JOURNAL OF ALGORITHMS **21,** 26–50 (1996)
ARTICLE NO. 0035

Determining the Evolutionary Tree Using Experiments

Sampath K. Kannan*

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Eugene L. Lawler†

Department of Computer Science, University of California, Berkeley, Berkeley, California 94720

and

Tandy J. Warnow‡

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Received November 4, 1991

DEDICATED IN FOND MEMORY TO EUGENE LAWLER BY HIS COAUTHORS

Evolutionary trees, also known as *phylogenetic trees*, are rooted vertex-labeled trees which describe the evolution of a species set S from a common ancestor. The determination of evolutionary trees is a fundamental problem in computational evolutionary biology, and has been studied in great depth. In this paper, we present a new model of computation which assumes that it is possible to determine the true evolutionary tree for each three species, perhaps through the use of Ahlquist–Sibley experimental techniques. We present tight upper and lower bounds for constructing evolutionary trees using experiments. © 1996 Academic Press, Inc.

* Supported by NSF Grant CCR 91-08969.
† Supported in part by NSF Grant IRI 89-02813.
‡ Supported in part by NSF Grants IRI 89-02813 and CCR-9457800, and by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

0196-6774/96 $18.00
Copyright © 1996 by Academic Press, Inc.
All rights of reproduction in any form reserved.

1. MODEL OF COMPUTATION

An *evolutionary tree* for a species set S is a rooted tree in which the leaves represent the species in S, and the internal nodes represent common ancestors. Determining evolutionary trees (also known in the biological literature as *phylogenetic trees* or *phylogenies*) is a fundamental problem in computational evolutionary biology, and one which has been studied in depth by many authors (for an overview of the subject, see [7] or [4]). One of the standard ways the input to the problem is given is as a distance matrix [6, 14]. In this case, a matrix M is given such that M_{ij} is the observed distance between species s_i and s_j, computed in some manner (such as by aligning and comparing DNA sequences for the species). The objective is to construct (if possible) an edge weighted tree T with leaves labeled by $s_1, s_2, \ldots, s_n$, so that the distance $d^T(i, j)$ between s_i and s_j is exactly M_{ij}. In such a case, the matrix M is said to be *additive*. Sometimes the observed distance M_{ij} is proportional to the time that has elapsed when
the species s_i and s_j have diverged from a common ancestor; in this case, the tree T can be rooted (at the common ancestor of all the species) so that the distance from the root to each leaf is identical. Such a tree or matrix is then called *ultrametric*. Note that every ultrametric matrix is by definition additive. Constructing trees from additive matrices can be done efficiently (see [2, 8, 9, 17, 18], for example); however, when the matrix is not additive, then the objective is to find a tree minimizing some objective criterion. Unfortunately, almost all optimization criteria result in *NP-hard* problems, whether one wishes to construct additive or ultrametric trees [3]. One variation that can be solved in polynomial time for constructing optimal ultrametric trees was found by Farach *et al*. [5].

Motivated by the importance of the problem, we want to determine whether there is a larger class of distance matrices for which meaningful evolutionary trees (which might not be constrained to be ultrametric, as in the algorithm of [5]) could be found efficiently. We therefore define the following class of distance matrices, which we call *noisy-ultrametric*. Let $lca_T(a, b)$ denote the least common ancestor in T of nodes a and b. We will say that a matrix M is *noisy-ultrametric* if there is a rooted tree T such that for all a, b, c, $M_{ab} < min\{M_{ac}, M_{bc}\}$ if and only $lca_T(a, b)$ is below $lca_T(b, c) = lca_T(b, c) = lca_T(a, c)$. Thus, the rooted topology of the subtree on a, b, c is as in Fig. 1. Note that a noisy-ultrametric matrix M may not be additive, so that while the topology of the tree T may be determined uniquely by the matrix M, it may be impossible to weight the tree T (or any tree) so that $d^T(i, j) = M_{ij}$ for all i, j. Thus, every ultrametric matrix is noisy-ultrametric, but a noisy-ultrametric matrix may not even be

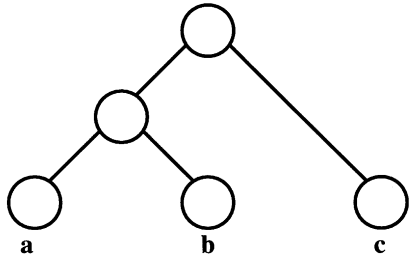

FIG. 1. Type 1 Responses.

additive. However, if the matrix is also additive, then the (unweighted) tree T determined by the noisy-ultrametricity rule can be edge-weighted so that $d^T(i, j) = M_{ij}$ as well.

In this paper, we will present algorithms to construct evolutionary trees when it is possible to perform experiments (perhaps by examining a noisy-ultrametric matrix) which can determine for any three species a, b, and c how they are related in the evolutionary tree. There are two possible types of outcomes to the experiment:

1. The experiment determines that the phylogeny for the species set $\{a, b, c\}$ is given by the tree in Fig. 1. Note that this response implies that species a and b are more closely related in time than are the other two pairs. We represent this outcome using the notation $((a, b), c)$.

2. The experiment determines that the phylogeny for the species set $\{a, b, c\}$ is given by the tree in Fig. 2. This response implies that the least common ancestor of each pair is the same. We represent this outcome using the notation (a, b, c).

When the phylogeny for S is a rooted binary tree, then the experiment will always return answers of the first type.

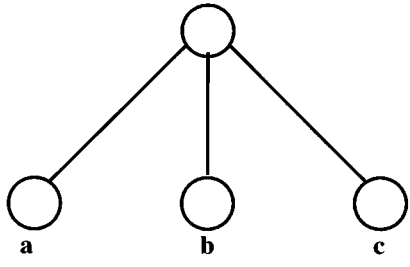

FIG. 2. Type 2 Responses.

As we have said, if the distance matrix is noisy-ultrametric, then these experiments can be implemented naturally. Another way in which these experiments can be implemented is through the use of Ahlquist and Sibley experimental techniques [13]. In their method, double strands of DNA from two species x and y are separated, then placed together and cooled, so that their strands bind together. These are then heated, and the temperature at which the strands separate is observed. The theory is that the more closely related the two species are, the higher the temperature must be to separate them. This technique allows one to determine any specific entry of the distance matrix and in particular decide which pair of species in a set of three species is most closely related.

It is important to note that our model does not require a distance matrix in order to determine the phylogenetic tree for the species. Other algorithms have been developed which achieve efficient running times when the input to the problem is an additive distance matrix; however, these algorithms explicitly exploit the numeric values of the distances between species. Thus, while an experiment in our model could be implemented through the use of a distance matrix, it is theoretically possible to determine the phylogeny on any three species without actually determining a distance matrix which would apply to all the species at once, and in fact without determining any absolute distances at all.

In Section 2, we present some preliminary material and definitions, as well as presenting some lower bounds for constructing phylogenetic trees from experiments. We then consider the problem of constructing binary phylogenetic trees, so that the oracle is restricted to answering queries with responses of type 1. We give three different algorithms to solve the problem. In Section 3, we present the first algorithm, which has a running time of $O(n^2)$ and performs at most $n \log_2 n$ experiments. In Section 4, we present a second algorithm which runs in $O(n \log^2 n)$ and performs at most $n \log_{3/2} n$ experiments. Finally, we present in Section 5 a third algorithm which achieves an optimal running time of $O(n \log n)$ at the cost of increasing the number of experiments to $4n \log_2 n$. All three algorithms are interesting because in the practical application being considered, the experiments are likely to be far more expensive than the computational costs. The choice between these algorithms depends on the relative costs of experimentation and computation.

