Better Methods for Solving Parsimony and Compatibility

MARIA BONET,[1] MIKE STEEL,[2] TANDY WARNOW,[3] and SHIBU YOOSEPH[4]

ABSTRACT

Evolutionary tree reconstruction is a challenging problem with important applications in biology and linguistics. In biology, one of the most promising approaches to tree reconstruction is to find the "maximum parsimony" tree, while in linguistics, the use of the "maximum compatibility" method has been very useful. However, these problems are NP-hard, and current approaches to solving these problems amount to heuristic searches through the space of possible tree topologies (a search which can, on large trees, take months to complete). In this paper, we present a new technique, *Optimal Tree Refinement*, for reconstructing very large trees. Our technique is motivated by recent experimental studies which have shown that certain polynomial time methods often return *contractions* of the true tree. We study the use of this technique in solving maximum parsimony and maximum compatibility, and present both hardness results and polynomial time algorithms.

1. INTRODUCTION

THE TASK OF THE SYSTEMATIC BIOLOGIST is to recover the order of speciation events that gave rise to the observed species under investigation. This order of speciation events is represented in the *topology* of the evolutionary tree for the species. For scientific as well as information-theoretic reasons, locating the root of the evolutionary tree is quite difficult (and even impossible under certain models of evolution), so that the primary objective of a phylogenetic method is the recovery of the *unrooted* version of the tree that gave rise to the taxa.

Many of the popular methods in phylogenetic tree reconstruction attempt to solve NP-hard optimization problems and are usually implemented as heuristic searches through the space of different trees. *Maximum parsimony, maximum compatibility,* and *maximum likelihood* are three such optimization problems which have been used in biology, but polynomial time methods, such as *neighbor-joining* (Saitou and Nei, 1987), also exist. By assuming a model of how biomolecular sequences evolve, it is possible to explore the performance of these methods. A recent result by Tuffley and Steel (1997) showed that under a certain model of biomolecular sequence evolution (which is less constrained than what is usually assumed), maximum likelihood and maximum parsimony are equivalent. On the other hand, there are conditions under this general model in which these two methods (and indeed, any method at all (Steel *et al.*, 1994; Chang, 1996) are *inconsistent* (a method is said to be *inconsistent* under a model of evolution if there are some model trees in that model for which the probability that the method will recover the correct tree topology given infinite length sequences is not 1; see Felsenstein (1978)). There are conditions under which it is possible to guarantee that the topology of the tree can be recovered (with arbitrarily high probability, given long enough sequences), but

[1]Universidad Politecnica de Catalunya, Department Lenguages y Sist. Informaticos, Jordi Girona Salgado 1-3, 08034 Barcelona.

[2]Biomathematics Research Centre, University of Canterbury, Christchurch, New Zealand.

[3]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104.

[4]DIMACS, Rutgers University, 96 Frelinghuysen Road, Piscataway, New Jersey 08854.

these conditions are unreasonable with respect to biomolecular sequences, since they assume that the sites within the sequences evolve identically and independently (the *iid* assumption). Under the *iid* assumption, however, it is known that all "reasonable" distance-based methods are *consistent*, that is, each will recover the correct topology given long enough sequences (Erdös *et al.*, 1997; Steel, 1994). For this reason, some statisticians and biologists prefer distance-based reconstruction methods over parsimony-based methods.

However, heuristic methods for solving parsimony continue to be a major technique used by systematic biologists, even for large trees (See, for example, Rice *et al.* (1997), reporting on a parsimony-based analysis of a 500-taxon data set), despite its computational difficulties and its inconsistency under some models of evolution (Kim, 1996; Felsenstein, 1978). Furthermore, both experimental (Rice and Warnow, 1997; Huson *et al.*, 1998a) and analytical studies (Erdös *et al.*, 1997; Atteson, 1997; Ambainis *et al.*, 1997; Farach and Kannan, 1996) suggest that the actual sequence length that suffices for the various promising distance-based methods to obtain (with high probability) an accurate topology estimation may be exceedingly large if the true tree contains significant divergence, even under the *iid* assumption. Consequently, although distance-based methods tend to be fast and are provably consistent under some models of biomolecular sequence evolution, they may not obtain even *approximately* correct reconstructions of the topology of the evolutionary tree, if that evolutionary tree is very large and contains significant divergence. On the other hand, heuristic approaches for the maximum parsimony problem have performed very well in experimental studies, and often perform significantly better than the best polynomial time distance-based methods, such as neighbor-joining (Saitou and Nei, 1987) in reconstructing very large trees (Hillis, 1997; Rice and Warnow, 1997; Huson *et al.*, 1998a). For this reason, many important data sets are analyzed in biology using heuristic parsimony searches. Unfortunately these searches can last for months or longer (see Rice *et al.* (1997)) without necessarily finding the most parsimonious tree(s). This is partly due to the fact that parsimony is NP-hard to solve exactly (Foulds and Graham, 1982; Day, 1983), and also perhaps due to the techniques (primarily branch-swapping) that have been used to search the tree space. Obtaining faster and more accurate methods for solving parsimony is an important objective in phylogenetics.

Evolutionary studies in historical linguistics similarly have the primary objective of recovering the topology of the evolutionary tree, and here too the most effective optimization problem (the *Maximum Compatibility* problem) is NP-hard to solve (Bodlaender *et al.*, 1992; Day and Sankoff, 1986; Steel, 1992). However, computationally expensive but exact algorithms exist for the Maximum Compatibility problem (Agarwala and Fernandez-Baca, 1996; Agarwala and Fernandez-Baca, 1994; Kannan and Warnow, 1994; Kannan and Warnow, 1995) (see Warnow (1993) for a survey of results related to maximum compatibility). The application of these maximum compatibility algorithms to linguistic data for the Indo-European family of languages has potentially resolved conflicts that have troubled the historical linguistics community for centuries (Warnow, 1997; Warnow *et al.*, 1996). Further studies by the researchers in this project have indicated that these current algorithms are not fast enough to handle the more complex data sets they are encountering in analyzing other language families, such as *Dravidian*, and that faster algorithms for the Maximum Compatibility problem (and the related problems that apply to *polymorphic data* (Bonet *et al.*, 1996)) need to be developed (Tandy Warnow, personal communication).

In this paper, we describe new methods for solving parsimony and compatibility, and at the same time we present a new general technique for tree reconstruction. Our methods and general technique are motivated by recent experimental studies (Berry, 1997; Huson *et al.*, 1998; Huson *et al.*, 1998a), which demonstrated that some fast tree construction methods return with high frequency a contraction of the true tree (that is, the reconstructed tree can be obtained from the true tree by contracting some set of edges). Consequently, the true tree is a *refinement* of the reconstructed tree. We pose and study the *Optimal Tree Refinement* Problem, and consider this problem specifically for the maximum parsimony and maximum compatibility optimization criteria. Although some of our results are negative, showing that generally the optimal tree refinement problem is NP-hard to solve, we also obtain algorithms which will be efficient for many of the cases that we may expect to find in practice. We conclude with open problems.

2. DEFINITIONS

Definition 1: *We let $[r]^k$ denote the set of k-tuples over $\{1, 2, \ldots, r\}$.*

Definition 2: *Let S be a set of n taxa. Then T is **leaf-labelled** by S if the leaves of T are bijectively labelled by S.*

Definition 3:   *An **n-star** is a tree with n leaves and exactly one internal node.*

Definition 4:   *Let T_1 and T_2 be two trees each leaf-labelled by S. Suppose T_2 can be obtained from T_1 by contracting some of the edges in T_1. Then, T_1 is a **refinement** of T_2, and T_2 is a **contraction** of T_1.*

Definition 5:   *Let $G = (V, E)$ be a graph. For any $v \in V$, the **degree of v** is $degree(v) = |\{w \in V : (v, w) \in E\}|$, i.e., the number of edges incident to v. Given a path between two nodes, the **length** of this path is the number of edges that it contains.*

Definition 6:   *The **height** of a rooted tree T is the maximum length of all root-to-leaf paths in T.*

Definition 7:   *Let S be a set of taxa (i.e., species, languages, etc.). A **character** is a function $c : S \rightarrow \{1, 2, \ldots, r\}$. Each c(s) is called the **character state** of c on species s.*

Let C be a set of k characters. Each species in S is identified with a vector of length k over the integers, so that $S \subseteq [r]^k$. Thus, for the ith character in C, the character state of this character on a species $s \in S$ is equal to the ith component of the vector for s. When dealing with a set of aligned biomolecular sequences, each column (or site) in the alignment corresponds to a character. For example, assuming no insertions or deletions in the alignment, for DNA sequences, each character has four character states—A, C, T, and G.

