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.

STOC'96, Philadelphia PA,USA
0-89791-785-5/96/05

*Department of Mathematics, University of Pennsylvania. bonet@math.upenn.edu. 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, SBR95-12092. Additional support is provided by Paul Angello, Esq.

§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

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.[28]*

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 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 [36] 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 [36] 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 3.3, we present a methodology for inferring perfect phylogenies from

[1] From [26], page 254.

data which combine monomorphic and polymorphic data. In Section 4 we present our analysis of the Indo-European data studied by Warnow, Ringe, and Taylor [36]. In Section 5, we consider the problem of inferring evolutionary trees from polymorphic data when a perfect phylogeny is an unlikely outcome. We conclude in Section 6.

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 $c_j : S \rightarrow (2^Z - \{\emptyset\})$, 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 $f : \{1, \ldots, r\} \rightarrow \{1, \ldots, l\}$ 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 α_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 α_i (hence the use of singletons).

In Biology, (a) alternative encodings of the same amino acid sequence, (b) alleles for a specific gene, and (c) jumping genes (transposons) can each cause polymorphic data. 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 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 α has r states present on S. Let T_i be the subgraph of T induced by the vertices labelled i by α. Since T is a perfect phylogeny, each T_i is a subtree. Define G_α to be the graph whose vertices are in one-to-one correspondence with the subtrees $T_i, i = 1, 2, \ldots, r$, and where $(T_i, T_j) \in E$ if and only if $T_i \cap T_j \neq \emptyset$. Note that since T has load l, G_α has max clique size at most l. G_α is triangulated since it is the intersection graph of subtrees of a tree [6], and hence G_α is perfect [16]. Since G_α is perfect, the chromatic number of G_α equals the max clique size, and hence is bounded by l. Hence we can partition the nodes of G_α into at most l independent sets, $V_1, V_2, \ldots, V_l$. Each V_i thus defines a monomorphic character (filled in with singletons), and hence T is a perfect phylogeny for each of these monomorphic characters. ∎

Polymorphism in characters that are based upon columns of molecular sequences behaves differently than polymorphism in morphological characters; for these characters, variations on the parsimony criterion are more appropriate optimization criteria. We discuss the computational complexity of these problems in Section 5.

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 3.2. 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 3.3.

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, 34]. 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 2 *The Min Load Problem can be solved in polynomial time for all fixed n and when $r = 2$, but is NP-hard for all fixed $r \geq 3$ and for all fixed k. Determining whether a 2-load perfect phylogeny exists is solvable in polynomial time if n or k is fixed, or if $r = 2$, but* NP-Complete *for fixed $r \geq 3$.*

Proof: 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.

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 3.2.2 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 α, with $\alpha(s_e) = \{i, j\}$, where $e = (i, j)$. Also, set $l = t + 1$. We claim that the instance to the partial t-tree problem has a solution iff the corresponding instance to the load problem has a solution. This can be seen by observing that G is the partition intersection graph of the instance of the load problem and thus we can use Theorems 4 and 6.

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 [34].

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 α^1 and α^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. It can be shown that I has a solution iff I' has a solution.

If k is fixed then the 2-load perfect phylogeny problem can be solved in polynomial time using the algorithms from Section 3.2.

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 [17] and that there is always an r-load perfect phylogeny on any input set containing characters with at most r states. ∎

This theorem shows that *any* polynomial time algorithm *requires* both k and l bounded.

3.2 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. Because of space

constraints, some of the proofs have been omitted. They will appear in the full version of the paper.

3.2.1 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.

Note that the number of possible labels for nodes is r^{lk}. This follows from the observation that each character has only r states, and each node can choose at most l of these. Let us call this set of possible node labellings S^*, and note that $S \subseteq S^*$ (since otherwise some node in S has load greater than l). We need some preliminary definitions and facts.

Definition 4 *The Hamming distance of $e = (x, y)$ is $\sum_{c \in C} |c(x) \triangle c(y)|$, where $\triangle$ is symmetric difference.*

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 [22]) 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.

Fact 1: Two species in S which are in the same equivalence class of $(S - \{x\})/x$ must be in the same component of $T - \{x\}$, for any perfect phylogeny for S that contains x.