In Section 6, we present algorithms for constructing trees when we do not presume the tree is binary. For the case of bounded degree trees, we present an $O(n^2)$ algorithm which performs $O(kn \log n)$ experiments where k is the bound on the degree; this algorithm is based upon the $O(n^2)$ algorithm of Section 3. For the general case (where the tree has unbounded degree), we present an $O(n^2)$ algorithm modeled after binary insertion sort which performs $O(n^2)$ experiments. We also show that this is

optimal. In Section 7, we present an algorithm for a related graph-theoretic problem. Finally, we discuss related open problems in Section 8.

2. PRELIMINARIES

In this section, we formally define the phylogenetic tree inference problem solved in this paper and state a few properties of an ultrametric in the context of this problem. The problem of determining a phylogenetic tree using experiments can be used to solve a purely graph-theoretic problem and this reduction is also described in this section.

2.1. *Ultrametric Inequality*

A distance function d satisfies the ultrametric inequality if for any three points i, j, k,

$$d_{ij} \leq \max(d_{jk}, d_{ki}).$$

Thus, an *ultrametric tree* is an edge-weighted tree such that the distance function on interleaf distances is ultrametric. Distances satisfying the ultrametric inequality arise naturally in a number of other contexts as well:

(i) In a weighted spanning tree with the distance function d_{ij} defined as the maximum weight of an edge along the path from i to j.

(ii) In an edge-weighted graph, with d_{ij} defined to be the maximum weight of an (i, j) separating cut.

(iii) In a heap-ordered tree, with the weight of a node at least as great as the weights of its children, with d_{ij} defined to be the weight of the least common ancestor of any two leaves i, j.

(iv) In a p-adic number system where the distance between i and j is the distance given by the p-adic metric.

We will define a *strict ultrametric* to be an ultrametric d such that, for any three indices i, j, k, the distances d_{ij}, d_{ik}, d_{jk} are not all equal. Note that a strict ultrametric leads to a binary evolutionary tree in our model.

The problem of determining a phylogenetic tree has two variations depending on whether we are given ultrametric distances or just the ability to perform experiments on triples of species to determine the closest pair (such as when the input is a noisy ultrametric matrix). In the first case, we require the output to be an edge-weighted tree, T, such that the given distance between any two leaves i and j is the length of the path from i to j in T. In the second case, even if actual distances are available, they may not be additive. Thus finding an edge-weighted tree such that the path distance in the tree is the same as the given distance for all interleaf pairs may be impossible. Thus, in the second case, our goal is only to infer the

topology of the evolutionary tree. We state these two cases as the two problems below, although all the algorithms we design work for both cases. By designing algorithms in this way, we can handle small errors in the values of interspecies distances even when these are given.

Problem 1a. Given the ability to find the distance between any two species (assuming that the set of distances define an ultrametric), construct an edge-weighted tree T such that for all pairs of species i and j, the length of the path from i to j in T is equal to the specified distance between i and j.

Algorithms to solve Problem 1a are given in Section 7. These algorithms yield a constructive proof of the following fact.

Fact 1. Given any set of interspecies distances that are ultrametric, we can always find a weighted phylogenetic tree realizing the specified distances.

Problem 1b. Given the ability to perform an experiment on any three species to decide which two have the most recent common ancestor, construct a tree T whose topology is consistent with all experiments (assuming that there is such a tree).

The solution to Problem 1a can be used to solve the following graph-theoretic problem.

Problem 2. Given a weighted complete graph G with weights (w_{ij} on edge (i, j)) satisfying the *strict* ultrametric inequality, find a minimum spanning tree in the graph in time $O(n \log n)$.

To solve this problem, we treat the vertices of the weighted complete graph as species, an the weight of the edge between two vertices as the distance between the two species. We then find a phylogenetic tree T using the $O(n \log n)$ algorithm of Section 5. By Fact 1, this algorithm will be able to find such a tree whenever the interspecies distances are ultrametric. T will have edge weights such that the distance in the tree between two leaves equals the weight of the edge between the two species in the complete graph. Our algorithm is then as follows: Let x and y be the left and right children of the root of T. Let T_x and T_y be the subtrees of T rooted at x and y respectively and let S_x and S_y be the leaf-sets of T_x and T_y, respectively. Recursively, find minimum spanning trees T_x^* and T_y^* in the subgraphs of G induced by S_x and S_y, respectively. Connect T_x^* and T_y^* by adding an edge between any two vertices v and w, with $v \in S_x$ and $w \in S_y$. Call the tree we create T^*.

LEMMA 1. *T^* is a minimum spanning tree.*

Proof. First note that every edge from a vertex in S_x to a vertex in S_y has the same weight, and this weight is the largest of the edge weights.

Now suppose that the true minimum spanning tree includes two such edges (x_1, y_1) and (x_2, y_2) where $x_1, x_2 \in S_x$ and $y_1, y_2 \in S_y$. If all four nodes x_1, x_2, y_1, y_2 are distinct, then we have a 4-cycle x_1, y_1, y_2, x_2. Edges (x_1, y_1) and (x_2, y_2) are of greater weight than (x_1, x_2) and (y_1, y_2). Hence, any minimum spanning tree cannot contain both (x_1, y_1) and (x_2, y_2), since any minimum spanning tree excludes a maximum weight edge from each cycle in the graph. Thus, there is exactly one edge between T_x^* and T_y^* in any minimum spanning tree. (If $x_1 = x_2$ or $y_1 = y_2$, a similar proof goes through.) Using the inductive hypothesis that T_x^* and T_y^* are minimum spanning trees for their vertex sets, we find that T^* is a minimum spanning tree. ∎

2.2. *Lower Bounds for Constructing Phylogenetic Trees*

We will prove two lower bounds in this section; the first is the information-theoretic lower bound, showing that even when the trees are constrained to be binary, we will require $\Omega(n \log n)$ experiments to find the tree. The second lower bound applies when the trees can be general (i.e., of unrestricted degree). In this case, we require $\Omega(n^2)$ in a worst case.

2.2.1. The Information-Theoretic Lower Bound

We begin by showing that the number t_n of distinct evolutionary trees with n leaves is $2^{\Omega(n \log n)}$. Since each experiment performed yields one of three possible results, we get a lower bound on the number of experiments, which is $\log_3(t_n)$. Thus the total number of experiments that need to be done is $\Omega(n \log n)$.

To count the number of distinct evolutionary trees with n leaves, we set up a recurrence relation, where t_n is the number of evolutionary trees on n leaves. Let $\mathcal{T}$ be an evolutionary tree with n leaves. We can create an evolutionary tree with $n + 1$ leaves by adding another leaf in one of the following two ways:

1. Choosing one of the edges of $\mathcal{T}$ to contain the parent vertex of the new leaf, s_{n+1}. This will require inserting a vertex into that edge, and letting s_{n+1} be a child of that vertex.
2. Creating a root r' for the new tree, and letting the two children of r' be s_{n+1} and r, the root of $\mathcal{T}$.