Definition 8:   *Let T be a tree leaf-labelled by S and let L be a labelling that assigns a vector from $[r]^k$ to each internal node in T. Then the **cost** of T under this labelling L (denoted by cost(T, L)) is the total number of times each character changes state in the tree. This may also be calculated as*

$$cost(T, L) = \sum_{e \in E(T)} Hamming(e),$$

where Hamming(e) is the Hamming distance between the endpoints of e, i.e., $Hamming(e) = |\{i : x_i \neq y_i\}|$, where $e = (x, y)$.

The **parsimony score** of T is then the minimum of $cost(T, L)$, taken over all possible internal labellings L of the nodes of T. The problem of finding the tree with the lowest cost for a given set of sequences which will label the leaves of the tree is the **maximum parsimony problem.**

We now define the **maximum compatibility problem.** As before, let T be a tree leaf-labelled by S, and let L be an internal labelling of the nodes of T, where all labels and elements in S are drawn from $[r]^k$. Now consider a particular character c, and let r_c be the number of distinct values at the leaves for this character. It should be clear that no matter how the internal nodes of the tree are labelled in L, that the tree T must contain at least $r_c - 1$ edges on which this character changes state between the endpoints.

Definition 9:   *Any character c which has the property that on (T, L) it changes exactly $r_c - 1$ times is said to be **compatible** or **convex** on (T, L). A character which is not compatible on T is said to be **incompatible.** The number of such characters that are compatible on (T, L) is denoted by compat (T, L).*

The **compatibility score** of T is then the maximum of *compat* (T, L), as L ranges over all possible ways of labelling the internal nodes of T. The problem of finding the tree with the highest compatibility score for a given set S is the **maximum compatibility** problem.

Both *maximum parsimony* and *maximum compatibility* are NP-hard optimization problems, even when set C consists of binary characters (that is, each character $c \in C$ maps S to $\{0, 1\}$) (Day and Sankoff, 1986; Day, 1983). However, the related problems of determining the optimal labelling (with respect to either the parsimony or compatibility criteria) of a fixed leaf-labelled tree is solvable in polynomial time, using the Fitch-Hartigan algorithm (1973).

There is a relationship between maximum compatibility and maximum parsimony, which we now describe. When it is possible to obtain a tree T for a given set S of taxa defined by characters C, on which all characters are compatible, then T also has the minimum possible parsimony score. Consequently, such a tree optimizes both the maximum compatibility and the maximum parsimony problem. When such a tree exists, it is called a **perfect phylogeny**, and the set C of characters is said to be **compatible.** Consequently, the perfect phylogeny problem is also known in the literature as the **character compatibility** problem. (See Warnow (1993), Warnow (1996) and Felsenstein (1982) for a review of this and related problems.)

As noted earlier in this paper, some experimental performance studies suggest that some methods frequently return contractions of the true tree. This motivates the following problems.

Optimal Tree Refinement with respect to Parsimony: (OTR-parsimony)

Input: A tree T which is leaf labelled by S (described by a set C of characters).

Output: A refinement T' of T, such that the cost of T' is the least among all refinements of T.

Optimal Tree Refinement with respect to Compatibility: (OTR-compatibility)

Input: A tree T which is leaf labelled by species from S (described by a set C of characters).

Output: A refinement T' of T, such that the compatibility score of T' is the largest among all refinements of T.

Throughout this paper, we will use the following notation for the parameters to the problem. We will let

- **S** denote the set of taxa (whether languages, biological species, or genes) with $\mathbf{n} = |S|$.
- **C** denote the set of characters with $\mathbf{k} = |C|$.
- **r** denote the maximum number of states per character.
- **d** denote the maximum degree of any node in T.

Some of these parameters may in general be expected to be small on some of the data sets we examine. For example, in biology, biomolecular data (such as DNA or RNA sequences) are typically used to reconstruct trees, and for such data r is typically 4. When the tree to be refined is significantly resolved, then d may be quite small, and in general we may expect d to be not particularly large. On the other hand, k is the sequence length, and when using biomolecular sequences k can be on the order of hundreds or thousands, and n may take any number in a range between, say, 10–1,000.

As discussed earlier, the maximum compatibility and maximum parsimony problems are NP-hard even for $r = 2$; consequently, OTR-parsimony and OTR-compatibility will be NP-hard in general. The related perfect phylogeny problem is NP-hard in general, but solvable in polynomial time for the various fixed-parameter versions of the problem (i.e., when r, k, or n is fixed). (See Kannan and Warnow (1995), Agarwala and Fernandez-Baca (1996), Agarwala and Fernandez-Baca (1994), Bodlaender *et al.* (1992), Steel (1992) and McMorris *et al.* (1993).)

3. OTR-PARSIMONY

3.1. Preliminary results on OTR-parsimony

Theorem 1.   *OTR-parsimony is*

(i) *Solvable in polynomial time when n is bounded,*
(ii) *NP-hard for $r = 2$, if k, d, and n are unbounded,*
(iii) *NP-hard for $k = 1$, if r, n, and d are unbounded.*

Proof.

(i) If n is bounded, then OTR-parsimony is tractable, since an exhaustive search through all the refinements of the tree T combined with the computation of the optimal labellings of the tree (using Fitch (1971) and Hartigan (1973)) is a polynomial time algorithm.
(ii) The reduction is from the maximum parsimony problem on binary characters (shown to be NP-hard (Day, 1983)). Given an instance of the maximum parsimony problem with n binary sequences each of length k, we create an instance of the OTR-parsimony problem by letting T be an n-star and labelling each leaf in T distinctly with one of the n sequences. It is then easily seen that a solution for the maximum parsimony problem is a solution for the OTR-parsimony problem and vice-versa.

(iii) We now prove that OTR-parsimony is NP-hard for $k = 1$, when the degree of the tree, the number of states per character, and the number of leaves are all unbounded. The proof is by a reduction from Vertex Cover which has been shown to be NP-complete (Garey and Johnson, 1979).

Vertex Cover Problem .

Input: A graph $G = (V, E)$ and an integer B.

Question: Does G have a vertex cover of size B, that is, is there a subset $V_1 \subseteq V$ of size B such that for all edges $(a, b) \in E$ at least one of a or b is in V_1?

Let $(G = (V, E), B)$ be an input to Vertex Cover. The input to OTR-parsimony is defined as follows. We make $degree(v)$ copies of each vertex v in V, and these constitute the species set S. We label these species by the pair (v, e), so that $e = (v, w)$ for some w. Thus, $|S| = 2|E|$. There is one character c defined on S by $c(v, e) = v$. Thus c has $|V|$ states. The topology of the tree T is as follows. T has an internal node r with $|E|$ neighbors, each labelled by an edge from G. The node in T labelled by e is adjacent to (v, e) and (w, e), where $e = (v, w)$. Thus the tree T has $3|E| + 1$ nodes: an inner node r, a middle ring for the edges from G (called the "edge" vertices), and the outer ring of leaves, two to each node in the middle ring. We can then show that G has a Vertex Cover of size B if and only if we can refine T so that the resultant tree T' has a parsimony score of at most $B - 1 + |E|$.

Suppose G has a Vertex Cover V_0 of size L, and let $V_0 = \{v_1, v_2, \ldots, v_L\}$. For each edge $e \in E$, pick an endpoint within V_0 and call it e^*. We now define the topology of T'.

For each $v \in V_0$, let $E_v = \{e \in E : e^* = v\}$. This defines a partition of E into $L' \leq L$ sets. We will introduce into T L' new nodes, one for each distinct selected e^*. Make every "edge" vertex e in T a child of the new node e^*, and let e^* be adjacent as well to the internal node r. This is a refinement of T. We set the c states as follows: $c(e) = c(e^*) = e^*$, but $c(r)$ is selected arbitrarily from V_0. To compute parsimony cost of this tree, note that the only edges on which any change occurs are from r to the new nodes with different labels, and exactly one edge incident to a leaf from each "edge" node. Thus the parsimony cost of T' is $L' - 1 + |E| \leq L - 1 + |E|$.

For the converse, suppose that a refinement T' of T has parsimony cost at most $L - 1 + |E|$. The only refinements of this tree are refinements around the node r, and no matter how one sets the states of the nodes, there is always at least one leaf-incident edge from each "edge" vertex e that contributes to the cost. Hence the parsimony cost is at least $|E|$. To obtain a refinement that has parsimony cost at most $L - 1 + |E|$ means that we can refine around r so that the introduced nodes each contribute at most $L - 1$. Without loss of generality, we can assume that all the introduced nodes have distinct c states, and hence there are at most L introduced nodes. Let these be $a_1, a_2, \ldots, a_p$ for $p \leq L$, and without loss of generality assume $c(r) = c(a_1)$. If $c(a_i) \neq c(e)$ for some edge node e adjacent to a_i, then we can make e adjacent to r directly. Thus, without loss of generality, $c(a_i) = c(e)$ for all e incident to a_i. Furthermore, if $c(e) \notin \{a, b\}$ for $e = (a, b)$, then by changing $c(e)$ to be one of a, b, we can modify the c states, and then the topology of the tree, without worsening the parsimony score. Thus, we can assume that $c(e) = c(a_i) \in \{a, b\}$ for $e = (a, b)$ and e incident to a_i.