Fact 2: If a perfect phylogeny exists for $S \cup \{x\}$, then there is a perfect phylogeny T in which the subtrees of x in T 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: Let T be a perfect phylogeny for a set of species S. Let x be an internal node of T and G an equivalence class of $(S - \{x\})/x$. Let y be an internal node of T such that the subtree rooted at y, when we think of T as having root x, contains G. Then there exist $H_1, \ldots, H_t$ in $(S - \{y\})/y$ such that $H_1 \cup \cdots \cup H_t = G$.

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) $\cup_t H_t = G$ 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 subtree T_i labelled by H_t. 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_t and set a counter $\mathbf{count}_t$ equal to b_t. 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 $[H_t, y]$ has been determined to be "true," and hence a topology $T[H_t, y]$ 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 ?? (omitted in this abstract). 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 1 *If there exists a perfect phylogeny for $S \cup \{x\}$, then the algorithm PHYLOGENY assigns true to $[G, x]$, for each $G \in (S - \{x\})/x$.*

Theorem 3 *The algorithm PHYLOGENY(S) runs in time $O(r^{lk+1} lkn)$, and returns "yes" if and only if S has a perfect phylogeny.*

Most of the details of this proof are similar to those provided in [2], except that they omit the question of how to label the nodes of the tree. (The initial labelling they provide can be shown to be faulty in some cases.)

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

3.2.2 A Graph-Theoretic Algorithm for Fixed k and l

The algorithm we present is based upon a characterization of intersection graphs of subtrees of l-load perfect phylogenies.

Preliminary Definitions Let $G = (V, E)$ be a graph. A **vertex coloring** of G is a function $color : V \rightarrow Z$. We do not require that *color* be a *proper* coloring (a coloring function is proper if and only if $\forall (v, u) \in E$, $color(v) \neq color(u)$). Given a graph $G = (V, E)$ and a vertex coloring $color : V \rightarrow Z$, a **monochromatic clique** in G is a clique $V_0 \subset V$ 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 chord-free 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. Let $G = (V, E)$ be a graph with a vertex coloring. 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. Let $\alpha \in C$ be a fixed character, and let $i \in Z$. We define $\alpha_i = \{s \in S : i \in \alpha(s)\}$. The **Partition-Intersection Graph** of I is the vertex colored graph $G_I = (V, E)$ in which $V = \{\alpha_i : \alpha \in C\}, E = \{(\alpha_i, \beta_j) : \alpha_i \cap \beta_j \neq \emptyset$, where $i \neq j$ if $\alpha = \beta\}$. The vertex coloring *color* is defined as follows: for $\alpha \neq \beta, color(\alpha_i) = color(\alpha_j) \neq color(\beta_s)$. Thus we have a color for each character; all nodes associated with a character receive the same color. Note that because the input I can have load greater than one, the coloring function *color* may not be proper.

The main results in this section 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 k-colored graph G (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.
There is a characterization of triangulated graphs as intersection graphs [6].

Theorem 4 *[6] A graph $G = (V, E)$ is triangulated if and only if it is the intersection graph of subtrees of a tree.*

We now look at an extension of this particular characterization for l-triangulated graphs.

Theorem 5 *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
(a) $(v, w) \in E(G)$ iff $\varphi(v) \cap \varphi(w) \neq \emptyset$.
(b) $\varphi(v) = \{u \in V(T) : v \in \phi(u)\}$.
(c) $\forall v \in V(T)$, $\phi(v)$ has at most l vertices of the same color.

Theorem 6 *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.*

Further Definitions 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 $\mathbf{C \cup cl(S)}$ to be the graph formed by adding to the subgraph of G induced by $C \cup S$ sufficient edges to make S into a clique. Let $G = (V, E)$ be a k-colored graph. We say that G is a **(k,l)-partition intersection graph** if (a) the maximum monochromatic clique size is l, and (b) G is edge covered by kl-cliques.

We will need the following lemma in our dynamic programming algorithm.