If we let t_n denote the number of evolutionary trees with n leaves, then since each way of inserting s_{n+1} into an n-leaf tree produces a distinct $(n + 1)$-leaf tree, we have the recurrence relation: $t_{n+1} = (2n - 1)t_n$, with

$t_2 = 1$. Thus,

$$t_n = 1 \cdot 3 \cdot \cdots \cdot (2n-3) = \frac{(2n-2)!}{(n-1)!2^{n-1}}.$$

2.2.2. *The $\Omega(n^2)$ Lower Bound*

Culberson and Rudnicki [2] proved the following.

LEMMA 2. *Every algorithm to construct trees using ultrametric distance matrices must examine every entry of the distance matrix, in a worst case.*

We can now prove the following theorem.

THEOREM 1. *Every deterministic algorithm to construct phylogenetic trees using oracle queries must perform $\Omega(n^2)$ queries in a worst case, if both types of oracle responses are permitted.*

Proof. An experiment on species a, b and c could be implemented using ultrametric distance matrices by performing five steps: look up three entries in the distance matrix, and perform two comparisons. Suppose we had an algorithm which performed at most $g(n)$ oracle queries to construct an evolutionary tree. We could modify it to produce an algorithm which examined at most $3g(n)$ matrix entries to construct an *unweighted* evolutionary tree from an ultrametric distance matrix. We can then determine the edge weights in the tree by looking at only n additional matrix entries. Thus, we could construct the edge-weighted tree, examining at most $3g(n) + n$ matrix entries. Culberson and Rudnicki's lower bound then shows that $g(n) \in \Omega(n^2)$. ∎

2.3. *Terminology*

We introduce some terminology here which will simplify our discussion of the algorithms.

For binary trees, all of our algorithms work by inserting each successive species into the tree formed by the previously inserted species. As the enumeration of binary trees indicates, the $(n+1)$th species, s_{n+1} can be inserted into the tree, T_n, formed by the first n species in one of $2n-1$ distinct ways. The parent p_{n+1} of s_{n+1} can be inserted into one of the $2n-2$ edges of T_n or it can be made the root of T_{n+1} with children s_{n+1} and the root of T_n. Note that these algorithms really decide where p_{n+1} should be inserted. Whether s_{n+1} should be made a left or right child of p_{n+1} is a choice that is entirely unconstrained. We will exploit this choice later.

If we delete any internal node v from a binary tree T, we create three subtrees: the left subtree, the right subtree, and the *upper* subtree. We will refer to the first two subtrees as *lower* subtrees. If v is a node of the evolutionary tree on species $s_1, s_2, \ldots, s_n$, we say that s_i lies *in a lower subtree of* v (or below v) if the leaf representing s_i is in the subtree rooted at v; otherwise, we say that s_i lies *in the upper subtree of* v (or above v). When we attempt to insert species s_k, if v is the least common ancestor of species s_i and s_j, then the experiment on s_i, s_j, and s_k tells us which of the three subtrees of $T - \{v\}$ will contain the leaf for species s_k. Since there is a possibility that the parent p_k of the new species s_k has to be inserted above the root of the subtree being considered, we always think of a subtree as including a "dangling edge" up from its root.

3. AN $O(n^2)$ ALGORITHM

The basis of the algorithm is the following lemma.

LEMMA 3. *In a tree T with n leaves there is an internal node, v, such that each of the lower subtrees of* $T - \{v\}$ *has at most* $n/2$ *leaves and the upper subtree has less than* $n/2$ *leaves.*

Proof. Walk down the tree from the root always walking to the child with the greatest number of leaves in its subtree. The first node, v, all of whose children have $\leq n/2$ leaves, is the required node. By construction v has more than $n/2$ leaves in its subtree, and hence there are fewer than $n/2$ leaves outside the subtree rooted at v. Also by construction none of v's children have more than $n/2$ leaves in their subtree. ∎

We will refer to this node v as a *weight center* of the tree. The algorithm is modeled after binary insert sort. It starts off with the first two species in a binary tree and successively inserts the ith species s_i into the tree T_{i-1} for the first $i - 1$ species.

We will now describe precisely how the ith stage is handled. Each internal node v of T_{i-1} is associated with a pair of species, v_l, v_r, with v_l occupying a leaf in the left subtree of v, and v_r occupying a leaf in the right subtree of v. The least common ancestor of v_l and v_r will therefore be v. Each time we add a species s_i to the tree we also add an internal node, p, which is the parent of s_i. We can then set the pair associated with p in the obvious way. For example, if s_i is the left child of p, then we will choose s_i as one member of the pair, and draw the other member for the pair from one of the entries of the pair associated with the right child of p. In this manner, each internal node will always have a pair of species associated with it, whose least common ancestor is the node itself.

To place the species s_i into the tree T_{i-1}, the algorithm locates the weight center, v, of the tree and notes the pair of leaves (a, b) associated with v. It then performs an experiment involving a, b, and s_i in order to determine in which of the three subtrees of $T_{i-1} - \{v\}$ s_i should be placed. Thus, we can reduce to a smaller tree, containing at most $n/2$ leaves. (Note that if the tree to be worked on is the upper subtree, then the entire subtree rooted at v can be replaced by the single node, v, making the number of leaves at most $n/2$.) In each of these cases, it recursively works within the smaller tree and places s_i in the right place.

It is clear that inserting the ith species takes at most $\lceil \log i \rceil$ experiments since each experiment performed eliminates half the species. Thus the total number of experiments performed is asymptotically equal to $n \log_2 n$. Once we have narrowed down the number of nodes down to 1, there is only one $(2 \times 1 - 1 = 1)$ possible edge (the dangling edge up from this node) in which the parent of s_i can be inserted, and thus we have determined s_i's position. Finding the weight center in a binary tree with n leaves can be done in $O(n)$ steps. The algorithm that achieves this time bound follows the constructive procedure given in Lemma 3.

In the binary insertion algorithm detailed above, consider the insertion of species s_{i+1}. Finding the weight center of tree T_i takes ci steps, where c is some constant, since T_i has i leaves. Subsequently, we find a weight center of a tree with at most $i/2$ leaves and this takes $ci/2$ steps. Thus the number of steps for finding the sequence the weight centers is bounded by $ci \cdot \sum_{k=0}^{\infty} (\frac{1}{2})^k = 2ci$. Thus the sequence of weight centers for inserting the $(i + 1)$th species stage can be found in $O(i)$ steps. Therefore this algorithm uses $O(n^2)$ time overall, but only $n \log n$ experiments asymptotically.

4. AN $O(n \log^2 n)$ ALGORITHM

Suppose $\mathcal{T}_{i-1}$ is the phylogenetic tree on the first $i - 1$ species, and we wish to construct $\mathcal{T}_i$ by adding s_i in the appropriate position to $\mathcal{T}_{i-1}$. It is convenient to view the task of inserting s_i as consisting of two phases. Suppose we have a left-to-right ordering on $s_1, s_2, \ldots, s_{i-1}$ consistent with some planar embedding of $\mathcal{T}_{i-1}$. In the first phase, we insert s_i into the left-to-right ordering, and in the second phase we find the topology of $\mathcal{T}_i$. The first phase of this algorithm is also modeled after binary insert sort. However, it uses a splitting method that can be computed more easily than the weight center.