We now show that the set $A = \{c(a_i), i = 1, 2, \ldots, p\}$ defines a vertex cover (we already know that the number of new nodes is at most L). Suppose not, so that there is some $e = (a, b) \in E$ such that $\{a, b\} \cap A = \emptyset$. Since $c(e) = c(a_i) \in \{a, b\}$, this is a vertex cover. ∎

The proof of Theorem 1(i) is given by a simple exhaustive search algorithm through all tree topologies. Since we will use that algorithm in a later section, we give it here:

Algorithm 1. *Given T, we root it at some node. Then, we simply examine each refinement and compute its optimal labelling using Fitch-Hartigan (Fitch, 1971; Hartigan, 1973). This uses $O((2d-3)!!)^{d^h/(d-1)}nkr)$ time, for a tree of height h, degree d, and having $n \leq d^h$ leaves, where $(2d - 3)!! = (2d - 3)(2d - 5)(2d - 7) \ldots 3$.*

3.2. Dynamic programming algorithms for OTR-parsimony

In this section we present three other algorithms for OTR-parsimony. Algorithm 2 is an exact algorithm, Algorithm 3 is a 2-approximation, and Algorithm 4 is a Polynomial Time Approximation Scheme (PTAS). All these algorithms share a common structure and are based upon a dynamic programming formulation of the OTR-parsimony problem.

Recall that parsimony is the *Steiner Tree problem for Hamming distances* in which the input is a set S of sequences of length k over the alphabet $1, 2, \ldots, r$, and any sequence of the same length can label the internal nodes of the evolutionary tree. However, if we restrict the set of labels that may be used at the internal nodes (the *Steiner points*), then we can *approximate* the optimal parsimony tree; for example, if we restrict these labels to be drawn from the input set S, then we obtain a 2-approximation, and it is achievable in polynomial time using standard minimum spanning tree algorithms (this is a well-known and classical result in phylogenetics, but see Warnow (1996) for a proof). Thus, although accuracy is compromised, the decrease in running time can be significant.

As we will show, we may also obtain approximate solutions for OTR-parsimony and bound the loss of accuracy through restricting the set of Steiner points.

The OTR-parsimony problem allows the internal nodes of the refinement to be assigned any sequence in $[r]^k$, and the variations we will consider will permit the internal nodes to be drawn from proper subsets of $[r]^k$.

Definition 10:   *Let $\mathcal{L}$ and A denote subsets of $[r]^k$. Let $pars_{\mathcal{L}}(A)$ denote the parsimony score of the most parsimonious tree for leaf set A, given that all internal nodes are labelled by $\mathcal{L}$. Let $OTR_{\mathcal{L}}$-Parsimony be the problem:*

- **Input:**   *Tree T leaf-labelled by $S \subset [r]^k$*
- **Output:**   *Tree T' refining T with all internal nodes labelled by sequences drawn from $\mathcal{L}$, of minimum parsimony score.*

Thus, if $\mathcal{L} = [r]^k$, then we have the original OTR-parsimony problem, but if we restrict $\mathcal{L}$ we may obtain faster algorithms at the cost of accuracy with respect to the original OTR-parsimony problem. We will show that by restricting $\mathcal{L}$ to be S, we can obtain a 2-approximation algorithm, and that by a variant in which some nodes in the refinement are labelled by S and others have labels drawn from $[r]^k$ we can obtain a polynomial time approximation scheme (PTAS). Both these approximation algorithms and the exact algorithm for OTR-parsimony have a dynamic programming structure, which we now describe.

Definition 11:   *Let T' be an arbitrary refinement of T. We associate each node in T to a unique node in T', as follows. For $v \in V(T)$, we denote by T_v the subtree of T rooted at v. For each node $v \in V(T)$, let S_v be the set of leaves in T_v; we then associate to v the node $v' \in V(T')$ defined by $v' = lca_{T'}(S_v)$. We will call this mapping from $V(T)$ to $V(T')$ the **canonical mapping**, and denote by v' the image of v in T' under the canonical mapping. Note that this natural mapping is injective. Given a node $v \in V(T)$ with children $v_1, v_2, \ldots, v_p (p \le d)$, let $F_{T'}(v)$ denote the minimal subtree of T' containing the node set $\{v', (v_1)', (v_2)', \ldots, (v_p)'\}$. Let $s \in \mathcal{L}$, and let $v \in V(T)$. Then $cost_{\mathcal{L}}(v, s)$ will denote the cost of the most parsimonious refinment of T_v given that v is labelled by s and all internal nodes of the refinement are drawn from $\mathcal{L}$. (In other words, $cost_{\mathcal{L}}(v, s)$ is the parsimony score of the solution to $OTR_{\mathcal{L}}$-parsimony applied to the set $L_v \cup \{s\}$, where L_v denotes the sequences labelling the leaves in T_v.)*

Thus, we can solve $OTR_{\mathcal{L}}$-parsimony by finding the minimum $cost(root, s)$, as s ranges over $\mathcal{L}$. Furthermore, if we compute all $cost(v, s)$ from the bottom up, then solving $cost(w, s)$ depends only upon the previously computed $cost(v, s')$ for $v \in \text{Children}(w)$ and $s' \in \mathcal{L}$, as we now show.

Let w have children $w_1, w_2, \ldots, w_p$, and let T' be a solution to $OTR_{\mathcal{L}}$-parsimony for $L_w \cup \{s\}$. Let $w'_1, w'_2, \ldots, w'_p$ be the nodes in T' associated to $w_1, w_2, \ldots, w_p$, and let $A \subseteq \mathcal{L}$ be the set of sequences labelling $w'_1, w'_2, \ldots, w'_p$ in T' given the optimal $\mathcal{L}$-labelling. Note that $|A| \le p \le d$, since A may have fewer than p sequences in it, if some sequence labels more than one node in the tree T'.

Now, the parsimony score of T' is the sum of the weights of all edges, and the edges in T' can be partitioned into $p + 1$ sets, as follows. Let B_i be the edges in T' below w_i, for $i = 1, 2, \ldots, p$, and let B_0 be the remaining edges, so that B_0 are edges which fall between w' and w'_i for some i. Then, it is easy to see that

$$cost(w, s) = \sum_{i=0}^{p} \sum_{e \in B_i} Hamming(e).$$

Now,

$$\sum_{e \in B_0} Hamming(e) = pars_{\mathcal{L}}(A \cup \{s\}),$$

and

$$\sum_{e \in B_i} Hamming(e) = cost(w_i, a),$$

for some $a \in A$. Therefore,

$$cost(w, s) = min_{A \subseteq \mathcal{L}} \left\{ pars_{\mathcal{L}}(A \cup \{s\}) + \sum_{i}^{p} best(w_i, A) \right\},$$

where the minimum is taken over all subsets $A \subseteq \mathcal{L}$ of cardinality at most p, and where

$$best(w_i, A) = min\{cost(w_i, a) : a \in A\}.$$

These observations allow us to formulate a dynamic programming algorithm for $OTR_{\mathcal{L}}$-parsimony as follows:

Dynamic Programming Algorithm for $OTR_{\mathcal{L}}$-parsimony:

1. **Preprocessing:** Compute $pars_{\mathcal{L}}(A')$ for all sets $A' \subseteq \mathcal{L}$ of at most $d + 1$ elements.
2. **Compute** $cost(w, s)$ for all $w \in V(T)$ and $s \in \mathcal{L}$: Starting at the leaves and moving upwards towards the root, compute

$$cost(w, s) = min_A \left\{ pars_{\mathcal{L}}(A \cup \{s\}) + \sum_{i=1}^{p} best(w_i, A) \right\},$$

where the minimum is taken over all subsets $A \subseteq \mathcal{L}$ of at most d elements.
3. Return $min_{s \in \mathcal{L}}\{cost(root, s)\}$.

Running time of the dynamic programming algorithm. We now discuss the running time of this dynamic programming algorithm for $OTR_{\mathcal{L}}$-parsimony.

We have two approaches for preprocessing, in which we compute $pars_{\mathcal{L}}(A')$ for all sets $A' \subseteq \mathcal{L}$ of cardinality at most $d + 1$. The first approach actually computes this value, but the second approach potentially obtains a better value than can be obtained by using only labels from $\mathcal{L}$; this does not cause us any problem for our applications, since we are using $OTR_{\mathcal{L}}$-parsimony in order to approximate OTR-parsimony.

