The Complexity of Constructing Evolutionary Trees Using Experiments

Gerth Stlting Brodal[1,⋆], Rolf Fagerberg[1,⋆],
Christian N. S. Pedersen[1,⋆], and Anna Östlin[2,⋆⋆]

[1] BRICS[†], Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Århus C, Denmark. E-mail: {gerth,rolf,cstorm}@brics.dk
[2] Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden. E-mail: Anna.Ostlin@cs.lth.se

Abstract We present tight upper and lower bounds for the problem of constructing evolutionary trees in the experiment model. We describe an algorithm which constructs an evolutionary tree of n species in time $O(nd \log_d n)$ using at most $n\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $n(\log n + O(1))$ experiments for $d = 2$, where d is the degree of the tree. This improves the previous best upper bound by a factor $\Theta(\log d)$. For $d = 2$ the previously best algorithm with running time $O(n \log n)$ had a bound of $4n \log n$ on the number of experiments. By an explicit adversary argument, we show an $\Omega(nd \log_d n)$ lower bound, matching our upper bounds and improving the previous best lower bound by a factor $\Theta(\log_d n)$. Central to our algorithm is the construction and maintenance of separator trees of small height, which may be of independent interest.

1 Introduction

The evolutionary relationship for a set of species is commonly described by an evolutionary tree, where the leaves correspond to the species, the root corresponds to the most recent common ancestor for the species, and the internal nodes correspond to the points in time where the evolution has diverged in different directions. The evolutionary history for a set of species is rarely known, hence estimating the true evolutionary tree for a set of species from obtainable information about the species is of great interest. Estimating the true evolutionary tree computationally requires a model describing how to use available information about species to estimate aspects of the true evolutionary tree. Given a model, the problem of estimating the true evolutionary tree is often referred to as constructing the evolutionary tree in that model.

⋆ Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
⋆⋆ Partially supported by TFR grant 1999-344.
† Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation.

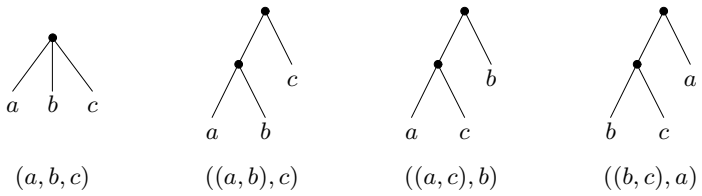

Figure 1. The four possible outcomes of an experiment for three species a, b and c

In this paper we study the problem of constructing evolutionary trees in the *experiment model* proposed by Kannan, Lawler and Warnow in [6]. In this model the information about the species is obtained by *experiments* which can yield the evolutionary tree for any triplet of species, cf. Fig. 1. The problem of constructing an evolutionary tree for a set of n species in the experiment model is to construct a rooted tree with no unary internal nodes and n leaves labeled with the species such that the topology of the constructed tree is consistent with all possible experiments involving the species. Hence, the topology of the constructed tree should be such that the induced tree for any three species is equal to the tree returned by an experiment on those three species.

The relevance of the experiment model depends on the possibility of performing experiments. A standard way to express phylogenetic information is by a distance matrix. A distance matrix for a set of species is a matrix where entry M_{ij} represents the evolutionary distance between species i and j, measured by some biological method (see [6] for further details). For three species a, b and c where $M_{ab} < \min\{M_{ac}, M_{bc}\}$ it is natural to conclude that the least common ancestor of a and b is below the least common ancestor of a and c, i.e. the outcome of an experiment on a, b and c can be decided by inspecting M_{ab}, M_{ac} and M_{bc}. The consistency of experiments performed by inspecting a distance matrix depends entirely on the distance matrix. Kannan *et al.* in [6] define a distance matrix as noisy-ultrametric if there exists a rooted evolutionary tree such that for all triplets of species a, b and c it holds that $M_{ab} < \min\{M_{ac}, M_{bc}\}$ if and only if the least common ancestor of a and b is below the least common ancestor of a and c in the rooted evolutionary tree. Hence, if a noisy-ultrametric distance matrix for the set of species can be obtained, it can be used to perform experiments consistently. Another and more direct method for performing experiments is DNA-DNA hybridization as described by Sibley and Ahlquist in [9]. In this experimental technique one measures the temperature at which single stranded DNA from two different species bind together. The binding temperature is correlated to the evolutionary distance, i.e. by measuring the binding temperatures between DNA strands from three species one can decide the outcome of the experiment by deciding which pair of the three species bind together at the highest temperature.