The following lemma describes what we can infer from the result of an experiment.

LEMMA 4. *Let a, b, and c be species in a set S. If the result of the experiment is $((a, b), c)$, then c cannot lie in the interval between a and b in the evolutionary tree for S.*

Proof. Without loss of generality, let us assume that in the evolutionary tree T for S, that a lies to the left of b in the left-to-right ordering on the leaves. The left-to-right ordering on the leaves is defined by the recursive rule: order the leaves of the left subtree, then order the leaves of the right subtree. We will set V to be the node in T which is the least common ancestor of a and b. It is clear from the recursive definition of the ordering on the leaves, that the set of leaves in subtree rooted at v occupies an interval I in the left-to-right order on the leaves, with interval $(a, b) \subset I$. Therefore, since the species c lies in the subtree above v, the correct position for c cannot be in (a, b). ▮

The above lemma immediately suggests a binary insert sort style algorithm. However, in order to keep the cost of algorithm down to $O(n \log^2 n)$, we will use *red–black* trees [15].

4.1. *The Data Structure*

The algorithm inserts the species s_i into the linear ordering of the species during the ith stage. We will use a red–black tree T_{i-1} with $i - 1$ leaves representing the first $i - 1$ species. The left-to-right ordering on the leaves in the red–black tree T_{i-1} is the same as the left-to-right ordering on the *true* phylogenetic tree $\mathcal{T}_{i-1}$ for the first $i - 1$ species. T_{i-1} will have depth at most $2\log(i - 1)$. Each of its internal nodes v will be labeled with a pair of numbers $(l(v), r(v))$, where $l(v)$ is the number of leaves in the left subtree, and $r(v)$ is the number of leaves in the right subtree. We will refer to $l(v)$ as the "*left label*" of v, and $r(v)$ as the "*right label*."

4.2. *Inserting s_i into T_{i-1}*

The algorithm we will describe iteratively inserts species into the red–black tree, so that the left-to-right ordering on the leaves in the red–black tree T_{i-1} is the same as the left-to-right ordering on the leaves in a tree phylogenetic tree $\mathcal{T}_{i-1}$ for the species. To do this, we will first make a copy T'_{i-1} of T_{i-1}, and repeatedly apply Lemma 4 to find the appropriate position for s_i in the left-to-right ordering on $s_1, s_2, \ldots, s_{i-1}$. In subsequent discussions of how we apply the previous lemma to these red–black trees, we will speak of *deleting* leaves in an interval within the tree, T'_{i-1}. This is effected without actually performing any deletions; rather, we reset the labels (within T'_{i-1}) at the internal nodes so that these

leaves are not counted as falling below any internal nodes. In this way, we can locate correctly the next experiment to be performed. At the end of the iteration, having found the correct location for the new leaf s_i, we add s_i to the red–black tree T_{i-1}, rebalance it and update the labels, and thus obtain T_i.

The initial cases are as follows: In the first two stages two species s_1 and s_2 are arbitrarily chosen and "inserted" in the order $[s_1, s_2]$. As a base case, consider the third stage where s_3 is inserted by performing the experiment involving s_1, s_2, and s_3. If the result of this experiment if $((s_1, s_3), s_2)$ or $(s_1, (s_2, s_3))$, then the linear order $[s_1, s_3, s_2]$ is chosen; otherwise the linear order $[s_1, s_2, s_3]$ is chosen. In each case, we had two choices for the position in which to insert s_3 and we arbitrarily chose one.

During the ith stage, the algorithm inserts s_i into the ordered list of the first $i - 1$ species. This stage begins (as noted before) by making a copy T'_{i-1} of T_{i-1}. We then located a pair of species, (a, b), which occur at the $(\lfloor i/3 \rfloor)$rd and $(\lfloor 2i/3 \rfloor)$rd positions respectively in the ordered list of species and perform the experiment (a, b, s_i). If the result of the experiment is $((a,b), s_i)$, then the entire interval of leaves between a and b is replaced by a single leaf a (by resetting the labels at the internal nodes). We will refer to a as the "representative" of the interval it replaces. Note that the interval between a and b may not contain all the leaves in the subtree rooted at $v = lca(a, b)$. However, this is not a problem since s_i will not be inserted between the representative a and any surviving leaf in the subtree rooted at v. If the result of the experiment is $((a, s_i), b)$ the interval to the right of b is replaced by a representative and if the result of the experiment is $(a, (s_i, b))$, the interval to the left of a is replaced by a representative. The result of each experiment also involves updating the various labels, so that $l(v)$ and $r(v)$ indicate properly the number of *uneliminated* leaves in the left and right subtrees, respectively, below v. These updates are described in Section 4.3.

At the end of the binary insert sort procedure, when we determine that s_i belongs between y and z in the left-to-right ordering on the leaves, we can add s_i to the red–black tree T_{i-1} as follows. Without loss of generality let $y < z$ in the left-to-right ordering. Then let e be the edge in T_{i-1} incident to the leaf for z. Subdivide the edge e, and let s_i's parent be the new vertex. Then let s_i be the left child of its parent, and let z be the right child. It can be seen that this will place s_i appropriately in the left-to-right ordering on the leaves. It is clear that we will need to adjust the labels for the internal nodes on the path from s_i toward the root, and that T_i will then need to be rebalanced, and again the labels will be adjusted appropriately. The details of how to achieve these updates efficiently are discussed in the next section.

4.3. *Updating the Balanced Binary Tree*

We will modify the labels only in T'_{i-1} while maintaining the topology. After we have determined the correct position for s_i within the left-to-right ordering, we will discard T'_{i-1}.

Thus, there are two types of modifications that the algorithm will perform.

1. During the ith stage, in which we are determining the position of the leaf for s_i in the tree T_{i-1}, we perform no more than $\lceil \log_{3/2}(i-1) \rceil$ experiments. Each time we perform an experiment we need to modify the labels of the nodes in T'_{i-1}.

2. Each time we add a new species to T_{i-1} and get T_i, we must adjust it, so that it remains balanced, and reset the labels.

The first type of modification, which occurs after we perform an experiment on species a, b, and s_i, causes us to delete an interval or to replace an interval by a single node. Both of these actions can be described as deleting the closed interval $[u, v]$, for some species u and v. That is, if we find that s_i does not fall in the interval $[a, b]$, we will replace the interval $[a, b]$ by the single leaf a. This amounts to "deleting" the interval $[a', b]$, where a' is the leaf immediately following a.

Consider the problem of updating the labels of T'_{i-1} after the deletion of the closed interval $[u, v]$, for some pair of species u, v. Let x represent the least common ancestor of u and v in T'_{i-1}. The modification of the labels of the vertices on the paths from u and v to the root depends on whether the vertex lies *above* x or *below* x. We work bottom-up from u to x. For any node w on the path from u to x, if u is in the left subtree of w, then the right label of w is set to 0. The left label of w is set to the sum of the labels of the left child of w. If u is in the right subtree, then the left label of w is unchanged and the right label of w is set to the sum of the labels of w's right child. We will adjust the labels of the vertices on the path between v and x similarly. We set the left label of x to be the sum of the labels of its left child, and the right label the sum of the labels of its right child. We then move up the path from x to the root r, adjusting the labels as we go, so that the right label of each vertex on the path is equal to the sum of the labels of its right children, while the left label is equal to the sum of the labels of its left children.