The first of our two approaches simply examines all sets $B \subseteq \mathcal{L}$ of cardinality at most $2d - 1$, and computes the minimum spanning tree on B. This technique amounts to determining not only the labels for the nodes w' and $w'_1, w'_2, \ldots, w'_p$, but also for all the nodes that will be introduced into T' between w' and w'_i for some i. This technique uses $O(d^2 k |\mathcal{L}|^{2d})$ time.

The other approach computes $pars_{[r]^k}(A')$ for all sets $A' \subseteq \mathcal{L}$ of cardinality at most $d + 1$. This may obtain better scores than can be obtained by solving $pars_{\mathcal{L}}(A')$, since the set of Steiner points is larger, but allows us to use standard techniques (exhaustive search and branch-and-bound) in conjunction with the $O(drk)$ Fitch-Hartigan (Hartigan, 1973) algorithm. There are $f(d) = (2d - 3)(2d - 5) \ldots 3$ possible trees on $d + 1$ leaves which have to be examined, so that this approach uses $O(rkd|\mathcal{L}|^{d+1} f(d))$ time. Since $f(d) < |\mathcal{L}|^d$, this second technique is faster than the first.

The third step, computing $min_{s \in \mathcal{L}}\{cost(root, s)\}$, takes only $O(|\mathcal{L}|)$ time. Consequently, we summarize this as follows:

Theorem 2. *We can "solve" $OTR_{\mathcal{L}}$-parsimony in $O(nrkdf(d)|\mathcal{L}|^{d+1})$ time.*

(Note: We have put quotes around "solve" in order to emphasize that using the algorithm described here, we may in fact obtain solutions that are better than are achievable by only permitting labels to be drawn from $\mathcal{L}$. Since our restrictions upon $\mathcal{L}$ are made only to improve upon the running time and at the same time obtain bounds upon the approximation ratio, this is acceptable to us.)

Finally, it is worth noting that although the dynamic programming formulation is stated with respect to computing the parsimony score of the optimal refinement under the restriction upon Steiner points, it is simple to modify the algorithm so as to obtain the optimal refinement as well.

3.3. Algorithm 2: Exact algorithm for OTR-parsimony

We can obtain an exact solution to the OTR-parsimony problem if we set $\mathcal{L} = [r]^k$. Applying the algorithm for $OTR_{\mathcal{L}}$-parsimony in this case, we obtain:

Theorem 3. *The optimal refinement of a tree with n leaves labelled by elements of $[r]^k$ and degree bound d can be computed in $O(nkr^{k(d+1)+1}df(d))$ time.*

3.4. Algorithm 3: 2-Approximating OTR-parsimony

We now show that solving OTR_S-parsimony yields a 2-approximation algorithm for OTR-parsimony, where S is the set of sequences labelling the leaves of the tree T. In fact, we can show that a 2-approximate solution can be obtained in $O(n^{d+1}d^3(\log d + k))$ time through a slight modification to the OTR_S-parsimony algorithm. These results are based upon Wang et al. (1996), which we now summarize.

Assume T is rooted and leaf-labelled by elements of $S \subseteq [r]^k$. We now use some techniques and ideas, with appropriate extensions to this problem, from Wang et al. (1996). We define a *lifted labelling* of a tree T to be an assignment of labels to the internal nodes of T such that for each node v in T, the label for v is drawn (i.e., *lifted*) from the set of labels of its children. (Our terminology differs from that in Wang et al. (1996), in which a lifted labelling is called a *lifted* tree.) Since in the OTR problem, we simultaneously seek a refinement as well as a labelling, we now extend the definition of a lifted labelling as follows. We say that T' is a **lifted labelled-refinement** of T if T' refines T, and for each internal node v in T', the label for v is drawn from the labels of its children. We now return to the results in Wang et al. (1996):

Theorem 4. *(From Wang et al. (1996)): For any tree T and any scoring function on pairwise alignments which satisfies the triangle inequality, there is a lifted labelling of cost no more than twice the cost of an optimal tree alignment.*

Theorem 5. *Let T be an arbitrary tree leaf-labelled by elements of $[r]^k$. Then there exists a lifted labelled refinement of T of cost at most twice the cost of the optimal refinement.*

Proof. Note that the Hamming distance between two sequences is a scoring function on pairs of sequences, in which deletions or insertions have a very large cost, and any substitution has the same cost as any other. Consequently, we can apply Theorem 4. Let T' be the optimal refinement for T; hence, each node in T' is labelled by elements of $[r]^k$, and T' refines T. If we consider T' as a leaf-labelled tree, then by Theorem 4, there exists a lifted labelling of T' yielding a 2-approximation for the optimal tree alignment problem on T'. T' with this lifted labelling of T' is a lifted labelled refinement of T with parsimony score at most twice that of the optimal. ∎

Thus, to 2-approximate the optimal refinement of T, we find an optimal lifted labelled refinement of T. This is done using the dynamic programming framework discussed in Section 3.2. If we set $\mathcal{L} = S$, then at each stage computing $cost_{\mathcal{L}}(A(y, s))$ is reduced to computing the Minimum Spanning Tree (MST) with the set $A' \subseteq S$ and $A' \leq d$. Note that the MST computation combined with the bottom-up approach of the dynamic programming also ensures that the labelling is lifted. Since $|S| = n$, there are $O(dn^d)$ sets A', and each MST computation can be done in $O(d^2(\log d + k))$. Thus the total running time is $O(n^{d+1}d^3(\log d + k))$.

Theorem 6. *We can obtain a 2-approximation to the OTR-parsimony problem in $O(n^{d+1}d^3(\log d + k))$ time.*

3.5. A PTAS

It is also possible to modify the approximation scheme given in Wang et al. (1996) (and reanalyzed in Wang and Gusfield (1997)) to obtain an approximation scheme for OTR-parsimony. We begin by discussing the material in Wang et al. (1996) and in Wang and Gusfield (1997), which is stated in terms of solving the *optimal tree alignment* problem which, as we showed earlier, relates directly to the OTR-parsimony problem.

Definition 12: *Let T be a rooted leaf-labelled tree and let X be a subset of $V(T)$ containing the leaves and the root of T. For $v \in X$, a X-child of v is a node $w \in X$ in the subtree of T_v such that the internal nodes*

*on the path from v to w in T are not also in X. A X-**lifted labelling** of T is a labelling of the internal nodes of T such that the label for each node in X is drawn from its X-children. A b-**family** for a tree T is a set $\mathcal{V} = \{V_0, V_2, \ldots, V_{b-1}\}$ of subsets of $V(T)$ such that $V_i \cap V_j = \{root(T)\} \cup \mathcal{L}(T) \forall i, j, i \neq j$, where $\mathcal{L}(T)$ indicates the leaves of T.*

Theorem 7. *Let $\mathcal{V} = \{V_0, V_1, \ldots, V_{b-1}\}$ be a b-family for a tree T, and let T_i be an optimal V_i-lifted labelling of T. Then $\sum_i cost(T_i) \leq (b + 2 - \frac{2}{2^b}) cost(T^*)$ where T^* is T equipped with an optimal internal labelling.*

In Wang and Gusfield (1997), this theorem is proved for a particular b-family, but the proof goes through for arbitrary b-familes as well. We continue.

Definition 13: *T' is a X-**lifted labelled refinement** of T if T' refines T and if the labelling of the internal nodes of T' is a X'-lifted labelling, where X' is the image of X under the canonical mapping.*

Using techniques similar to those used in earlier proofs, we can prove the following:

Theorem 8. *Let $\mathcal{V} = \{V_0, V_2, \ldots, V_{b-1}\}$ be a b-family for T, and let T_i be the optimal V_i-lifted refinement of T. Then*

$$\sum_i cost(T_i) \leq \left(b - 2 - \frac{2}{2^b}\right) cost(T^*)$$

where T^* is an optimal labelled refinement of T. Hence for some i,

$$cost(T_i) \leq \left(1 + \frac{2}{b} - \frac{2}{b2^b}\right) cost(T^*).$$

We now define the PTAS, that is, a family of algorithms, $\{\mathcal{A}_b, b = 1, 2, \ldots\}$, where $\mathcal{A}_b$ has a guaranteed approximation ratio of $(1 + \frac{2}{b} - \frac{2}{b2^b})$.

Algorithm $\mathcal{A}_b$:

- Step 1: Compute a b-family $\mathcal{V} = \{V_0, V_1, \ldots, V_{b-1}\}$.
- Step 2: Compute an optimal (S, V_i)-lifted labelled refinement T_i of T for each $i = 0, 1, \ldots, b - 1$.
- Step 3: Let T' be the T_i of minimum parsimony score.