Kannan *et al.* introduce and study the experiment model in [6] under the assumption that experiments are flawless in the sense that they do not contradict

each other, i.e. it is always possible to construct an evolutionary tree for a set of species that is consistent with all possible experiments involving the species. They present algorithms for constructing evolutionary trees with bounded as well as unbounded degree, where the degree of a tree is the maximum number of children for an internal node. For constructing binary evolutionary trees they present three different algorithms with running times $O(n \log n)$, $O(n \log^2 n)$ and $O(n^2)$ respectively, using $4n \log n$, $n \log_{3/2} n$ and $n \log n$ experiments respectively, where $\log n$ denotes $\log_2 n$. For constructing an evolutionary tree of degree d they present an algorithm with running time $O(n^2)$ using $O(dn \log n)$ experiments. Finally, for the general case they present an algorithm with running time $O(n^2)$ using $O(n^2)$ experiments together with a matching lower bound. Kao, Lingas, and Östlin in [7] present a randomized algorithm for constructing evolutionary trees of degree d with expected running time $O(nd \log n \log \log n)$. They also prove a lower bound $\Omega(n \log n + nd)$ on the number of experiments. The best algorithm so far for constructing evolutionary trees of degree d is due to Lingas, Olsson, and Östlin, who in [8] present an algorithm with running time $O(nd \log n)$ using the same number of experiments.

In this paper we present the first tight upper and lower bounds for the problem of constructing evolutionary trees of degree d in the experiment model. We present an algorithm which constructs an evolutionary tree for n species in time $O(nd \log_d n)$ using at most $n\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $n(\log n + O(1))$ experiments for $d = 2$, where d is the degree of the constructed tree. The algorithm is a further development of an algorithm from [8]. Our construction improves the previous best upper bound by a factor $\Theta(\log d)$. For $d = 2$ the previously best algorithm with running time $O(n \log n)$ had a bound of $4n \log n$ on the number of experiments. The improved constant factors on the number of experiments are important because experiments are likely to be expensive in practice, cf. Kannan *et al.* [6]. By an explicit adversary argument, we show an $\Omega(nd \log_d n)$ lower bound, matching our upper bounds and improving the previous best lower bound by a factor $\Theta(\log_d n)$.

Our algorithm also supports the insertion of new species with a running time of $O(md \log_d(n + m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n + m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where n is the number of species in the tree to begin with, m is the number of insertions, and d is the maximum degree of the tree during the sequence of insertions. Central to our algorithm is the construction and maintenance of separator trees of small height. These algorithms may be of independent interest. However, due to lack of space we have omitted the details on separator trees. For further details we refer the reader to the full version of the paper [5].

The rest of this paper is organized as follows. In Sect. 2 we define separator trees and state results on the construction and efficiently maintenance of separator trees of small height. In Sect. 3 we present our algorithm for constructing and maintaining evolutionary trees. In Sect. 4 and 5 the lower bound is proved using an explicit adversary argument. The adversary strategy used is an extension of an adversary used by Borodin, Guibas, Lynch, and Yao [3] for proving

a trade-off between the preprocessing time of a set of elements and membership queries, and Brodal, Chaudhuri, and Radhakrishnan [4] for proving a trade-off between the update time of a set of elements and the time for reporting the minimum of the set.

2 Separator Trees

In this section we define separator trees and state results about efficient algorithms for their constructing and maintenance. For further details see [5].

Definition 1. *Let T be an unrooted tree with n nodes. A* separator tree *S_T for T is a rooted tree on the same set of nodes, defined recursively as follows: The root of S_T is a node u in T, called the* separator node*. The removal of u from T disconnects T into disjoint trees $T_1, \ldots, T_k$, where k is the number of edges incident to u in T. The children of u in S_T are the roots of separator trees for $T_1, \ldots, T_k$.*

Clearly, there are many possible separator trees S_T for a given tree T. An example is shown in Fig. 2.