Placing s_i correctly in the left-to-right ordering of the leaves of T_{i-1} involves up to $\lceil \log_{3/2}(i-1) \rceil$ experiments. With each experiment, we modify the labels of T'_{i-1}, and use the modified labels of this temporary copy of the original tree to determine the next experiment to perform. After we determine the correct position for s_i within the left-to-right

ordering of the species, we place s_i within the *original tree* T_{i-1}, and then rebalance T_{i-1} using techniques which we now describe.

Modifications of the second type caused by the addition of species s_i to T_{i-1} involve resetting labels and rebalancing the tree. When we insert s_i into T_{i-1} by creating a new leaf, we trace the path from that leaf to the root, updating the labels as we meet them. Thus, if the path goes from a left child to its parent, we increase the left label of the parent by one, and if the path goes from a right child to its parent, we increase the right label of the parent by one. We repeat this process until we reach the root.

The tree we now have includes the first i species, and has correct labels, but may need to be rebalanced. The standard technique [15] for rebalancing red−black trees after an insertion involves a sequence of $O(\log i)$ promotions followed by at most two single rotations. The promotions do not affect the topology of the tree, and hence do not require modification of the labels at the nodes. Updating the labels after performing a single rotation is straightforward, and affects labels at only three nodes. Thus, we can rebalance the tree and adjust the labels after the insertion of s_i in $O(\log i)$ time. Furthermore, the tree T_i we produce containing the first i species will have depth at most $2 \log i$. Thus, the overall cost to the algorithm from this type of modification will be $O(n \log n)$, since we update the balanced binary tree at most n times, and each update costs us $O(\log n)$.

4.4. *Constructing the Tree During Phase I*

Phase I may be modified to produce an actual phylogenetic tree $\mathscr{T}$ for the species set S with an additional n operations. In this way, we can avoid phase two entirely. We will now describe how we can construct $\mathscr{T}$ as we go along.

Suppose, for example, we have determined the phylogenetic tree $\mathscr{T}_{i-1}$ for the first $i - 1$ species. We also assume we have a planar embedding P_{i-1} of this tree $\mathscr{T}_{i-1}$, and a left-to-right ordering on these $i - 1$ species given by P_{i-1}. Using the experiments in Phase I, we then determine the correct position for the next species, s_i, within the left-to-right ordering.

We have already shown how to determine the correct left-to-right ordering on $s_1, s_2, \ldots, s_i$ consistent with a planar embedding of the phylogenetic tree $\mathscr{T}_i$, through the use of these experiments. So assume this ordering is given and that we have constructed $\mathscr{T}_{i-1}$, the phylogenetic tree for the first $i - 1$ species. We will now show how to use this information to determine how to place s_i with $\mathscr{T}_{i-1}$.

Since we know the correct position for s_i in the left-to-right ordering, we have determined one of the following two cases:

- The leaf for s_i follows all the nodes $s_1, s_2, \ldots, s_{i-1}$ in the left-to-right ordering.

- The leaf for s_i falls between a and b in the left-to-right ordering.

In the first case, we must forbid from consideration any topology which might permit, under a rotation, $a < s_i < b$ for any pair of leaves a, b. Therefore, the correct topology is the one in which s_i is made a child of the root of the new tree. So in this case, to create $\mathcal{T}_i$ from $\mathcal{T}_{i-1}$, we should add a new root r', and make r and s_i the children of r', where r is the root of $\mathcal{T}_{i-1}$.

In the second case, a and b might be "representatives" for subtrees of $\mathcal{T}_{i-1}$. So we need to be careful in arguing about the position of s_i. Let $v = lca_{\mathcal{T}_{i-1}}(a, b)$. Let L_v and R_v be the left and right subtrees of v, respectively. Since a and b are adjacent to each other in the list of surviving species, it must be the case that a represents all of L_v and b represents all of R_v. For this to occur, it must be true that $\forall x, y \in L_v$, the experiment on x, y, and s_i has outcome $((x, y), s_i)$ and similarly for all $x, y \in R_v$. Also, since s_i lies between a and b the result of the experiment on a, b, and s_i must be one of $((a, s_i), b)$ or $(b, s_i), a)$. In the first case, the least common ancestor of a and s_i lies below v but above the root of L_v thus uniquely identifying the edge in which the parent p_i of s_i should be inserted. Similarly, in the second case, the edge in which p_i must be inserted is the edge between v and its right child. Making s_i a right child of p_i in the first case and a left child of p_i in the second case ensures that s_i lies between a and b.

Thus, after determining that s_i falls between a and b we can construct $\mathcal{T}_i$ from $\mathcal{T}_{i-1}$ with at most one additional experiment on a, b, and s_i. Although the topology of the evolutionary tree can be determined as we go along constructing the left-to-right ordering of the leaves, we present an interesting algorithm which starts with a valid linear ordering and constructs the topology using at most a linear number of experiments. This algorithm finds an interesting application in the results of [10].

4.5. *Phase II*: *Determining the Tree from the Ordering*

We now have a left to right ordering of all n species. Assume that the species are renumbered so that this ordering $(s_1, s_2, \ldots, s_n)$. To reconstruct the tree, we need the following lemma.

LEMMA 5. *The species s_{i+1} and s_{i+2} are siblings if and only if the outcome of the experiments on sets $\{s_i, s_{i+1}, s_{i+2}\}$ and $\{s_{i+1}, s_{i+2}, s_{i+3}\}$ are $((s_{i+1}, s_{i+2}), s_i)$ and $((s_{I+1}, s_{i+2}), s_{i+3})$, respectively. The experiment on*

$\{s_1, s_2, s_3\}$ produces the outcome $((s_1, s_2), s_3)$ iff s_1 and s_2 form a sibling pair of leaves. The experiment on $\{s_{n-2}, s_{n-1}, s_n\}$ produces the outcome $((s_{n-1}, s_n), s_{n-2})$ iff s_{n-1} and s_n form a sibling pair of leaves.

Proof. We prove only the first statement of the lemma, the other two being easier to prove. If s_{i+1} and s_{i+2} form a sibling pair of leaves, it is clear that the experiments on $\{s_i, s_{i+1}, s_{i+2}\}$ and $\{s_{i+1}, s_{i+2}, s_{i+3}\}$ will be as indicated. Suppose now that (s_{i+1}, s_{i+2}) is the outcome of both experiments. Since we have a full binary tree, if s_{i+1} is a right child of its parent, s_i must be in the left subtree of the parent of s_{i+1}. Thus the least common ancestor of s_i and s_{i+1} is the parent of s_{i+1}. Thus the outcome of the experiment has to be the pair s_i, s_{i+1} contradicting our assumption. So s_{i+1} is a left child. s_{i+2} must be in the right subtree of the parent of s_{i+1}. A similar argument proves that s_{i+2} must be a right child of its parent. This is only possible if s_{i+2} has the same parent as s_{i+1} meaning that they are sibling leaves. ∎

We know that a full binary tree must have at least one pair of sibling leaves. This observation together with the previous lemma gives us the following algorithm for tree reconstruction.