We now present the details of how we accomplish each of these steps.

Step 1: The b-family is computed as follows. Partition the nodes of T by levels so that each node v is in level i, where $i = dist(v, root)$. Hence, the root is at level 0, the children of the root are at level 1, etc. V_i consists of the root, internal nodes at levels which are congruent to i modulo b, and all the leaves. Clearly, $\mathcal{V}_b = \{V_0, V_2, \ldots, V_{b-1}\}$ is a b-family for T.

Step 2: We show how to compute an optimal (S, V_i)-lifted labelled refinement of T for each $i = 0, 1, 2, \ldots, b - 1$.

- **Stage 1:** For each node $v \in V_i$ let $X(v)$ denote the V_i-children of v, and let $T(v)$ denote the minimal subtree of T containing at v and $X(v)$; thus $T(v)$ is rooted at v and has leaf set $X(v)$. Note that $T(v)$ has depth at most b and degree at most d, and hence at most d^b leaves. For each S-labelling L of the nodes in $\{v\} \cup X(v)$, let $\phi(v, L)$ denote the cost of the optimal labelled refinement of $T(v)$. The cost of computing $\phi(v, L)$ is $O(f(d^b)rkd^b)$ at worst, where $f(d^b)$ denotes the number of trees on d^b leaves. Hence, we can complete stage 1 in $O(f(d^b)n^{d^b}rk)$ time.
- **Stage 2:** We define $\Phi(y, s)$ to be the cost of the optimal (S, V_i)-lifted labelling refinement of T_y such that s is the label assigned to y. We now describe the dynamic programming technique for computing $\Phi(y, s)$. Suppose we have computed $\Phi(v, s)$ for each $v \in V_i$ below y and each $s \in S$, and we now wish to compute $\Phi(y, s)$. Let $v_1, v_2, \ldots, v_x$ be the V_i-children of y. For each S-labelling L of $\{v\} \cup X(v)$ such that v is

labelled s, we have already computed $\Phi(v_j, L(v_j))$, and we have already computed $\phi(y, L)$. Hence we can set $\Phi(y, s)$ to be the minimum over all such L of $\phi(y, L) + \sum_j \Phi(v_j, L(v_j))$. There are $O(n^{d^b})$ such L for each y, s, and hence this costs us $O(n^{d^b+2})$ time.

- **Stage 3:** To find the minimum cost S-labelled refinement, find $min_{s \in S}\{\Phi(root, s)\}$.

The runtime analysis we have given shows that for fixed i, b with $0 \le i \le b - 1$ and fixed tree T, we can compute an optimal (S, V_i)-labelled refinement of T in $O(f(d^b)rkn^{d^b+2})$ time.

Step 3: Having found the optimal (S, V_i)-labelled refinement T_i of T for each $i = 0, 1, 2, \ldots, b - 1$, we then select the best of all such refinements.

We obtain then:

Theorem 9. *We can find tree T' such that cost $(T') \le (1 + \frac{2}{b} - \frac{2}{b2^b}) cost(T^*)$ where T' is a refinement of T and T^* is an optimal refinement of T, in $O(f(d^b)rkn^{d^b+2})$ time.*

4. OTR-COMPATIBILITY

4.1. Preliminary results on OTR-compatibility

Theorem 10. *OTR-compatibility is*

(i) *Solvable in polynomial time when n is bounded.*
(ii) *NP-hard for $r = 2$, if k, d, and n are unbounded.*
(iii) *NP-hard when both $r \ge 4$ and $d \ge 5$ are fixed.*

Proof. Parts (i) and (ii) are similar to Theorem 1. (The reduction in part (ii) is from the maximum compatibility problem on binary characters shown to be NP-hard in Day and Sankoff (1986).) (iii) We now prove that the version of OTR-compatibility for $r = 4$ and $d = 5$ is NP-hard. The reduction is from the NP-Complete problem, Exact Cover by 3-Sets (Garey and Johnson, 1979). ∎

Exact Cover by 3-Sets problem

Input: A set $X = \{x_1, x_2, \ldots, x_{3q}\}$ and a set $S = \{Y_1, Y_2, \ldots, Y_m\}$ where each $Y_i \in S$ is a three-element subset of X.

Question: Is there a subset $S' \subseteq S$ with $|S'| = q$ such that $\cup_{Y_i \in S'} Y_i = X$?

We will consider the version of this problem in which each element $x_i \in X$ appears in at most three elements of S. This version is also known to be NP-complete (Garey and Johnson, 1979).

Let I be an instance of the Exact Cover by 3-Sets problem. We will construct an instance I' of the OTR-compatibility problem as follows: The tree T will have the topology as shown in Figure 1.

The idea is to have each element of X correspond to an internal node of T. Thus T has $|X|$, or $3q$, internal nodes. Each internal node v_i (corresponding to element $x_i \in X$) has 3 leaves incident on it. Thus the total number of species in I' is $3|X|$, or $9q$. We will use s_i^1, s_i^2, s_i^3 to denote the three species adjacent to the internal node v_i in T. Also, each element of S will have a character that corresponds to it. Thus we will have m characters in the character set of I'. We will use $C = \{c_l | Y_l$ is an element of $S\}$ to denote the set of characters.

Each $c_l \in C$ will be a 4-state character with states 0, 1, 2, 3, and is such that for each i, either $c_l(s_i^1) = c_l(s_i^2) = c_l(s_i^3) = 3$, or two of the species in the set $\{s_i^1, s_i^2, s_i^3\}$ will have state 3 and the other species will have
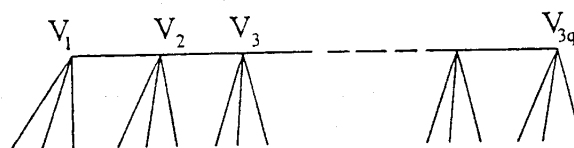

FIG. 1. Topology of tree T.

state 0, 1 or 2. We will now describe how the states are assigned to the species. Let $Y_{l_1} = \{x_{i_1}, x_{i_2}, x_{i_3}\}$. Now, for all $i \notin \{i_1, i_2, i_3\}$, set $c_{l_1}(s_i^1) = c_{l_1}(s_i^2) = c_{l_1}(s_i^3) = 3$. Let x_{i_1} also appear in Y_{l_2} and Y_{l_3}. We will describe how to set the character states for the characters c_{l_1}, c_{l_2} and c_{l_3} on the species $s_{i_1}^1, s_{i_1}^2, s_{i_1}^3$. Each of the three characters will have the state 3 on two of the species and one of either 0, 1, or 2, whichever has been *unused before*, on the third species. Without loss of generality, assume that for c_{l_1}, we have not used any of the states 0,1 or 2 earlier. Similarly, for c_{l_2}, suppose we have already used state 0 earlier and for c_{l_3}, we have already used states 0 and 1 earlier for some species. For the species $s_{i_1}^1, s_{i_1}^2, s_{i_1}^3$, we will use states 0, 1, and 2 respectively for c_{l_1}, c_{l_2}, and c_{l_3}. The setting will be as follows: $c_{l_1}(s_{i_1}^1) = c_{l_1}(s_{i_1}^2) = 0$, $c_{l_1}(s_{i_1}^3) = 3$, $c_{l_2}(s_{i_1}^1) = c_{l_2}(s_{i_1}^3) = 1$, $c_{l_2}(s_{i_1}^2) = 3$, $c_{l_3}(s_{i_1}^2) = c_{l_3}(s_{i_1}^3) = 2$, $c_{l_3}(s_{i_1}^1) = 3$.

Claim: The tree T has a refinement T' in which q characters are compatible if and only if the instance I has an exact cover, i.e., there exist a subset $S' \subseteq S$ such that $\cup_{Y_i \in S'} Y_i = X$ and $|S'| = q$.

Proof. Suppose T has a refinement T' in which q characters are compatible. Let $C' = \{c_{l_1}, c_{l_2}, \ldots, c_{l_q}\}$ denote this set of compatible characters. Then we claim that $Y_{l_1}, Y_{l_2}, \ldots, Y_{l_q}$ is an exact cover. To see why, suppose that Y_{l_1} and Y_{l_2} share an element x_{i_1} in common. Then, looking at species $s_{i_1}^1, s_{i_1}^2$ and $s_{i_1}^3$, we see that on no refinement of T can both c_{l_1} and c_{l_2} be made compatible at the same time. Thus the set C' is pairwise disjoint and since it has cardinality q, we infer that $Y_{l_1}, Y_{l_2}, \ldots, Y_{l_q}$ is an exact cover.

