CHAPTER 1

AUTOMATED PLAUSIBILITY ANALYSIS OF LARGE PHYLOGENIES

David Dao[1], **Tomáš Flouri**[2], **Alexandros Stamatakis**[1,2]
[1]Karlsruhe Institute of Technology, Institute for Theoretical Informatics, Postfach 6980, 76128 Karlsruhe
[2]Heidelberg Institute for Theoretical Studies, Germany

1.1 INTRODUCTION

Disentangling the evolutionary history and diversity of species has preoccupied mankind for centuries. Ever since Darwin's work on evolutionary theory [7], evolutionary trees or phylogenies are typically used to represent evolutionary relationships between species.

Although phylogenies have been used for almost 150 years, statistical, computational, and algorithmic work on phylogenies - often referred to as computational phylogenetics - is barely 50 years old. The analysis of phylogenetic trees does not only serve human curiosity but also has practical applications in different fields of science. Phylogenies help to address biological problems such as, drug design [5], multiple sequence alignment [8], protein structure

[27], gene function prediction [14], or studying the evolution of infectious diseases [12].

In the past decade, the molecular revolution [24] has led to an unprecedented accumulation of molecular data for inferring phylogenies. Public databases such as GenBank [3] grow exponentially, which, in conjunction with scalable software, allows for computing extremely large phylogenies that contain thousands or even tens of thousands of species (see [13, 28], for instance). In practice, however, our ability to infer such comprehensive trees resurrects old problems and gives rise to novel challenges in computational phylogenetics.

First of all, reconstructing the phylogeny that best fits the data is a combinatorial optimization problem. The number of phylogenetic trees increases super-exponentially with the number of taxa [9]. For example, there are three distinct unrooted trees for four species. However, for only 23 species there already exist 1.32×10^{25} possible unrooted trees, which is ten times the number of estimated stars in the universe.

Due to continuous progress in processor technology, for instance in transistor size [4] and parallel processing techniques [17] as well as phylogenetic analysis software such as RAxML [29] and MrBayes [25] biologists can now routinely reconstruct trees with about 100 to 1 000 species for their studies. However, for even larger phylogenies with up to tens of thousands of species, we are not sure if we are obtaining a plausible, let alone correct answer, given the literally astronomical size of the tree search space. Reconstructing large phylogenies is particularly difficult when the alignment length remains constant. In other words, accuracy decreases as we add taxa while keeping the number of sites in the alignment fixed [16, 23].

A further problem is the non-uniform sampling of data. Although genomic data is becoming available at an increasing rate, many species are undersampled because it is difficult to obtain or collect the sequencing samples. For example, prokaryotic organisms that have small genomes, or model organisms as well as model genes are sampled more frequently than other species.

For that reason, increasing the number of taxa in a data set typically also increases the amount of missing data [26], which leads to potentially biasing the results, and, therefore, to decreased phylogenetic accuracy [30]. So given a large phylogeny, how can we assess the biological plausibility of such a large tree?

Visual inspection to assess the plausibility of trees with more than 50 000 taxa is not an option because (i) it is impossible to do so for humans and (ii) there are only but a few tools available for visualizing such large phylogenies.

One can follow two avenues to address the plausibility assessment problem: either devise novel visual tools for phylogenetic data exploration or design algorithms for automated plausibility analysis. Here we focus on the latter idea.

We introduce a new approach to assess the plausibility of large phylogenies by computing all pairwise topological *Robinson-Foulds* (RF) distances [22] of a 55 000 taxon tree of plants [28], for instance, and a set containing a large number of substantially smaller reference trees. These small reference trees

are available in curated databases, such as, STBase [15], for instance and comprise a subset of taxa of the large tree. The underlying assumption is that, the small trees are substantially more accurate. Hence, the lower the average RF distance, or any other reasonable topological distance, between all small reference trees and the large tree is, the more plausible the large tree will be. While we use a reference database containing a billion of small reference trees one could also use reference trees from the literature or from the treebase database [21].

Our main contribution is the design and production-level implementation in RAxML[1] of an effective algorithm for extracting induced subtrees from the large tree with respect to the taxon set of the small reference tree. This step is necessary to compute the RF distance.