We perform the experiments on $\{s_i, s_{i+1}, s_{i+2}\}$, $i = 1, 2, \ldots,$ until we discover an index i such that the outcome of the experiment on $\{s_{i-1}, s_i, s_{i+1}\}$ is $((s_i, s_{i+1}, s_{i-1})$ and the outcome of the experiment on $\{s_i, s_{i+1}, s_{i+2}\}$ is $((s_i, s_{i+1}), s_{i+2})$. In this case, s_i and s_{i+1} form a sibling pair. Now in any experiment involving s_i if s_{i+1} is substituted for s_i the outcome will be "unchanged." Precisely, if x and y are any other pair of species, then the results of the experiments on $\{x, y, s_i\}$ and $\{x, y, s_{i+1}\}$ will be isomorphic (i.e., with i replaced by $i + 1$). Consequently, we can drop s_i from the list of species we are considering. Now s_{i-1}, s_{i+1}, and s_{i+2} become consecutive species and the algorithm must perform the experiment between these three species. If we think of s_{i+1} as representing the parent of s_i and s_{i+1} we can inductively construct a tree on $n - 1$ species. Giving this parent node, the two children, s_i and s_{i+1} now gives us the tree on all n species.

We perform a linear number of experiments. There are $n - 2$ "original" triples on which experiments need to be done. In addition, each time a species drops out one more experiment needs to be done. This happens only $n - 2$ times causing, in all, $2n - 4$ experiments to be performed. It can also be seen that with suitable data structures the overall running time of the second stage is linear. Note that this stage of the algorithm does not affect the asymptotic number of experiments performed.

4.6. *Analysis of Running Time*

We first analyze the number of experiments used by this algorithm. As we have already noted, we use $\lceil \log_{3/2}(i-1) \rceil$ experiments to place the species s_i in the correct position within the left-to-right ordering of the first $i-1$ species. Thus, overall this algorithm will perform only $n \log_{3/2} n$ experiments in this first place, in which it constructs the left-to-right ordering of the n species. Constructing the tree from the linear ordering on the leaves uses only $O(n)$ experiments, so that the number of experiments used by this algorithm is $n \log_2 n + O(n)$.

We now examine the cost of the algorithm other than that contributed by experiments. Finding least common ancestors of two nodes can be done in $O(\log i)$ time for a red–black tree with i leaves, as we now show. To find the least common ancestor of nodes a, b, in $O(\log i)$ time write down the paths from a and b to the root of the red–black tree. Then compare these two paths, starting at the root and working backward. The last node for which the paths agree is the least common ancestor. Thus, the ith stage (inserting s_i into the linear order) costs us $O(\log^2 i)$ for least common ancestor queries. Still, overall this only contributes $O(n \log^2 n)$ towards the cost of the algorithm.

In order to achieve the overall complexity of $O(n \log^2 n)$ for this algorithm, we need to efficiently pick our experiments to perform. Here as well the red–black trees will be useful in picking our experiments efficiently. We now describe this in detail.

At the beginning of the ith stage, we need to select species a and b defined above. We first compute the indices at which these nodes should be (at a cost of $O(\log n)$), and then we use the labels at the internal nodes of the red–black tree T_{i-1} to determine that pair, a, b. We begin at the root, and trace paths down from the root to the two nodes a and b. It is easy to see that this can be done simply by examining the left and right labels at the internal nodes, and following the correct edge, and thus contributes also an $O(\log i)$ to the cost. This is repeated $O(\log i)$ times, for an overall contribution of $O(\log^2 i)$ for finding the correct position for the species s_i. Thus, these costs contribute overall $O(n \log^2 n)$ towards the cost of the algorithm.

The remaining costs come from the updates to the red–black trees (rebalancing the red–black tree after an insertion and updating labels). With each experiment, we must update the labels on the balanced binary tree T_{i-1} to reflect this information. As noted before, the rebalancing involves modifications to the labels as well, but only costs us $O(\log i)$. Thus, the overall contribution from these updates is $O(n \log n)$.

The overall cost of the algorithm is $O(n \log^2 n)$, but we only require linear space.

5. AN $O(n \log n)$ ALGORITHM

In this section, we give an $O(n \log n)$ deterministic algorithm for determining the evolutionary tree. The algorithm we present has the same overall structure as the algorithm of Section 4 in that it first determines the left-to-right ordering on the species, and then uses the order to construct the tree. As was the case for the algorithm in Section 4, the algorithm can construct the tree as it goes along, rather than using the second phase to construct the tree from the ordering. However, we will only describe the manner by which this algorithm determines the left-to-right ordering on the leaves. The interested reader is directed to Section 4 for details on how to handle both ways of completing the algorithm to construct the tree. Each of these ways only adds a linear number of operations, and so does no change the complexity of the algorithm.

5.1. *The Data Structure*

The red−black tree T'_{i-1} we use to insert the ith species has node set $\{s_1, s_2, \ldots, s_{i-1}\}$. Furthermore, the inorder traversal of *all* the nodes of T'_{i-1} will produce a left-to-right ordering on the first $i-1$ species set which equals the left-to-right ordering of the leaves of a planar embedding of the phylogeny of these species. (An *inorder traversal* of the tree is an ordering of the nodes of the tree which obeys the following recursive rule: *the nodes of the left subtree occur before the root, which occurs before the nodes of the right subtree*.) Thus, the red−black trees T'_{i-1} we use this algorithm differs from the red−black trees T_{i-1} we used in the previous algorithm, where only the leaves of T_{i-1} were drawn from the species set, S.

5.2. *Determining the Left-to-Right Ordering of the Species*

The algorithm works by successively inserting species into the red−black tree for the previously inserted species, in such a way that the inorder traversal of the nodes of the tree we construct at each stage is compatible with the left-to-right ordering of the leaves of a phylogenetic tree for those species.

We will now describe how we insert the species s_i in $O(\log i)$ time.

Let L'_{i-1} be the red−black tree for the first $i-1$ species, into which we wish to insert species s_i. Suppose a and b are the leftmost and rightmost species in T'_{i-1}, respectively. By performing the experiment (a, b, s_i), we can determine if s_i should be put in the interval $[a, b]$ or outside it. Assume that s_i belongs in $[a, b]$ since otherwise s_i can be inserted immediately (into the dangling edge from the root of the phylogenetic tree).

We maintain the following invariants during the $O(\log i)$ steps of the ith insertion. At the beginning of step j of the insertion of s_i, we have narrowed down the possible positions for s_i to two subtrees: a left subtree, L and a right subtree, R, each having its root at level j of the red−black tree, T'_{i-1}. As the names imply, the nodes of L occur before the nodes of R in an inorder traversal of the tree. We let A denote the root of L, and C denote the root of R. Finally, we assume that we know some element B that lies between L and R in the inorder traversal of the red−black tree.

Note that initially these invariants are true. Before step 1, the root of the search tree, which is an element between the elements of the left subtree and the elements of the right subtree, serves as the element B. Its left child is A, the root of L, and its right child is C, the root of R.

At step j we perform the experiments, (A, B, s_i) and (B, C, s_i). We need to show that after performing the experiments we can descend to the next level of the tree while maintaining the invariants. There are three possible outcomes to the experiment (A, B, s_i). We use Lemma 4 to determine which subtree(s) to delete.

$((A, B), s_i)$: In this case, s_i cannot lie between A and B and we can eliminate the right subtree of L.

$((A, s_i), B)$: In this case, s_i cannot lie to the right of B and we can eliminate the entire right tree, R.

