Math., 5 (1992), pp. 249–258; a preliminary version of this appeared in Proc. 2nd Annual ACM/SIAM Symposium on Discrete Algorithms.

[23] S. Kannan and T. Warnow, *A fast algorithm for finding and enumerating perfect phylogenies*, Proc. 6th Annual ACM/SIAM Symposium on Discrete Algorithms, 1995, San Francisco.

[24] W. J. Le Quesne, *A method of selection of characters in numerical taxonomy*, Syst. Zool., 18 (1969), pp. 201–205.

[25] W. J. Le Quesne, *Further studies based on the uniquely derived character concept*, Syst. Zool., 21 (1972), pp. 281–288.

[26] M. F. Mickevich and C. Mitter, *Treating Polymorphic Characters in Systematics : A Phylogenetic Treatment of Electrophoretic Data*, pages 45-58 in Advances in cladistics, Volume I (1981), V.A. Funk and D.R. Brooks eds., New York Botanical Garden, New York.

[27] F. R. McMorris, T. Warnow, and T. Wimer, *Triangulating vertex colored graphs*, SIAM J. on Discrete Mathematics, Vol. 7. No. 2, (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., Vol. 5, No. 2, June 1976.

[33] A.K. Roychoudhury and M. Nei, *Human Polymorphic Genes: World Distribution*, 1988, Oxford University Press.

[34] D. Sankoff and R.J. Cedergren, 1983. *Simultaneous comparison of three or more sequences related by a tree*, pp. 253-263 in "Time Warps, String Edits, and Macromolecules: the theory and practice of sequence comparison" edited by D. Sankoff and J.B. Kruskal, Addison-Wesley, Reading MA.

[35] D. Sankoff and P. Rousseau, 1975, *Locating the vertices of a Steiner tree in arbitrary space,* Mathematical Programming, 9:240-246.

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

[37] T. Warnow, *Constructing phylogenetic trees efficiently using compatibility criteria,* New Zealand Journal of Botany, 1993, Vol. 31: 239-248.

[38] T. Warnow, D. Ringe and A. Taylor, *A character based method for reconstructing evolutionary history for natural languages*, Tech Report, Institute for Research in Cognitive Science, 1995, and *Proceedings 1996 ACM/SIAM Symposium on Discrete Algorithms.*

[39] J. J. Weins, *Polymorphic Characters in Phylogenetic Systematics*, Systematic Biology 44(4):482-500, 1995.