The rest of this chapter is organized as follows: Initially, we discuss the preliminaries and present the formal problem description. Subsequently, we first present a naïve and then, an effective algorithm for inducing subtrees. Next, we provide an experimental evaluation of both algorithms using simulated and real data from STBase [15] and studies by Smith *et al.* [28]. Finally, we present a brief summary and conclusion.

1.2 PRELIMINARIES

A *rooted tree* is a connected, directed and acyclic graph with an *internal node* r (i.e. $deg(r) > 2$) designated as the *root node*. We say a node v has *height* $h(v)$ if the length of the longest path from v to some leaf is $h(v)$.

We denote a node v of a tree T as *central* if there is no other node u such that the length of its longest undirected path to a leaf is shorter than that of v. Trivially, a tree can have at most two central nodes, since, otherwise there would exist a cycle.

We say that node w is the *Lowest Common Ancestor* (LCA) of nodes u and v *if and only if* (iff) u and v are both descendants of w and there is no node w' with descendants u and v such that $h(w') \leq h(w)$. We denote the LCA of nodes u and v by $lca(u, v)$. We further denote the path from node u to v in a tree T by $u \rightsquigarrow v$.

An *unrooted tree* is a connected, undirected and acyclic graph with no root node. The notion of node height (and hence LCA) is not defined in unrooted trees. Therefore, we henceforth imply that we are dealing with rooted trees when we use the terms node height and LCA. We further only consider rooted binary trees and unrooted trees that only consist of inner nodes of degree 3 and leaves (degree 1), that is, strictly bifurcating trees.

Definition 1.1 (Euler tour of a rooted tree) *The Euler tour of a rooted tree is a sequence of nodes formed by executing the following* Step *with the root node r as input parameter.* Step(v): *List/print node v. If v is not a leaf:*

[1] `https://github.com/stamatak/standard-RAxML`

call Step *with the left child as parameter, then call* Step *with the right child as parameter and finally list/print* v.

The length of the Euler tour for a rooted binary tree of n nodes is exactly $2(n-1)$. The number of leaves in such a tree is $\frac{n+1}{2}$. We list each leaf only once, each inner node with the exception of the root node three times, and the root node exactly twice. Hence this sums to exactly $2(n-1)$ elements. See Fig. 1.1 for a graphical illustration of the Euler tour on a tree.

Definition 1.2 (Inorder traversal of a rooted tree) *The inorder traversal of a rooted tree is a sequence of nodes formed by executing the following* Step *with the root node* r *as input parameter.* Step(v): *Call* Step *with left-child as parameter, list* v *and call* Step *with right-child as parameter.*

Analogously to the Euler tour, the length of the inorder traversal sequence for a tree with n nodes is exactly n.

Lemma 1.1 *Let* $v_1, v_2, \ldots, v_n$ *be the inorder notation of a tree* T. *For every* $k = 1, 2, \ldots, \lfloor n/2 \rfloor$ *it holds that* $v_{2k} = lca(v_{2k-1}, v_{2k+1})$.

Proof: We prove the lemma by induction on the size of T. We first show that the lemma holds for our base case which is a binary tree with three nodes. Then, under the assumption that our claim holds for all trees with up to m nodes, we prove that the claim also holds for trees with $m+1$ nodes.

- Let T be a tree with root node u and two child nodes v and w. The inorder traversal of T yields the sequence v, u, w and hence the base case holds.
- Assuming that the claim holds for trees with up to m nodes, we now prove that the claim holds for any tree T with $m+1$ nodes. Let u be the root node of T and $u_1, u_2, \ldots, u_k$ its direct descendants. The inorder traversal of T yields the sequence $I(T(u_1)), u, I(T(u_2)), u, \ldots, u, I(T(u_k))$ where $I(T(u_i))$ is the inorder notation of the subtree rooted at the i-th direct descendant of u. Trivially, it holds that $|T(u_i)| < m$, for $1 \leq i \leq k$, and based on our assumption, the claim holds for any node in $I(T(u_i))$. Now, consider the i-th occurrence of u in the sequence. Trivially, we observe that a node from $T(u_i)$ (specifically its rightmost leaf) appears immediately before u and a node from $T(u_{i+1})$ (specifically its leftmost leaf) immediately after u. Since the LCA of any pair (p, q) such that $p \in T(u_i)$ and $q \in T(u_{i+1})$ is u, the lemma holds.