Suppose that instance I has an exact cover, denoted by $Y_{l_1}, Y_{l_2}, \ldots, Y_{l_q}$. Then it can easily be seen that the set of characters given by $C' = \{c_{l_1}, c_{l_2}, \ldots, c_{l_q}\}$ can be made compatible on a refinement of T. It is clear how to obtain this refinement and thus we omit this part. ∎

We have shown that OTR-compatibility is NP-hard for unbounded degree trees with binary characters and also for bounded degree trees with $r \geq 4$. In the following sections we will show:

1. OTR-compatibility is hard to approximate, even when d is bounded.
2. OTR-compatibility on binary characters can be solved in polynomial time for every fixed degree bound d.
3. The imperfection of a set of binary characters can be 2-approximated in $O(k^2 n)$ time, and can be solved exactly in $O(k^2 n + kt + (4/3)^t t^2)$ time.
4. Maximum compatibility of r-state characters can be solved in $O(2^{2r} k^{t+2} n)$.

4.2. Preliminary material about binary character compatibility

In the earlier discussion of compatibility, we stated that a character $c : S \to Z$ is "compatible" (or "convex") on a tree T leaf labelled by S if the internal nodes of the tree can each be labelled by states for c so that c changes state exactly $r_c - 1$ times (where r_c is the number of states c attains on S), and a set C of characters is compatible if there exists some tree T on which all the characters in C are compatible.

There are particular statements we can make when the characters are binary, because then the space of trees on which the characters are compatible has a unique minimal element with respect to refinement.

First, we define the *character encoding* of the tree T as follows.

Definition 14: *Each edge e in the tree T defines a bipartition on the leaves of T and hence can be described by a binary character c_e. The collection $\{c_e : e \in E(T)\}$ is the* **character encoding** *(or set of splits) of T, and is denoted by $C(T)$.*

The following lemma is part of the standard literature (see Gusfield (1991)):

Lemma 1. *Let C be a collection of binary characters defined on a set S. Then there is a tree T on which every character in C is compatible if and only if every pair of characters in C is compatible. Furthermore, given a set C of compatible binary characters, there is a unique tree T satisfying $C(T) = C$.*

Now, consider the OTR-compatibility problem, in which we are given a tree T and a set C of binary characters defined on the leaves of T, and we wish to find a refinement of T on which the maximum number of characters in C are compatible. We will show we can solve OTR-compatibility on a tree T by solving Max Compatibility on a derived set of characters.

Note the following:

Observation 1. *Suppose that T' is some refinement of T, and that $c \in C$ is compatible on T' (for some labelling L). Then c is compatible with every character in $C(T)$.*

Consequently, we can *preprocess* the set C by deleting from C any character which is incompatible with some character in $C(T)$. (Note that this is not the same as deleting characters which are incompatible *on* T.) This preprocessing can be done in $O(nk)$ time using an algorithm in Gusfield (1991) for "tree-compatibility." We will call the reduced set of characters C^*.

We will now prove the following:

Theorem 11. *Let T be a tree with a set C of binary characters defined on the leaves of T. Let T' be the optimal refinement of T with respect to compatibility, and let C' be the characters in C which are compatible on T'. Then C' is a maximum compatible subset of C^*, where $C^* = \{c \in C : c$ is compatible with every character in $C(T)\}$. Conversely, if C' is a maximum compatible subset of C^*, then the tree T' defined by $C(T') = C(T) \cup C'$ is an optimal refinement of T with respect to compatibility.*

Proof. We first prove that the size of a maximum compatible subset of C^* is a lower bound for the number of characters in C compatible on an optimal refinement of T. If $C' \subseteq C^*$ is a maximum cardinality subset of compatible characters, then the set $C' \cup C(T)$ is a set of pairwise compatible characters, and hence it is *setwise* compatible by Lemma 1. Furthermore, there is a tree T' defined by $C(T') = C(T) \cup C'$. T' is a refinement of T and every character in $C(T) \cup C'$ is compatible on T'. Consequently, the size of a maximum compatible set of characters in C^* is a lower bound on the number of characters compatible on an optimal refinement of T.

Now, suppose T' is an optimal refinement of T, and that $C' \subset C$ is the set of characters compatible on T'. By Observation 1, it follows that $C' \subseteq C^*$, and that C' is a set of compatible characters. Hence, the number of characters from C compatible on an optimal refinement of T is a lower bound on the size of a maximum compatible subset of C^*. Since both quantities are lower bounds of each other, they are identical. ∎

To summarize, when the set C only consists of binary characters, then in $O(kn)$ preprocessing, we can reduce the OTR-compatibility problem to the Max Compatibility problem on a derived set of binary characters.

4.3. Maximum compatibility and OTR-compatibility for binary characters

The result in Theorem 16 indicates that the OTR-compatibility problem is hard, even to approximate, for bounded degree trees when the set of characters is allowed to have an unbounded number of character states. The exact algorithm for OTR-compatibility for $r \geq 3$ state characters was not potentially fast enough to be of general use except under unusual circumstances. However, when $r = 2$, then many problems related to maximum compatibility become tractable. In this section we describe several of these results.

We now describe how to approximate the *imperfection* of an optimal solution to the Max Compatibility problem for binary characters.

4.3.1. Approximating the imperfection of binary characters.
Let C be a set of characters defined on a set S of taxa. We begin with some definitions.

Given C and S, we will define the **incompatibility graph**, G_C, as follows. The nodes of G_C are the elements of C, and the edge (i, j) is in $E(G_C)$ if and only if the characters i and j are incompatible. An *independent set* in a graph $G = (V, E)$ is a subset $V_0 \subseteq V$ such that for all nodes v, w in $V, (v, w) \notin E$. A *vertex cover* in G is a subset $V_1 \subseteq V$ such that for all edges $(a, b) \in E$, at least one of a or b is in V_1. Consequently, the complement of an independent set is a vertex cover, and vice-versa. Since pairwise compatibility of binary characters suffices for setwise compatibility by Lemma 1, the maximum independent set of G_C corresponds to the maximum compatible subset of C. Consequently, if we can find a maximum independent set in G_C we will have solved the Max Compatibility problem.

The maximum independent set problem is however very hard to solve (Bellare and Sudan), but in the cases that we are interested in, the imperfection is generally quite low (see the discussion in the introduction). Consequently, the size of the maximum independent set will in general be quite large, and consequently the size of the minimum vertex cover (the complement to the maximum independent set) will be quite small. There is also a very simple and fast 2-approximation algorithm for finding the minimum vertex cover (see

Corman *et al.* (1990)) which we will use to obtain a 2-approximation algorithm for the imperfection t. We now briefly describe the 2-approximation algorithm.

A greedy 2-approximation algorithm for the minimum vertex cover. A *matching* in a graph $G = (V, E)$ is a subset $E_0 \subseteq E$ such that no two edges in E_0 share an endpoint. It follows that if V_1 is a vertex cover E_0 is a matching, then V_1 contains at least one endpoint from each of the edges in E_0. Consequently, the size of a minimum vertex cover is bounded from below by the size of *any* matching. On the other hand, if E_0 is a *maximal matching* (meaning there does not exist any matching E_1 which properly contains E_0), then the set V^* which contains *both* endpoints of every edge is a vertex cover (since its complement is an independent set), and its cardinality is bounded from above by twice the size of a minimum vertex cover. Obtaining a maximal matching in a graph is a trivially easy problem—simply remove an edge and both its endpoints from the graph, until there are no remaining edges. The edges that have been removed constitute a maximal matching. This is an $O(|V| + |E|)$ algorithm, and it produces a 2-approximate solution to the minimum vertex cover. Consequently, the following holds:

Theorem 12.   *The imperfection of a set of binary characters can be 2-approximated in $O(k^2 n)$ time.*

Proof.   We begin by constructing the incompatibility graph G_C; this takes $O(k^2 n)$ time since pairwise compatibility of two characters takes $O(n)$ time. Then we use the 2-approximation algorithm for the minimum vertex cover (Corman *et al.*, 1990), which takes $O(k^2)$ time. The complement of the vertex cover is then an independent set in G_C and hence denotes a set C' of pairwise (and hence setwise) compatible characters, with $|C'| \geq k - 2t$.   ∎

4.3.2. Solving Max Compatibility of binary characters.   We now describe a simple algorithm to solve the maximum compatibility problem for binary characters.