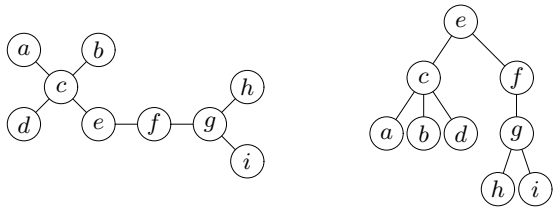

Figure 2. A tree T (left) and a separator tree S_T for T (right)

For later use, we note the following facts for separator trees:

Fact 1 *Let S_T be a separator tree for T, and let v be a node in T. If S_v denotes the subtree of S_T rooted at v, then:*

1. *The subgraph T_v induced by the nodes in S_v is a tree, and S_v is a separator tree for T_v.*
2. *For any edge from T with exactly one endpoint in T_v, the other endpoint is an ancestor of v in S_T, and each ancestor of v can be the endpoint of at most one such edge.*

The main point of a separator tree S_T is that it may be balanced, even when the underlying tree T is not balanced for any choice of root. The notion of balanced separator trees is contained in the following definition, where the size $|T|$ of a tree T denotes the number of nodes in T, and where T_i refers to the trees $T_1, \ldots, T_k$ from Definition 1.

Definition 2. *A separator tree is a* t-separator tree*, for a threshold* $t \in [1/2, 1]$*, if* $|T_i| \leq t|T|$ *for each* T_i *and the separator tree for each* T_i *is also a* t*-separator tree.*

In [5] we first give a simple algorithm for constructing 1/2-separator trees in time $O(n \log n)$. We then improve the running time of the algorithm to $O(n)$ by adopting additional data structures. We note that a 1/2-separator tree has height at most $\lfloor \log n \rfloor$.

We also consider dynamic separator trees under the insertion of new nodes into a tree T and its corresponding separator tree S_T, and show how to maintain separators trees with small height in logarithmic time per insertion. Our methods for maintaining balance and height in separator trees during insertions of new nodes are based on rebuilding of subtrees, and are inspired by methods of Andersson and Lai described in [1,2] for maintaining small height in binary search trees. We first show how the linear time construction algorithm for 1/2-separator trees leads to a simple algorithm for keeping separator trees well balanced. The height bound achieved by this algorithm is $O(\log n)$, using $O(\log n)$ amortized time per update. We then use a two-layered structure to improve the height bound to $\log n + O(1)$ without sacrificing the time bound. The improved constant factor in the height bound is significant for our use of separator trees for maintaining evolutionary trees in the experiment model, since the number of experiments for an insertion of a new species will turn out to be proportional to the height of the separator tree. Furthermore, this height bound is within an additive constant of the best bound possible, as trees exist where any separator tree must have height at least $\lfloor \log n \rfloor$, e.g. a tree which is a single path.

Finally, we extend the separator trees with a specific ordering of the children, facilitating our use of separator trees in Sect. 3 for finding insertion points for new species in evolutionary trees. The basic idea is to speed up the search in the separator tree by considering the children of the nodes in decreasing size-order. This ensures a larger reduction of subtree size in the case that many children have to be considered before the subtree to proceed the search in is found. Our main result about separator trees is summarized in the following theorem.

Theorem 1. *Let* T *be an unrooted tree initially containing* n *nodes. After* $O(n)$ *time preprocessing, an ordered separator tree for* T *can in time* $O(m \log(n+m))$ *be maintained during* m *insertions in a way such that the height is bounded by* $\log(n+m) + 5$ *and such that for any path* $(v_1, v_2, \ldots, v_\ell)$ *from the root* v_1 *to a node* v_ℓ *in the separator tree, the followings holds*

$$\prod_{d_i \leq 2} 2 \cdot \prod_{d_i > 2} d_i < 16d(n+m) , \tag{1}$$

where d_i *is the number which* v_{i+1} *has in the ordering of the children of* v_i*, for* $1 \leq i < \ell$*, and* d *is* $\max\{d_1, \ldots, d_{\ell-1}\}$*.*

3 Algorithm for Constructing and Maintaining Evolutionary Trees

In this section we describe an algorithm for constructing an evolutionary tree T in the experiment model for a set of n species in time $O(nd \log_d n)$, where d is the degree of the tree. Note that d is not known by the algorithm in advance. The algorithm is a further development of an algorithm by Lingas *et al.* in [8]. Our algorithm also supports the insertion of new species with running time $O(md \log_d(n+m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n+m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where n is the number of species in the tree to begin with, m is the number of insertions, and d is the maximum degree of the tree during the sequence of insertions.