□

For instance, let node u be the parent of two nodes v and w where v is the root node of subtree $T(v)$ and w is the root node of subtree $T(w)$. Let p be the last node in the inorder traversal of $T(v)$ and q the first node in the inorder traversal of $T(w)$. By definition of inorder traversal, p is the rightmost leaf of $T(v)$ and q is the leftmost leaf of $T(w)$. Hence, the LCA of p and q is u.

Definition 1.3 (Preorder traversal of a rooted tree) *The preorder traversal of a rooted tree is a sequence of nodes formed by executing the following* Step *with the root node r as parameter.* Step(v): *List node v. Call* Step *with the left child as parameter. Call* Step *with the right child as parameter.*

As for the inorder traversal, the generated sequence is n elements long.

Next, we define the binary relation $< \subset V^2$ on a tree with nodes drawn from set V, such that $v < u$ *iff* the preorder id of v is smaller than the preorder id of u.

Definition 1.4 (Induced subgraph) *Let T be a tree such that L is the set of its leaves and $L' \subseteq L$ a proper subset of its leaves. We call induced subgraph the minimal subgraph that connects the elements of L'.*

We now give the formal definition of an *induced subtree*.

Definition 1.5 (Induced subtree) *Let $G(T, L')$ be the induced subgraph of a tree T on some leaf-set L'. Remove nodes $v_2, \ldots, v_{q-1}$ if there exist paths of the form $v_1, v_2, \ldots, v_q$ resp. $r, v_2, v_3, \ldots, v_q$ such that r is the root, $deg(v_1) > 2$, $deg(v_q) \neq 2$, and $deg(v_2), \ldots, deg(v_{q-1}) = 2$, and replace the corresponding edges with a single edge (v_1, v_q), resp. (r, v_q).*

Definition 1.6 (Bipartition) *Let T be a tree. Removing an edge from T disconnects the tree into two smaller trees, which we call T_a and T_b. Cutting T also induces a bipartition of the set S of taxa of T into two disjoint sets A of taxa of T_a and B of taxa of T_b. We call a bipartition trivial, when the size of either A or B is 1. These bipartitions are called trivial because, as opposed to non-trivial bipartitions, they do not contain any information about the tree structure; a trivial bipartition A of size 1 occurs in every possible tree topology for S. We also denote by $B(T)$ the set of all trivial bipartitions of tree T.*

Definition 1.7 (Robinson-Foulds distance) *Given a set S of taxa, two phylogenetic trees T_1 and T_2 on S, and their respective sets of nontrivial bipartitions $B(T_1)$ and $B(T_2)$, the RF distance between T_1 and T_2 is $RF(T_1, T_2) = \frac{1}{2}((B(T_1) \setminus B(T_2)) \cup (B(T_2) \setminus B(T_1)))$. In other words, the RF distance is the number of bipartitions that occur in one of the two trees, but not in both. This measure of dissimilarity is easily seen to be a metric [22] and we can compute it in linear time [19].*

1.3 A NAÏVE APPROACH

In the following we introduce the Plausibility-Check algorithm.

The algorithm assesses whether a comprehensive phylogenetic tree T is plausible or not by comparing it to a set of smaller reference trees that contain

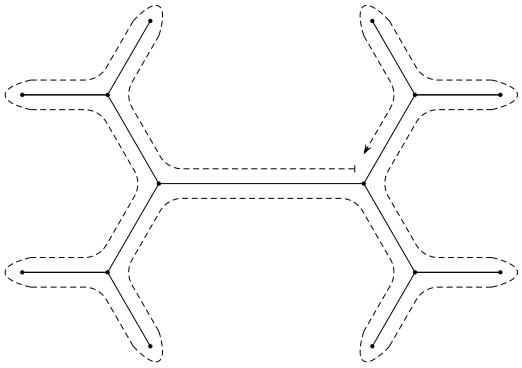

Figure 1.1 Euler traversal of a tree

Algorithm 1: PLAUSIBILITY-CHECK

Input : Tree T of n nodes, set $F = \{t_1, t_2, \ldots, t_m\}$ of small reference trees that contain a proper subset of the taxa in T