$((B, s_i), A)$: In this case, s_i cannot lie to the left of A and we can eliminate the left subtree of L.

Symmetrically, the outcome of the experiment (B, C, s_i) eliminates either one of the two subtrees rooted at the children of R or the entire left subtree L. Note that the outcomes that eliminate all of R and all of L cannot occur simultaneously. Thus the two experiments either eliminate one child each of L and R, or they eliminate one of L and R entirely.

If the experiments eliminate a child of each of L and R, the old element B still serves the function of B. The new element A and C are just the children of the old A and C that have not been eliminated.

If the experiments eliminate an entire subtree, say L, the old C now serves as the intermediate element B. The left and right children of the old C in the search tree serve as A and C respectively. In all cases, we have maintained the invariant and descended a level in the tree. Thus in $O(\log i)$ stages we can place s_i.

5.3. *Running Time*

Updating the search tree to include s_i can now be done in $O(\log i)$ steps using standard techniques. The overall running time for the ith insertion is $O(\log i)$. To count the number of experiments note that a red−black tree

with i leaves ($i - 1$ internal nodes) has depth at most $2 \log i$. Descending one level of depth requires performing two experiments. Thus, in the worst case, a maximum of $4 \log i$ experiments are performed. The total number of experiments over all insertions is upper bounded by $4n \log_2 n$.

6. CONSTRUCTING TREES IN GENERAL

A natural extension is to general trees rather than just binary trees. When we do not require that the evolutionary tree have binary branching, then the result of an experiment on three species a, b, c can indicate that all three pairs are equally closely related. When this occurs, the evolutionary history for a, b, and c is described by the tree in Fig. 2. This outcome would be indicated by returning (a, b, c) as the result of the experiment on the set $\{a, b, c\}$.

6.1. *Constructing Unbounded Degree Trees*

We will describe an algorithm which can construct unbounded degree trees, and which uses $O(n^2)$ experiments. By Theorem 1, this algorithm is optimal.

6.1.1. The Algorithm for Unbounded Degree Trees

Let x be any species in S. We can define an equivalence relation E_x on the species set as follows: $v \sim w$ if the least common ancestor of v and x is the same as the least common ancestor of w and x. The equivalence class for species s is indicated by $[s]$. If T is the true evolutionary tree for S with root r, then the path P from r to x is given by the sequence of nodes $r = v_0, v_1, v_2, \ldots, v_k, v_{k+1} = x$. Note that each v_i has two subtrees, one containing species x as a leaf, and another, which we call T_i. This is described in Fig. 3.

It is then obvious that the equivalence relation E_x which we defined partitions S into the equivalence classes T_i, $i = 0, \ldots, k$. These equivalence classes are ordered (by the distance from the root of the least common ancestor of x and the class), and $T_i < T_j$ for $i < j$. Furthermore, we can compute this equivalence relation E_x and the ordering on the equivalence classes using the following rules:

1. If $((v, w), x)$ or (v, w, x), then $v \sim w$, and
2. If $((v, x), u)$ then $[v] > [u]$ (where $[v]$ is the equivalence class of v).

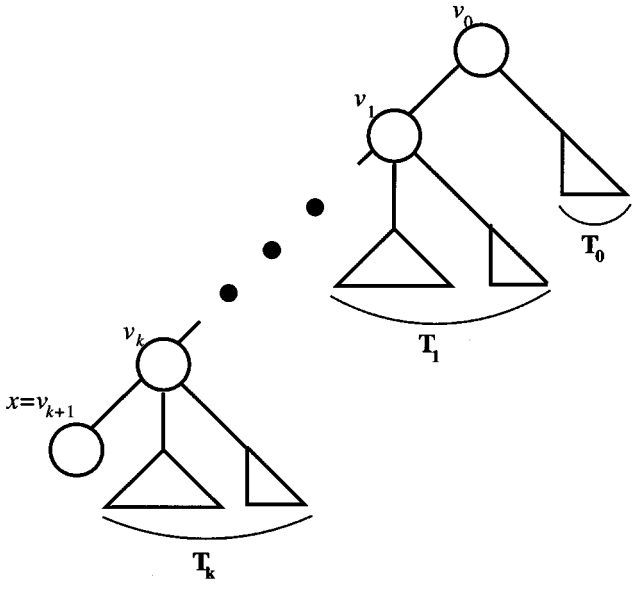

FIG. 3. Computing the Equivalence Relation E_x.

We can compute the equivalence relation and determine the order on the equivalence classes by performing a binary insertion sort, repeatedly finding the correct equivalence class of the next species s_i among the equivalence classes computed thus far. After computing the equivalence relation the algorithm then recurses on each equivalence class.

If the evolutionary tree has a path of length k from root to x, then this algorithm will need $O(n \log k)$ experiments to compute the equivalence relation and determine the ordering on the equivalence classes. If $f(n)$ is the worst case complexity of this algorithm, then $f(n)$ can be seen to satisfy

$$f(n) < \max_k \left\{ n \log k + \left\{ \max_{n_1+n_2+\cdots+n_k=n-1} \{f(n_1) + f(n_2) + \cdots + f(n_k)\} \right\} \right\}.$$

Since $f(n)$ is clearly linear or superlinear, for any choice of k, the worst-case choices of $n_1, n_2, \ldots, n_k$ are where $n_1, n_2, \ldots, n_{k-1}$ are all set to 1 and n_k is set to $n - k$. In this case, the above recurrence can be

simplified as

$$f(n) < \max_k \{n \log k + f(n-k)\}$$

which is upper bounded by $(n^2 \log k)/k$ which is $O(n^2)$. This is optimal, since by Lemma 2, any such algorithm will need to perform $\Omega(n^2)$ experiments.

6.2. *Bounded Degree Trees*

In this section, we will describe a generalization of the $O(n^2)$ algorithm of Section 3 for the construction of bounded degree trees. We will use $O(kn \log n)$ experiments in the case where the number of children for each node is bounded by k.

Let us suppose we have determined the tree T_i for the first i species, and we now wish to place s_{i+1} in T_i. We make the assumption that each node of T_i has at most k children. By Lemma 3, T_i has a weight center v. This node has the property that each of the subtrees of $T_i - \{v\}$ has at most $n/2$ leaves. We will show that we can in at most $\lceil k/2 \rceil$ experiments determine which of these $k+1$ subtrees should contain the leaf for s_{i+1}. Thus, the $O(n^2)$ algorithm of Section 3 can be modified to produce an $O(n^2)$ algorithm which uses $O(kn \log n)$ experiments.

We will label the distinct subtrees rooted at the weight center by $T_1', T_2', \ldots, T_r'$, with $r \leq k$, and let a_i be a species chosen from the set of leaves for the subtree T_i', for $i = 1, 2, \ldots, r$. We wish to determine which of the $k+1$ subtrees should contain the leaf for s_{i+1}. If the answer is one of the subtrees T_j', we say that s_{i+1} lies *below* v, and otherwise we say s_{i+1} lies *above* v.

Consider the result of an experiment on the set of species $\{a_j, a_k, s_{i+1}\}$. The experiment can have one of three results, which are:

$((a_j, a_k)s_{i+1})$: In this case, we know that s_{i+1} belongs *above* v.