The construction algorithm inserts one species at the time into the tree in time $O(d \log_d n)$ until all n species have been inserted. The search for the insertion point of a new species a is guided by a separator tree S_T for the internal nodes of the evolutionary tree T for the species inserted so far. The search starts at the root of S_T. In a manner to be described below, we decide by experiments which subtree, rooted at a child of the root in S_T, the search should continue in. This is repeated recursively until the correct insertion point in T for a is found. We keep links between corresponding nodes in S_T and T for switching between the two trees. To facilitate the experiments, we for each internal node in T maintain a pointer to an arbitrary leaf in its subtree. When inserting a new internal node in T this pointer is set to point to the new leaf which caused the insertion of the node.

We say that the insertion point of a is *incident* to a node v, if

1. a should be inserted directly below v, or
2. a should split an edge which is incident to v by creating a new internal node on the edge and make a a leaf below the new node, or
3. if v is the root of T, a new root of T should be created with a and v as its two children.

The invariant for the search is the following. Assume we have reached node v in the separator tree for the internal nodes in T, and let S_v be the internal nodes of T which are contained in the subtree of S_T rooted at v (including v). Then the insertion point of the new species a is incident to a node in S_v.

Let v be the node in S_T for which we want to decide if the insertion point for the new species a is in the subtree above v in T; if it is in a subtree rooted at a child of v in T; or if a should be inserted as a new child of v. We denote by $u_1, \ldots, u_k$ the children of v in T, where $u_1, \ldots, u_{k'}$ are nodes in distinct subtrees $T_1, \ldots, T_{k'}$ below v in S_T, whereas $u_{k'+1}, \ldots, u_k$ are leaves in T or are nodes above v in S_T. The order of the subtrees $T_1, \ldots, T_{k'}$ below v in S_T is given by the ordered separator tree S_T and determines the order of $u_1, \ldots, u_{k'}$. The remaining children $u_{k'+1}, \ldots, u_k$ of v may appear in any order.

We perform at most $\lceil k/2 \rceil$ experiments at v. The i'th experiment is on the species a, b and c, where b and c are leaves in T below u_{2i-1} and u_{2i} respectively.

The leaves b and c can be located using the pointers stored at u_{2i-1} and u_{2i}. Note that the least common ancestor of b and c in T is v. If k is odd then the species b and c in the $\lceil k/2 \rceil$'th experiment is chosen as leaves in T below u_k and u_1 respectively, and note that the two leaves are distinct because $k \geq 2$ by definition. There are four possible outcomes of the i'th experiment corresponding to Fig. 1:

1. (a, b, c) implies that the insertion point for a is incident to adescendent of u_j, where b and c are not descendents of u_j, or a is a new leaf below v.
2. $((a, b), c)$ implies that the insertion point for a is incident to a descendent of u_{2i-1}, since the least common ancestor of a and b is below v in T.
3. $((a, c), b)$ is symmetric to the above case and the insertion point of a is incident to a descendent of u_{2i} (u_1 for the $\lceil k/2 \rceil$'th experiment if k odd).
4. $((b, c), a)$ implies that the insertion point of a is in the subtree above v, since the least common ancestor of a and b is above v. If v is the present root of T, a new root should be created with children a and v.

We perform experiments for increasing i until we get an outcome difference from Case 1, or until we have performed all $\lceil k/2 \rceil$ experiments all with outcome cf. Case 1. In the latter case species a should be inserted directly below v in T as a new child. In the former case, when the outcome of an experiment is different from Case 1, we know in which subtree adjacent to v in T the insertion point for species a is located. If there is no corresponding subtree below v in S_T, then we have identified the edge incident to v in T which the insertion of species a should split. Otherwise we continue recursively searching for the insertion point for species a at the child of v in S_T which roots the separator tree for the subtree adjacent to v which has been identified to contain the insertion point for a. When the insertion point for species a is found, we insert one leaf and at most one internal node into T, and S_T is updated according to Theorem 1.

Lemma 1. *Given an evolutionary tree T for n species with degree d, and a separator tree S_T for T according to Theorem 1, then a new species a can be inserted into T and S_T in amortized time $O(d \log_d n)$ using at most $\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $\log n + O(1)$ experiments for $d = 2$.*

Proof. Let $v_1, \ldots, v_\ell$ be the nodes in S_T (and T) visited by the algorithm while inserting species a, where v_1 is the root of S_T and v_{j+1} is a child of v_j in S_T. Define d_i by v_{i+1} being the d_i'th child of v_i in S_T, for $1 \leq i < \ell$.