We begin by constructing G_C, the incompatibility graph. We then compute a 2-approximation to the imperfection using the above algorithm. This explicitly gives us a vertex cover V_0 of size at most $2t$, and furthermore V_0 consists of both endpoints of $p \leq t$ edges (this follows from the greedy algorithm used to approximate the vertex cover). The complement of V_0 is an independent set, V_1. Now, consider a maximum independent set I in G_C. Let $A_I = I \cap V_0$. Note that although many different I define the same A_I, for each set $A \subseteq V_0$ there is a unique maximum such independent set $I(A)$ such that $I(A) \cap V_0 = A$. This $I(A)$ is defined by $I(A) = A \cup (V_1 - \Gamma(A))$, where $\Gamma(A)$ denotes the neighbors of A. Consequently, to find a maximum independent set, we examine all possible sets $A \subseteq V_0$ and then in $O(k^2)$ time per set A, we compute $I(A)$. At the end, we return a largest such set $I(A)$. All we need to do is to compute how many different sets A there are and show we can list them efficiently. There are only 3^p sets A, since each A contains at most one of the endpoints of each of the p edges used to construct V_0. We do not have to keep track of more than one set A at a time, so that this does not incur a cost in terms of space.

This technique, which is similar to the one in Papadimitriou and Yannakakis (1993), provides an $O(k^2 n + 3^t k^2)$ solution to the maximum compatibility problem. However, a faster algorithm is given in Balasubramanian *et al.* (1998) which computes a vertex cover of size t in $O(kt + (4/3)^t t^2)$.

Consequently, we have the following:

Theorem 13.   *The maximum compatibility problem can be solved in $O(k^2 n + kt + (4/3)^t t^2)$ time, where there are k binary characters defined on n taxa, and having imperfection t.*

We now address the final result regarding character compatibility for binary characters, which is useful when t (the imperfection) is not small enough to make the exact algorithm feasible, but d (the maximum degree) is small enough to use another technique. Once again, we note that no character which is incompatible with T will be compatible on any refinement of T, and hence we preprocess the data by deleting all such characters.

Let V be the set of internal nodes in T. Consider the set E' of edges of T where each $e_i = (u, v) \in E'$ is such that neither u nor v is a leaf. Subdivide each edge $e_i = (u, v) \in E'$, by adding a new node w_i on e_i (i.e., add edges (u, w_i), (w_i, v) and remove (u, v)). Let T' be the resulting labelled tree. Recall that each leaf in T (and hence in T') is a species from S and hence each leaf is assigned a state for each character. We now show how to *extend* the characters to take values on the added nodes, w_i. The deletion of the node w_i from T' creates two subtrees. For each character c, if there are leaves in the two subtrees having the same state for c, then we assign $c(w_i)$ to that shared state. If there are two different states which are shared between the two

subtrees, then c is not compatible on any refinement of T, and we would have deleted c from the set. The final possibility is that no state is shared between the two subtrees, in which case we assign a new state to that node. In this way, we have set states for each of the characters on each of the newly introduced nodes, w_i. We now show how this permits us to solve $O(n)$ independent Max Compatibility instances.

Let v be one of the internal nodes in T and let S_v be $\{v\} \cup \Gamma(v)$ where $\Gamma(v)$ is the neighbors of v in the new tree, T'. It is then easy to see that for each character $c \in C$, there is at most one set S_v on which c is *not constant* (this is due to the fact that the characters are all binary, and all characters that are not compatible on any refinement of T have already been eliminated). For this reason, the imperfection of the set C (i.e., the number of characters which are incompatible on some fixed optimal refinement of T) is precisely the sum, over all internal nodes $v \in V(T)$, of the *imperfection* of C on S_v. This quantity can be estimated (either exactly or approximately) by using the appropriate Max Compatibility algorithm for binary characters.

We summarize our results on binary characters as follows:

Theorem 14. *The imperfection of the optimal refinement can be 2-approximated in $O(nk^2)$ time. The OTR-compatibility problem can be solved exactly in $O((4/3)^q nq^2 + k^2 n^2 + kqn)$ time, where q is the maximum imperfection on any subproblem S_v, and in $O(f(d)nk)$ time, where $f(d)$ is the number of rooted binary trees on d nodes.*

4.4. Max Compatibility and OTR-compatibility for $r \geq 3$

We can show that approximating OTR-compatibility for a *bounded* degree tree is as hard as approximating MAX Independent set on graphs (Arora *et al.*, 1992). We begin with a result from Arora *et al.* (1992) on maximum independent set.

Theorem 15. *(Arora et al., 1992) There is an $\epsilon > 0$ such that approximating MAX independent set within a factor n^ϵ is NP-hard, where n is the number of vertices in the input graph.*

Theorem 16. *Let T be a bounded degree tree with leaf set S (described by a set C of k characters). Then there is an $\epsilon > 0$ such that approximating OTR-compatibility within a factor k^ϵ is NP-hard.*

Proof. We will give a simple reduction from the Independent Set problem.

Let $G = (V, E)$ be the graph, with n nodes and m edges, in the instance I of the Independent Set problem. We will construct the instance I' to the OTR-compatibility problem as follows. The tree T is given in Figure 2. Note that it has degree bounded by 5.

T has m internal nodes, $u_1, u_2, \ldots, u_m$, where node u_i corresponds to edge $e_i \in E$. We will use s_i^1, s_i^2, s_i^3 to denote the three species adjacent to the internal node u_i in T. Let A denote the set of species. We will define n characters, $c_1, c_2, \ldots, c_n$, where the idea is that character c_j corresponds to node $v_j \in V$. We now describe how the character states are set for the species in A.

Let $e_i = (v_j, v_k)$ be an edge in E. Then, for all $l \notin \{j, k\}$, set $c_l(s_i^1) = c_l(s_i^2) = c_l(s_i^3) = l$. Also, set $c_j(s_i^1) = c_j(s_i^2) = k, c_j(s_i^3) = j$ and set $c_k(s_i^1) = c_k(s_i^3) = j, c_k(s_i^2) = k$.

We make the following easily proven claim.

Claim: The graph G has an independent set $\{v_{i_1}, v_{i_2}, \ldots, v_{i_p}\}$ if and only if the character set $\{c_{i_1}, c_{i_2}, \ldots, c_{i_p}\}$ can be made compatible on some refinement of T.

Because of the nature of the reduction (i.e., an independent set of size p corresponds to a refinement of T with p compatible characters and vice-versa), it can be seen that approximating MAX OTR-compatibility for bounded degree trees is at least as hard as approximating MAX independent set. ■
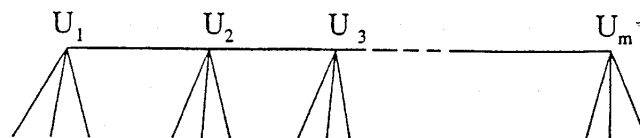

FIG. 2. Topology of tree T.

We describe an algorithm to solve the OTR-compatibility problem that runs in time exponential in the incompatibility score t and r. We make use of the $O(2^{2r}nk^2)$ time algorithm of Kannan and Warnow (1995) for finding perfect phylogenies.

Examine all subsets of characters, from largest to smallest, and find the first such subset C_0 such that C_0 is compatible with T (that is, every character in C_0 is compatible on some refinement of T). This is equivalent to testing whether $C(T) \cup C_0$ has a perfect phylogeny, where $C(T)$ is the character encoding of T. By the above, this can be tested in $O(2^{2r}k^2n)$ time for each set C_0, for a total of $O(2^{2r}k^{t+2}n)$ time. Thus, we have:

Theorem 17. *OTR-compatibility can be solved in $O(2^{2r}k^{t+2}n)$ time.*

Algorithm:

Begin
For each $C_0 \subseteq C$, where $|C_0| = k - t$, do

- For each internal $v \in T$ which has degree greater than 3, do:
    1. Let $\Gamma(v) = \Gamma_1(v) \cup \Gamma_2(v)$ where $\Gamma_1(v)$ consists of all the neighbors of v which are leaves and $\Gamma_2(v)$ consists of all the nonleaf 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 in C_0 convex (each character must contain every state that appears on both sides of w_j).
    2. If some new node has a character that requires more than one character state for that character to be convex, then 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 the algorithm of Kannan and Warnow (1995) to determine if a perfect phylogeny exists for (S_v, C_0). If any (S_v, C_0) fails to have a perfect phylogeny then RETURN(No), else RETURN(yes).

Return (Yes) if and only if some C_0 returns (Yes).
End

The run time of the above algorithm is $O(2^{2r}k^{t+2}n)$ since there are $O(k^t)$ subsets of size $k - t$, and for each subset C_0 of this size, we apply the perfect phylogeny algorithm to each (S_v, C_0).

5. DISCUSSION AND CONCLUSION

In this paper we have discussed the computational complexity of solving the Optimal Tree Refinement problem with respect to two optimization criteria, *parsimony* and *compatibility*. This approach to tree reconstruction is motivated by the empirical observation (see Huson et al. (1998) and Huson et al. (1998a)) that certain methods are likely to produce trees that are either contractions or close to being contractions of the true tree. This suggests that a two-phase process of evolutionary tree reconstruction (first obtain a contraction of the tree, and then refine the tree optimally) may produce more accurate topologies than existing methods. This approach is especially appropriate when the optimization problem is hard to solve and the two optimization criteria we have examined (maximum parsimony and maximum compatibility) are both hard to solve. Although we show that the OTR-parsimony and OTR-compatibility problems are also generally hard to solve, the exact algorithms we presented may be sufficiently fast to be useful in cases where the tree obtained during the first phase has sufficiently low vertex degree.