$((a_j, s_{i+1})a_k)$: In this case, we know that s_{i+1} must belong to subtree T_j'.

(a_j, a_k, s_{i+1}): This tells us that s_{i+1} does not belong in subtrees T_j' or T_k'.

Thus, the experiment on (a_j, a_k, s_{i+1}) either determines which of the $r+1$ subtrees the leaf for s_{i+1} belongs to or eliminates two of the lower subtrees. Therefore, we need only perform at most $\lceil k/2 \rceil$ experiments to determine the correct subtree for s_{i+1}, for the case where the maximum degree of the tree is constrained to be at most k.

7. SOLUTION TO PROBLEM 1a

So far, we have described several algorithms to find the topology of the phylogenetic tree when the distances or other information available satisfy noisy-ultrametricity. We now show how we can use the topology deduced to solve Problem 1a defined in Section 2. Here the input to the problem is a set of distances that are truly ultrametric.

Let T be the phylogenetic tree topology deduced by our algorithms. Let r be any node of T, $a_1, a_2, \ldots, a_k$ be the children of r, and T_{a_i} be the tree rooted at a_i for $i = 1, \ldots, k$. Assume inductively that we have assigned weights to all edges within T_{a_i} for all i in such a way that the following two conditions hold:

1. For all i and for leaves x and y of T_{a_i} the distance from a_i to x is the same as the distance from a_i to y.

2. For all i and for leaves x and y of T_{a_i} the distance from x to y is the given distance $d(x, y)$.

We show that we can assign weights to the edges (r, a_i) for all i so as to maintain these two invariants. Let the *height* of an internal node be its distance from any leaf in the subtree rooted at this node. For an ultrametric tree this is well-defined. Also by the inductive hypothesis, we know the heights of $a_1, a_2, \ldots, a_k$.

Let $x \in T_{a_1}$ and $y \in T_{a_2}$ be two leaves. Choose r to be of height $d(x, y)/2$ and for each i let the weight of edge (r, a_i) be the difference of the heights of r and a_i. We claim that these weights continue to satisfy the two inductive statements.

CLAIM 1. *Let x, y, z be three leaves in T_{a_i}, T_{a_j}, and T_{a_k}, respectively, where i, j, and k where $i \neq j$ and $j \neq k$. Then if the input distances are ultrametric it must be the case that $d(x, y) = d(y, z)$.*

Proof. The experiment on x, y, z must have had the outcome (x, y, z) or $((x, z), y)$ because of the conditions on i, j, and k. In either case, by ultrametricity $d(x, y) = d(y, z)$. ∎

Note that by transitivity this claim implies that the distance between any two leaves which lie in different subtrees below r is the same as the distance between any other two such leaves. This shows that our definition of the height of r does not depend on the x and y chosen to define it. Note also that by the fact that we have a noisy ultrametric input, the height of r is greater than or equal to the height of a_i for all i and hence that all edges are assigned nonnegative weights.

The above discussion immediately implies that the distance from r to any of the leaves in the subtree rooted at r is equal to the height of r thus

establishing one of the two inductive invariants. It also shows that for any x, y in the subtree rooted at r, the distance from x to y in the tree is the same as the given distance, $d(x, y)$. This proves the inductive hypothesis and shows that given any ultrametric set of distances, we can find an edge-weighted phylogenetic tree realizing these distances. Note that the computational complexity of determining the edge-weights is $O(n)$ since the weight of each edge from bottom-up can be computed in constant time by maintaining an array of heights of the various nodes. If we think of experiments as probes into the distance matrix, then no additional experiments are needed for this stage since the experiments used in determining the topology also contain the information necessary to determine the node-heights.

8. CONCLUSIONS AND OPEN PROBLEMS

A natural question that arises in this model is: given a set of k experiments involving n species and their outcomes, decide if there is an evolutionary tree consistent with the experiments without performing additional experiments. This problem occurs naturally in the theory of relational databases, as the problem of synthesizing a relational algebra expression from a simple tableau. In [1], Aho *et al*. gave a simple $O(kn)$ algorithm for this problem. Recently, Henzinger *et al*. [11] have found a $min\{O(kn^{1/2}), O(K + n^2 \log n)\}$ deterministic algorithm and an $O(k \log^3 n)$ randomized algorithm for this problem. The only lower bound known for this question is the obvious one of n.

The consistency question can be thought of as being analogous to the question of whether a set of ordering relations defines a partial order. Carrying the analogy a little further, one could ask for the number of binary trees consistent with a given set of experiments. Another question is: How many additional experiments need to be performed in order to determine a unique binary tree? To the best of our knowledge, all of these questions are still open and possibly at least as hard as the corresponding questions for partial orders and sorting in the usual comparison tree model.

REFERENCES

1. A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions, *SIAM J. Comput*. **10**, No. 3 (1981), 405–421.
2. J. Culberson and P. Rudnicki, A fast algorithm for constructing trees from distance matrices, *Inform. Process. Lett*. **30** (1989), 215–220.

3. W. H. E. Day, Computational complexity of inferring phylogenies from dissimilarity matrices, *Bull. Math. Biol.* **49**, No. 4 (1987) 461–467.
4. A. Dress and A. von Haessler (Eds.), "Trees and Hierarchical Structures, Proceedings, Bielefeld 1987," Lecture Notes in Biomathematics, Springer-Verlag, Berlin/New York, 1987.
5. M. Farach, S. Kannan, and T. Warnow, A robust model for inferring optimal evolutionary trees, *Algorithmica* **13**, No. 1 (1995), 155–179.
6. J. S. Farris, Estimating phylogenetic trees from distance matrices, *Amer. Nat.* **106** (1972), 645–668.
7. J. Felsenstein, Numerical methods for inferring evolutionary trees, *Quart. Rev. Biol.* **57**, No. 4 (Dec. 1982), 379–404.
8. J. Hein, An optimal algorithm to reconstruct trees from additive distance matrices, *Bull. Math. Biol.* **51**, No. 5 (1989), 597–603.
9. J. Hein, A tree reconstruction method that is economical in the number of pairwise comparisons used, *Mol. Biol. Evol.* **6**, No. 6 (1989), 669–684.
10. S. Kannan, T. Warnow, and S. Yooseph, Computing the local consensus of trees, *in* "Proceedings of ACM–SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, pp. 68–77.
11. M. Henzinger, V. King, and T. Warnow, Constructing a tree from homeomorphic subtrees, with applications to computational molecular biology, unpublished manuscript.
12. Deleted in proof.
13. C. G. Sibley and J. E. Ahlquist, Phylogeny and classification of birds based on the data of DNA–DNA hybridization, *Current Ornithol.* **1** (1983), 245–292.
14. R. R. Sokal and P. H. A. Sneath, "Principles of Numerical Taxonomy," Freeman, San Francisco.
15. R. E. Tarjan, "Data Structures and Network Algorithms," SIAM, Philadelphia, 1983.
16. T. Warnow, "Combinatorial Algorithms for Constructing Phylogenetic Trees," Ph.D. Thesis, Univ. of California, Berkeley, 1991.
17. M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer, Additive evolutionary trees, *J. Theoret. Biol.* **64** (1977), 199–213.
18. M. S. Waterman, T. F. Smith, and W. A. Beyer, Some biological sequence metrics, *Adv. Math.* **20** (1976), 367–387.