For $d = 2$ we perform exactly one experiment at each v_i. The total number of experiments is thus bounded by the height of the separator tree. By Theorem 1 it follows that the number of experiments is bounded by $\log n + O(1)$. In the following we consider the case where $d \geq 3$.

For $i < \ell$, let x_i denote the number of experiments performed at node v_i. We have $x_i \leq \lceil d/2 \rceil$ and $d_i \geq 2x_i - 1$, since each experiment considers two children of v_i in T and the first experiment also identifies if a should be inserted into the subtree above v_i. At v_ℓ we perform at most $\lceil d/2 \rceil$ experiments.

For $d_1, \ldots, d_{\ell-1}$ we from Theorem 1 have the constraint $\prod_{d_i \leq 2} 2 \cdot \prod_{d_i > 2} d_i \leq 16dn$, since $|S_T| \leq n-1$. To prove the stated bound on the worst case number of experiments we must maximize $\sum_{i=0}^{\ell} x_i$ under the above constraints. We have

$$\begin{aligned}
\log(16dn) &\geq \sum_{d_i \leq 2} 1 + \sum_{d_i > 2} \log d_i \\
&\geq \sum_{x_i = 1} 1 + \sum_{x_i > 1} \log d_i \\
&\geq \sum_{x_i = 1} x_i + \sum_{x_i > 1} x_i \frac{1}{x_i} \log(2x_i - 1) \\
&\geq \frac{1}{\lceil d/2 \rceil} \log(2\lceil d/2 \rceil - 1) \sum_{i=1}^{\ell-1} x_i \ ,
\end{aligned}$$

where the second inequality holds since $x_i > 1$ implies $d_i \geq 3$. The last inequality holds since for $f(x) = \frac{1}{x}\log(2x-1)$ we have $1 > f(2) > f(3)$ and $f(x)$ is decreasing for $x \geq 3$, i.e. $f(x)$ is minimized when x is maximized.

We conclude that $\sum_{i=1}^{\ell-1} x_i \leq \lceil d/2 \rceil \log_{2\lceil d/2 \rceil - 1}(16dn)$, i.e. for the total number of experiments we have $\sum_{i=1}^{\ell} x_i \leq \lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(16dn) + 1)$.

The time needed for the insertion is proportional to the number of experiments performed plus the time to update S_T. By Theorem 1 the total time is thus $O(d \log_d n)$. □

From Lemma 1 and Theorem 1 we get the following bounds for constructing and maintaining an evolutionary tree under the insertion of new species in the experiment model.

Theorem 2. *After $O(n)$ preprocessing time an evolutionary tree T for n species can be maintained under m insertions in time $O(dm \log_d(n+m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n+m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where d is the maximum degree of the tree during the sequence of insertions.*

4 Adversary for Constructing Evolutionary Trees

To prove a lower bound on the number of experiments required for constructing an evolutionary tree of n species with degree at most d, we describe an adversary strategy for deciding the outcome of experiments. The adversary is required to give consistent answers, i.e. the reported outcome of an experiment is not allowed to contradict the outcome of previously performed experiments. A construction algorithm is able to construct an unambiguous evolutionary tree based on the performed experiments when the adversary is not able to answer any additional experiments in such a way that it contradicts the constructed evolutionary tree. The role of the adversary is to force any construction algorithm

to perform provably many experiments in order to construct an unambiguous evolutionary tree.

To implement the adversary strategy for deciding the outcome of experiments in a consistent way, the adversary maintains a rooted infinite d-ary tree, D, where each of the n species are stored at one of the nodes, allowing nodes to store several species. Initially all n species are stored at the root. For each experiment performed, the adversary can move the species downwards by performing a sequence of *moves*, where each move shifts a species from the node it is currently stored at to a child of the node.

By deciding the outcome of experiments, the adversary reveals information about the evolutionary relationships between the species to the construction algorithm performing the experiments. The distribution of the n species on D represents the information revealed by the adversary (together with the forbidden and conflicting lists introduced below). The evolutionary tree T to be established by the construction algorithm will be a connected subset of nodes of D including the root. Initially, when all species are stored at the root, the construction algorithm has no information about the evolutionary relationships. The evolutionary relationships revealed to the construction algorithm by the current distribution of the species on D corresponds to the tree formed by the paths from the root of D to the nodes storing at least one species. More precisely, the correspondence between the final evolutionary tree T and the current distribution of the species on D is that if v is a leaf of T labeled a then species a is stored at some node on the path in D from the root to the node v.