Output: m pairwise RF distances $RF(T, F) = \{r_1, r_2, \ldots, r_m\}$ between induced tree $T|t_i$ and t_i

```
B(T) ← EXTRACT-NON-TRIVIAL-BIPARTITIONS(T)
for i ← 1 to m do
    ▷ Extract leaf-set L'_i from t_i
    B(T|t_i) ← COMPUTE-INDUCED-SUBTREE-NBP(T, L'_i)
    B(t_i) ← EXTRACT-NON-TRIVIAL-BIPARTITIONS(t_i)
    r_i ← RF-DISTANCE(B(T|t_i), B(t_i))
end
Let RF(T, F) = r_1, r_2, ..., r_m
```

a proper subset of taxa of T. We denote an induced tree as $T|t_i$ and read it as the tree induced by the taxon set of t_i in T.

PLAUSIBILITY-CHECK (Algorithm 1) takes as input parameters a large tree T and a set F of small reference trees. It is important to ensure that trees in F only contain proper subsets of the taxa in T. In a preprocessing phase, the algorithm extracts all bipartitions of T, which we denote as $B(T)$ and stores them in a hash table. Then, the algorithm iterates over all small trees t_i in F and for every small tree t_i it extracts the corresponding leaf-set L'_i. After obtaining L'_i, PLAUSIBILITY-CHECK computes the induced subtree $T|t_i$, its bipartitions $B(T|t_i)$, and hence the Robinson-Foulds distance for $T|t_i$ and t_i. The algorithm finishes when all small trees have been processed and returns a list of m pairwise RF distances.

EXTRACT-NON-TRIVIAL-BIPARTITIONS (Algorithm 2) calculates all non-trivial bipartitions of T. The implementation requires $\mathcal{O}(n^2)$ time for travers-

Algorithm 2: EXTRACT-NON-TRIVIAL-BIPARTITIONS

Input : Tree T of n nodes
Output: List of all non-trivial bipartitions $B(T)$ of T

```
▷ Extract all bipartitions from T
Let B(T) = b_1, b_2, ...b_{n-3} be the list of non-trivial bipartitions of T
```

ing the tree and storing all bipartitions in a suitable data structure, typically, a hash table. For further implementation details see Chapter 4.1 in [18].

Algorithm 3: COMPUTE-INDUCED-SUBTREE-NBP

Input : List of all non-trivial bipartitions $B(T) = b_1, b_2, \ldots, b_{n-3}$ of T
Leaf-set L'
Output: List of all non-trivial bipartitions $B(T|t_i)$ of induced tree $T|t_i$

```
▷ Iterate through all bipartitions B(T)
for i ← 1 to |B(T)| do
    ▷ Filter bipartition b with L'
    forall the taxa in b_i do
        if taxon is in L' then
            b'_i ← taxon
        end
    end
end
```

COMPUTE-INDUCED-SUBTREE-NBP is a naïve approach to extract the induced subtree $T|t_i$ and return its non-trivial bipartitions. These steps are described in Algorithm 3. It iterates through all bipartitions b_i of T stored in the hash table, and generates the induced non-trivial bipartitions b'_i for $T|t_i$ by deleting the taxa which are not present in the leaf-set L'. The resulting induced bipartitions are then re-hashed. Therefore, COMPUTE-INDUCED-SUBTREE-NBP has a complexity of $\mathcal{O}(n^2)$, where n is the number of leaves of T. Note that, we can reduce the complexity to $\mathcal{O}(\frac{n^2}{w})$ using a bit-parallel implementation, where w is the vector width of the target architecture (e.g., 128 bits for architectures with SSE3 support).

In summary, PLAUSIBILITY-CHECK has an overall complexity of $\mathcal{O}(\frac{n^2mk}{w})$, where n is the size of the large tree, k is the number of small trees in the set F and m is the average size of the small trees.