Lemma 2 *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.*

We are thus motivated to make the following definition:

Definition 7 *Let $G = (V, E)$ be a k-colored graph with maximum monochromatic clique size no greater than 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.**

We will present a dynamic programming algorithm which 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 3 *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: (a) 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, (b) G can be l-triangulated if and only if G' can be l-triangulated, and (c) The number of vertices in G' is $n + m(kl - 2)$, where $n = |V|, m = |E|$.*

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

Algorithm B
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. 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.

We state the following theorem:

Theorem 7 *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:*
(a) 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.
(b) 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.
(c) Every edge in C is in exactly one C_x given above.

Proof:[Sketch] Suppose that $C \cup cl(S_0)$ can be l-triangulated, and let G' be an l-triangulation of $C \cup cl(S_0)$ It can be shown 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. It can then be shown 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. These graphs 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 $C \cup cl(S_0)$ is l-triangulatable. ∎

Theorem 8 *Let $G = (V,E)$ be a (k,l)-partition intersection graph. Algorithm B can, in $O(|V|^{kl+1})$ time, determine whether G can be l-triangulated, and produce the l-triangulation when it exists.*

Summary 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''_I if it exists. If there is no l-triangulation, Return No. Else, use G''_I to compute the l-load perfect phylogeny T.

Theorem 9 *The l-load perfect phylogeny problem 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 [25]. 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 $L = \sum_{c \in C} l_c$.

3.3 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 [22].
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 [22] 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 10 *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 $min\{O(r^{L+1}Ln^2), O(n^2L^2 + n(rkL^2)^{L+1})\}$.*

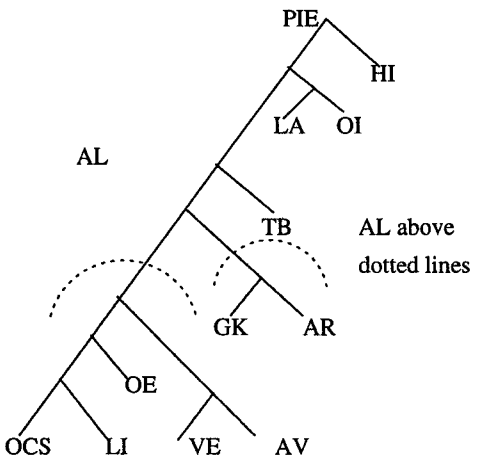
Proof: If the algorithm returns NO, it is clear that no perfect phylogeny within the constraints of the problem exists. If it returns YES, then the perfect phylogenies refining each of the stars can be hooked up via the new nodes. The refinement can be done by using the algorithms in Section 3.2. It 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 3.2 to determine the existence of perfect phylogenies on S_v.

4 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, [36] began with 220 characters, 185 of which were monomorphic and 35 of which were polymorphic. 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. [36] found that there was *one* perfect phylogeny on the monomorphic characters. We used Algorithm D on this phylogeny and verified that the entire set of characters (i.e. including the 36 polymorphic characters) was compatible with this single perfect phylogeny.

The discovery by Ringe, Taylor, and Warnow of the extent of polymorphism within linguistics led them to recheck each of the characters for evidence of polymorphism. This careful examination revealed that one of the characters used in their preliminary analysis (which originally appeared monomorphic, and upon which the presentation at the NAS Symposium on the Frontiers of Science in November 1995 was based) was polymorphic. Their subsequent search for new grammatical characters led to a discovery of a character (based upon the construction of abstract nouns) which they hoped might provide additional evidence for Indo-European subgroupings. Our algorithms, applied to the enlarged data set, now indicate weak support for the Italo-Celtic hypothesis. Thus, the tree they now posit (presented in Figure 1) as the evolutionary tree in Indo-European is different from their earlier hypothesis, and is the direct result of this analysis.



HI - Hittite, OI - Old Irish, LA - Latin
TB- Tocharian B, GK - Greek, AR - Armenian
OE - Old English, OCS - Old Church Slavic
LI - Lithuanian, AV - Avestan, VE - Vedic
AL - Albanian

Figure 1: The tree on the Indo-European data set.

5 Polymorphism in Biology

The evolution of biological polymorphic characters can be modelled using the following operations [26]. 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* [8, 9], but for fixed topology it is in P [15].