Our objective is to prove that if an algorithm computes T, then the n species on average must have been moved $\Omega(\log_d n)$ levels down by the adversary, and that the number of moves by the adversary is a fraction $O(1/d)$ of the number of experiments performed. These two facts imply the $\Omega(nd \log_d n)$ lower bound on the number of experiments required.

To control its strategy for moving species on D, the adversary maintains for each species a a *forbidden list* $\mathrm{F}(a)$ of nodes and a *conflicting list* $\mathrm{C}(a)$ of species. If a is stored at node v, then $\mathrm{F}(a)$ is a subset of the children $c_1, \ldots, c_d$ of v, and $\mathrm{C}(a)$ is a subset of the other species stored at v. If $c_i \in \mathrm{F}(a)$, then a is not allowed to be moved to child c_i, and if $b \in \mathrm{C}(a)$ then a and b must be moved to two distinct children of v. It will be an invariant that $b \in \mathrm{C}(a)$ if and only if $a \in \mathrm{C}(b)$. Initially all forbidden and conflicting lists are empty. The adversary maintains the forbidden and conflicting lists such that the size of the forbidden and conflicting lists of a species a is bounded by the invariant

$$|\mathrm{F}(a)| + |\mathrm{C}(a)| \leq d - 2 \ . \tag{2}$$

The adversary uses the sum $|\mathrm{F}(a)|+|\mathrm{C}(a)|$ to decide when to move a species a one level down in D. Whenever the invariant (2) becomes violated because $|\mathrm{F}(a)|+|\mathrm{C}(a)| = d-1$, for a species a stored at a node v, the adversary moves a to a child $c_i \notin \mathrm{F}(a)$ of v. Since $|\mathrm{F}(a)| \leq d-1$, such a $c_i \notin \mathrm{F}(a)$ is guaranteed to exist. When moving a from v to c_i, the adversary updates the forbidden and conflicting lists as follows: For all $b \in \mathrm{C}(a)$, a is deleted from $\mathrm{C}(b)$ and c_i is

inserted into $\mathrm{F}(b)$. If c_i was already in $\mathrm{F}(b)$, the sum $|\mathrm{F}(b)| + |\mathrm{C}(b)|$ decreases by one, if c_i was not in $\mathrm{F}(b)$ the sum remains unchanged. Finally, $\mathrm{F}(a)$ and $\mathrm{C}(a)$ are assigned the empty set.

For two species a and b, we define their *least common ancestor*, $\mathrm{LCA}(a, b)$, to be the least common ancestor of the two nodes storing a and b in D. We denote $\mathrm{LCA}(a, b)$ as *fixed* if it cannot be changed by future moves of a and b by the adversary. If $\mathrm{LCA}(a, b)$ is fixed then the least common ancestor of the two species a and b in T is the node $\mathrm{LCA}(a, b)$. If a is stored at node v_a and b is stored at node v_b, it follows that $\mathrm{LCA}(a, b)$ is fixed if and only if one of the following four conditions is satisfied.

1. $v_a = \mathrm{LCA}(a, b) = v_b$ and $a \in \mathrm{C}(b)$ (and $b \in \mathrm{C}(a)$).
2. $v_a \neq \mathrm{LCA}(a, b) = v_b$ and $c_i \in \mathrm{F}(b)$, where c_i is the child of v_b such that the subtree rooted at c_i contains v_a.
3. $v_a = \mathrm{LCA}(a, b) \neq v_b$ and $c_i \in \mathrm{F}(a)$, where c_i is the child of v_a such that the subtree rooted at c_i contains v_b.
4. $v_a \neq \mathrm{LCA}(a, b) \neq v_b$.

In Case 1, species a and b are stored at the same node and cannot be moved to the same child because $a \in \mathrm{C}(b)$, i.e. $\mathrm{LCA}(a, b)$ is fixed as the node which currently stores a and b. Cases 2 and 3 are symmetric. In Case 2, species a is stored at a descendant of a child c_i of the node storing b, and b cannot be moved to c_i because $c_i \in \mathrm{F}(b)$, i.e. $\mathrm{LCA}(a, b)$ is fixed as the node which currently stores b. Finally, in Case 4, species a and b are stored at nodes in disjoint subtrees, i.e. $\mathrm{LCA}(a, b)$ is already fixed.