It is worth noting that the approximation algorithms we provided for the OTR-parsimony and OTR-compatibility problems can be used to obtain *bounds* on the scores achieved by the optimal solutions to these problems, as Gusfield and Wang as noted in Gusfield and Wang (1998). Thus, these fast algorithms can be used to obtain better evaluations of trees obtained heuristically and for which no guaranteed performance ratio can be inferred.

Finally, we have not addressed the problem of optimally refining a tree with respect to maximum likelihood estimation, which is potentially a very powerful tool, and we have also not addressed the use of heuristics for solving OTR problems in general. We leave these problems to later work.

ACKNOWLEDGMENTS

REFERENCES

Agarwala, R., and Fernández-Baca, D. 1996. A Polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. on Computing* 23(6), 1216–1224.

Agarwala, R., and Fernández-Baca, D. 1994. Fast and simple algorithms for perfect phylogeny and triangulating colored graphs. *DIMACS Tech Report*, 94–51.

Ambainis, A., Desper, R., Farach, M., and Kannan, S. 1997. Nearly tight bounds on the learnability of evolution. *IEEE Symposium on Foundations of Computer Science.*

Arora, S., Lund, C., Motwani, R., Sudan, M., and Seegedy, M. 1992. Proof verification and intractability of approximation problems. *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 13–22.

Atteson, K. 1997. The performance of neighbor-joining algorithms of phylogeny reconstruction. *Computing and Combinatorics, Third Annual International Conference, COCOON '97*, Shanghai, China, August 1997 *Proceedings. Lecture Notes in Computer Science*, 1276, Jiang, T., and Lee, D.T. (Eds.). Springer-Verlag, Berlin, 1997, 101–110.

Balasubramanian, R., Fellows, M.R., and Raman, V. 1998. An improved fixed parameter algorithm for vertex cover. *Inf. Proc. Lett.* (in press).

Bellare, M., and Sudan, M. Improved non-approximability results. *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, Montreal, ACM, 184–193.

Berry, V. 1997. Méthodes et algorithmes pour reconstruire les arbres de l'Évolution. Ph.D. thesis, Université Montpellier III.

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

Bonet, M., Phillips, C.A., Warnow, T., and Yooseph, S. 1996. Constructing evolutionary trees in the presence of polymorphic characters. *ACM Symposium on the Theory of Computing*, 1996. To appear in SIAM J. Computing.

Chang, J.T. 1996. Inconsistency of evolutionary tree topology reconstruction methods when substitution rates vary across characters. *Math. Biosci.* 189–215.

Cormen, T., Leiserson, C., and Rivest, R. 1990. *Introduction to Algorithms*, MIT Press, 1990.

Day, W.H.E., and Sankoff, D. 1986. Computational complexity of inferring phylogenies by compatibility. *Syst. Zool.* 35(2), 224–229.

Day, W.H.E. 1983. Computationally difficult parsimony problems in phylogenetic systematics. *J. of Theo. Bio.* 103, 429–438.

Erdös, P.L., Steel, M., Székeley, L., and Warnow, T. 1997. Inferring big trees from short sequences. *Proceedings of International Congress on Automata, Languages, and Programming.*

Farach, M., and Kannan, S. 1996. Efficient algorithms for inverting evolution. *Proceedings of the ACM Symposium on the Foundations of Computer Science*, 230–236.

Felsenstein, J. 1978. Cases in which parsimony or compatibility methods will be positively misleading. *Syst. Zoology* 27, 401–410.

Felsenstein, J. 1982. Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology* 57(4).

Fitch, W. 1971. Toward defining the course of evolution: Minimum change for a specified tree topology. *Syst. Zool.* 20, 406–416.

Foulds, L.R., and Graham, R.L. 1982. The Steiner Problem in Phylogeny is NP-Complete, *Adv. Appl. Math.* 3, 43–49.

Garey, M.R., and Johnson, D.S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.

Gusfield, D. 1991. Efficient algorithms for inferring evolutionary trees, *Networks* 21, 19–28.

Gusfield, D., and Wang, L. 1998. New uses for lifted alignments, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.*

Hartigan, J.A. 1973. Minimum mutation fits to a given tree, *Biometrics* 29, 53–65.

Hillis, D. 1997. Inferring complex phylogenies, *Nature* 383, 130–131.

Huson, D., Nettles, S., Parida, L., Warnow, T., and Yooseph, S. 1998. The Disk-Covering Method for Tree Reconstruction, *Proceedings of the Workshop on Algorithms and Experiments (ALEX)*, Trento, Italy.

Huson, D., Nettles, S., Rice, K., Warnow, T., and Yooseph. S. 1998a. Hybrid tree reconstruction methods. *2nd Workshop on Algorithm Engineering (WAE'98)*, Saarbrucken, Germany (in press).

Kannan, S., and Warnow, T. 1995. Finding and enumerating all perfect phylogenies when the number of states is fixed. *SIAM J. on Computing* (to appear). A preliminary version appeared in *SODA '95*.

Kannan, S., and Warnow, T. 1990, 1994. Inferring Evolutionary History from DNA Sequences. *SIAM J. on Computing* 23(4), 713–737. (A preliminary version of this paper appeared at *FOCS 1990.*)

Kim, J. General inconsistency conditions for maximum parsimony: Effects of branch lengths and increasing numbers of taxa. *Syst. Biol.* 45(3), 363–374.

McMorris, F.R., Warnow. T., and Wimer, T. 1993. Triangulating vertex colored graphs. *SIAM Journal of Discrete Mathematics* 7(2), 296–306.

Papadimitriou, C.H.. and Yannakakis, M. 1993. On limited nondeterminism and the complexity of the V-C dimension. *Proceedings of the 8th Annual Conference on Structure in Complexity Theory (SCTC '93)*, 12–18, IEEE Computer Society Press.

Rice, K., Donoghue. M., and Olmstead, R. 1997. Analyzing large data sets: rbcL 500 revisited. *Systematic Biology* 554–563, 46(3).

Rice, K., and Warnow, T. 1997. Parsimony is Hard to Beat!, *Computing and Combinatorics, Third Annual International Conference, COCOON '97*, Shanghai, China, August 1997 Proceedings. Lecture Notes in Computer Science, 1276, Jiang, T., and Lee. B.T. (Eds.). Springer-Verlag, Berlin, 1997, 124–133.

Saitou, N., and Nei. M. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* 4, 406–425.

Steel, M.A. 1992. The complexity of reconstructing trees from qualitative characters and subtrees. *J. of Classification* 9, 91–116.

Steel, M.A. 1994. Recovering a tree from the leaf colourations it generates under a Markov model. *Appl. Math. Lett.* 7, 19–24.

Steel. M.A., Székely, L.A., and Hendy, M.D. 1994. Reconstructing trees when sequence sites evolve at variable rates. *J. Computational Biology* 1(2), 153–163.

Tuffley, C., and Steel. M.A. 1997. Links between maximum likelihood and maximum parsimony under a simple model of site substitution. *Bulletin of Math. Bio.* 59(3), 581–607.

Wang, L., Jiang. T., and Lawler. E. 1996. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica* 16, 302–315.

Wang, L., and Gusfield. D. 1997. Improved approximation algorithms for tree alignment. *J. of Algorithms* 25(2), 255–273.

Warnow, T. 1997. Mathematical approaches to comparative linguistics. *Proceedings of the National Academy of Sciences* 94, 6585–6590.

Warnow, T. 1996. Some combinatorial problems in phylogenetics. Invited to appear in the *proceedings of the International Colloquium on Combinatorics and Graph Theory*, Balatonlelle, Hungary, July 15–20, 1996, eds. Gyárfás. A.. Lovász. L., Székely, L.A. in a forthcoming volume of *Bolyai Society Mathematical Studies*.

Warnow, T. Constructing phylogenetic trees efficiently using compatibility criteria. *New Zealand Journal of Botany* 31, 239–248.

Warnow, T., Ringe, D.. and Taylor, A. 1996. Reconstructing the evolutionary history of natural languages. *Proceedings of ACM-SIAM*, SODA, 314–322.

Address reprint requests to:
Shibu Yooseph
Department of Mathematics
University of Southern California
1042 West 36th Place
DRB 155
Los Angeles, CA 90089-1113

yooseph@dimacs.rutgers.edu