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

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

- If $|X| < |Y|$ then $cost(\alpha,(u,v)) = cost_d[|Y| - |X|] + cost_m|X|$.

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

We consider the following **problem**: Given a fixed leaf-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 11 *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 [19], 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 11. 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 11. 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. ∎

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 12 *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 labelling 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 $O(nkr^l)$-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 [32] 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 extend the special case algorithm if Sankoff and Cedergren for weighted parsimony of monomorphic characters with weights all 1. We can show 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 will have load l. Since we have only mutations, the cost of an edge is simply the number of mismatches between the labels at the opposite ends. Thus, if the best labelling for the subtree rooted at v has cost $c(v)$ (number of mismatches), then we need only consider labels with cost at most $c(v) - l + 1$. For each internal node v, we compute the set of labels with cost $c(v) - x$ for $0 \leq l - 1$ and store it in set $L(v, x)$. Set $L(v, 0)$ is never empty, but any of the others can be. For leaves, set $L(v, 0)$ is the single label and other sets are empty.

Because there can be r^l labels in the $L(v', x)$ sets of each child v' of node v, we must be somewhat careful when computing the sets $L(v, x)$ to achieve an overall time of r^l. Intuitively, a label at node v is good if the states in it appear in many of the labels of its children. To compute the $L(v, x)$ for node v, we begin by initializing an array of nl buckets. Each possible label for v begins in the 0th bucket (with pointers to them so they can be located quickly). For each child v', each label in set $L(v', x)$ earns $l - x$ *points*. This means that this label is $l - x$ better than the "worst" possible cost of $c(v') - l$. If label $l_{v'} \in L(v', x)$, then we locate v' in the set of buckets and move it $l - x$ buckets forward. We step through all $L(v', x)$ of all children in time $O(r^l)$ per child. Then we find the highest non-empty bucket b. All labels in bucket b are placed in set $L(v, 0)$. All labels, if any, in bucket $b - x$ are placed in set $L(v, x)$. We can compute all the sets $L(v, x)$ in time $O(nkr^l)$ time. ∎

6 Discussion

In this paper we introduce an algorithmic study of the problem of inferring the evolutionary tree in the presence of polymorphic data. We consider parsimony analysis for polymorphic data on fixed topologies, and present algorithms as well as hardness results. We also present 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] P. BUNEMAN, *A characterization of rigid circuit graphs*, Discrete Math 9 (1974), pp. 205-212.

[7] L. LUCA CAVALLI-SFORZA, P. MENOZZI AND A. PIAZZA, *The History and Geography of Human Genes*, Princeton University Press, 1994.

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

[9] W.H.E. DAY, D.S. JOHNSON, AND D. SANKOFF, *The computational complexity of inferring phylogenies by parsimony*, Mathematical biosciences, 81:33-42, 1986.

[10] W. H. E. DAY AND D. SANKOFF, *Computational complexity of inferring phylogenies by compatibility*, Syst. Zool., Vol. 35, No. 2 (1986), pp. 224–229.

[11] A. DRESS AND M. STEEL, *Convex tree realizations of partitions*, Appl. Math. Letters, Vol. 5, No. 3 (1992), pp. 3–6.

[12] 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.

[13] 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.

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

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

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

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

[18] R. IDURY AND A. SCHAFFER, *Triangulating three-colored graphs in linear time and linear space*, to appear, SIAM J. Discrete Mathematics.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.

[26] M. NEI, *Molecular Evolutionary Genetics*, Columbia University Press, New York. 1987.

[27] A. PROSKUROWSKI, *Separating subgraphs in k-trees: cables and caterpillars*, Discrete Math., 49 (1984), pp. 275-285.

[28] D. RINGE, personal communication, 1995.

[29] D.J. ROSE, *On simple characterization of k-trees*, Discrete Math., 7 (1974), pp. 317-322.

[30] 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.

[31] A.K. ROYCHOUDHURY AND M. NEI, *Human Polymorphic Genes: World Distribution*, 1988, Oxford University Press.

[32] 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.

[33] D. SANKOFF AND P. ROUSSEAU, 1975, *Locating the vertices of a Steiner tree in arbitrary space*, Mathematical Programming, 9:240-246.

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

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

[36] 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.*