The operation $\mathrm{Fix}(a, b)$ ensures that $\mathrm{LCA}(a, b)$ is fixed as follows:

1. If $v_a = \mathrm{LCA}(a, b) = v_b$ and $a \notin \mathrm{C}(b)$ then insert a into $\mathrm{C}(b)$ and insert b into $\mathrm{C}(a)$.
2. If $v_a \neq \mathrm{LCA}(a, b) = v_b$ and $c_i \notin \mathrm{F}(b)$, where c_i is the child of v_b such that the subtree rooted at c_i contains v_a, then insert c_i into $\mathrm{F}(b)$.
3. If $v_a = \mathrm{LCA}(a, b) \neq v_b$ and $c_i \notin \mathrm{F}(a)$, where c_i is the child of v_a such that the subtree rooted at c_i contains v_b, then insert c_i into $\mathrm{F}(a)$.

Otherwise $\mathrm{Fix}(a, b)$ does nothing. If performing $\mathrm{Fix}(a, b)$ increases $|\mathrm{F}(a)|$ such that $|\mathrm{F}(a)| + |\mathrm{C}(a)| = d - 1$, then a is moved one level down as described above. Similarly, if $|\mathrm{F}(b)| + |\mathrm{C}(b)| = d - 1$ then b is moved one level down. After performing $\mathrm{Fix}(a, b)$ we thus have that $|\mathrm{F}(a)| + |\mathrm{C}(a)| \leq d - 2$ and $|\mathrm{F}(b)| + |\mathrm{C}(b)| \leq d - 2$, which ensures that the invariant (2) is not violated.

When the construction algorithm performs an experiment on three species a, b and c, the adversary decides the outcome of the experiment based on the current distribution of the species on D and the content of the conflicting and forbidden lists. To ensure the consistency of future answers, the adversary first fix the least common ancestors of a, b and c by applying the operation Fix three times: $\mathrm{Fix}(a, b)$, $\mathrm{Fix}(a, c)$ and $\mathrm{Fix}(b, c)$. After having fixed $\mathrm{LCA}(a, b)$, $\mathrm{LCA}(a, c)$, and $\mathrm{LCA}(b, c)$, the adversary decides the outcome of the experiment by examining $\mathrm{LCA}(a, b)$, $\mathrm{LCA}(a, c)$, and $\mathrm{LCA}(b, c)$ in D as described below. The four cases correspond to the four possible outcomes of an experiment cf. Fig. 1.

1. If $\mathrm{LCA}(a,b) = \mathrm{LCA}(b,c) = \mathrm{LCA}(a,c)$ then return (a,b,c).
2. If $\mathrm{LCA}(a,b) \neq \mathrm{LCA}(b,c) = \mathrm{LCA}(a,c)$ then return $((a,b),c)$.
3. If $\mathrm{LCA}(a,c) \neq \mathrm{LCA}(a,b) = \mathrm{LCA}(b,c)$ then return $((a,c),b)$.
4. If $\mathrm{LCA}(b,c) \neq \mathrm{LCA}(a,b) = \mathrm{LCA}(a,c)$ then return $((b,c),a)$.

5 Lower Bound Analysis

We will argue that the above adversary strategy forces any construction algorithm to perform at least $\Omega(nd \log_d n)$ experiments before being able to conclude unambiguously the evolutionary relationships between the n species.

Theorem 3. *The construction of an evolutionary tree for n species requires $\Omega(nd \log_d n)$ experiments, where d is the degree of the constructed tree.*

Proof. We first observe that an application of $\mathrm{Fix}(a,b)$ at most increases the size of the two conflicting lists, $\mathrm{C}(a)$ and $\mathrm{C}(b)$, by one, or the size of one of the forbidden list, $\mathrm{F}(a)$ or $\mathrm{F}(b)$, by one. If performing $\mathrm{Fix}(a,b)$ increases the sum $|\mathrm{F}(a)| + |\mathrm{C}(a)|$ to $d-1$, then species a is moved one level down in D and $\mathrm{F}(a)$ and $\mathrm{C}(a)$ are emptied, which causes the overall sum of the sizes of forbidden and conflicting lists to decrease by $d-1$. This implies that a total of k Fix operations, starting with the initial configuration where all conflicting and forbidden lists are empty, can cause at most $2k/(d-1)$ moves. Since an experiment involves three Fix operations, we can bound the total number of moves during m experiments by $6m/(d-1)$.

Now consider the configuration, i.e. the distribution of species and the content of conflicting and forbidden lists, when the construction algorithm computing the evolutionary tree terminates. Some species may have nonempty forbidden lists or conflicting lists. By forcing one additional move on each of these species as described in Sect. 4, we can guarantee that all forbidden and conflicting lists are empty. At most n additional moves must be performed.

Let T' be the tree formed by the paths in D from the root to the nodes storing at least one species. We first argue that all internal nodes of T' have at least two children. If a species has been moved to a child of a node, then the forbidden list or conflicting list of the species was nonempty. If the forbidden list was nonempty, then each of the forbidden subtrees already contained at least one species, and if the conflicting list was nonempty there was at least one species on the same node that was required to be moved to another subtree, at the latest by the n additional moves. It follows that if a species has been moved to a child of a node then at least one species has been moved to another child of the node, implying that T' has no node with only one child.

We next argue that all n species are stored at the leaves of T' and that each leaf of T' stores either one or two species. If there is a non-leaf node in T' that still contains a species, then this species can be moved to at least two children already storing at least one species in the respective subtrees, implying that the adversary can force at least two distinct evolutionary trees which are consistent with the answers returned. This is a contradiction. It follows that all species

are stored at leaves of T'. If a leaf of T' stores three or more species, then an experiment on three of these species can generate different evolutionary trees, which again is a contradiction. We conclude that each leaf of T' stores exactly one or two species, and all internal nodes of T' store no species. It follows that T' has at least $n/2$ leaves.

For a tree with k leaves and degree d, the sum of the depths of the leaves is at least $k \log_d k$. Since each leaf of T' stores at most two species, the n species can be partitioned into two disjoint sets of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ such that in each set all species are on distinct leaves of T'. The sum of the depths of all species is thus at least $\lceil n/2 \rceil \log_d \lceil n/2 \rceil + \lfloor n/2 \rfloor \log_d \lfloor n/2 \rfloor \geq n \log_d (n/2)$. Since the depth of a species in D is equal to the number of times the species has been moved one level down in D, and since m experiments generate at most $6m/(d-1)$ moves and we perform at most n additional moves, we get the inequality

$$n \log_d (n/2) \ \leq \ 6m/(d-1) + n \ ,$$

from which the lower bound $m \geq (d-1)n(\log_d(n/2) - 1)/6$ follows. □

References

1. A. Andersson. Improving partial rebuilding by using simple balance criteria. In *Proc. 1st Workshop on Algorithms and Data Structures (WADS)*, volume 382 of *Lecture Notes in Computer Science*, pages 393–402. Springer-Verlag, 1989.
2. A. Andersson and T. W. Lai. Fast updating of well-balanced trees. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 447 of *Lecture Notes in Computer Science*, pages 111–121. Springer-Verlag, 1990.
3. A. Borodin, L. J. Guibas, N. A. Lynch, and A. C. Yao. Efficient searching using partial ordering. *Information Processing Letters*, 12:71–75, 1981.
4. G. S. Brodal, S. Chaudhuri, and J. Radhakrishnan. The randomized complexity of maintaining the minimum. *Nordic Journal of Computing, Selected Papers of the 5th Scandinavian Workshop on Algorithm Theory (SWAT)*, 3(4):337–351, 1996.
5. G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. Technical Report BRICS-RS-01-1, BRICS, Department of Computer Science, University of Aarhus, 2001.
6. S. K. Kannan, E. L. Lawler, and T. J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21:26–50, 1996.
7. M. Y. Kao, A. Lingas, and A. Östlin. Balanced randomized tree splitting with applications to evolutionary tree constructions. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 184–196, 1999.
8. A. Lingas, H. Olsson, and A. Östlin. Efficient merging, construction, and maintenance of evolutionary trees. In *Proc. 26th Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 544–553. Springer-Verlag, 1999.
9. C. G. Sibley and J. E. Ahlquist. Phylogeny and classification of birds based on the data of DNA-DNA-hybridization. *Current Ornithology*, 1:245–292, 1983.