1.4 TOWARDS A FASTER METHOD

In this section we present a novel method for speeding up the computation of induced subtrees from a given leaf-set. The key idea is to root the large tree

at an inner node and compute the LCA of each and every pair of leaves in the leaf-set. We can then build the induced subtree from the leaves and the LCAs. Although at first glance this may seem computationally expensive, with a series of lemmas we show that it is sufficient to compute the LCAs of only specific pairs of leaves. In fact, we only have to compute the LCAs of $m-1$ particular pairs, and with a special type of preprocessing, we can compute the LCA of each pair in $\mathcal{O}(1)$ time.

The induced subgraph for a leaf-set of size two is a path of nodes and edges from one taxon to the other, and hence the induced tree is a simple line segment with the two taxa as end-points. Therefore, in the rest of the text we only consider leaf-sets of three taxa and more. For an arbitrary leaf-set L', we partition the set V of vertices of the induced subgraph into three disjoint sets

$$V = V_1 \uplus V_2 \uplus V_3$$

such that $V_i = \{\, v \mid v \in V,\ \mathit{deg}(v) = i \,\}$. From the properties of unrooted binary trees we obtain that the size of V_3 is exactly $|L'| - 2$.

In the next lemma we show that all nodes in V_3 are LCAs of some pairs of leaves in L' when rooting the tree at an arbitrary inner node (Lemma 1.2. In fact, V_3 consists of all LCAs of all possible pairs of leaves in L' except possibly at most 1 LCA, which will be in V_2. We prove this last claim in lemma 1.3.

Lemma 1.2 *Let $G(T)$ be the induced subgraph of an unrooted tree T. Rooting T at an arbitrary inner node r allows us to compute V_3 from the LCAs of pairs of leaves in L'.*

Proof: By contradiction.

- Let us assume there exists a node v in V_3 that is not the lowest common ancestor of any two nodes from L'. Because $\mathit{deg}(v) = 3$, in the rooted tree exist exactly two paths $v \rightsquigarrow u$ and $v \rightsquigarrow w$, where $u, w \in L'$. Now let $p = \mathit{lca}(u, w)$ be the least common ancestor of u and w, $r \rightsquigarrow u$ and $r \rightsquigarrow w$ the two paths leading from root r to u and w, and $r \rightsquigarrow p$ their common path. However, $p \neq v$ implies a cycle in the subgraph.

Therefore, any node in V_3 is the LCA of two leaves in L'. □

Fig. 1.2 portrays any node v from set V_3, that is, the root node of a rooted tree with three subtrees t_1, t_2 and t_3. Since v is in V_3 we know that all three subtrees t_1, t_2 and t_3 contain at least one leaf from L'.

The next lemma proves that V_3 is the set of all LCAs for all pairs of leaves from L', except possibly at most one LCA v. This node is part of V_2 (of degree 2) and results from rooting the tree at a node that is in V_2 (in which case v is the root) or at a node that does not appear in V.

Lemma 1.3 *There may exist at most one node in V_2 that is the LCA of two leaves from L'. That node appears* if and only if *the root is not part of the induced subgraph or if it is the root itself.*

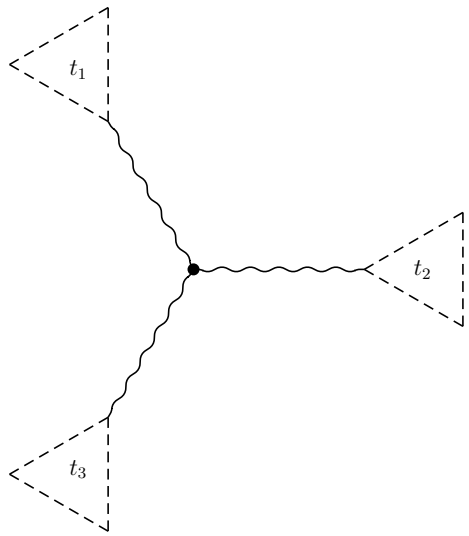

Figure 1.2 Node from V_3

Proof: First we show a necessary condition that must hold for a node to be in V_2 and to simultaneously be an LCA of two leaves. Then, we show that only one such node exists.

- An internal node v (as depicted in Fig. 1.2) which is the LCA of two leaves after rooting the tree at an arbitrary point, ends in V_2 only if one of the subtrees, for instance t_1, does not contain leaves from L'. Moreover, the root must either be at node v or be in t_1.

- Now, assume that there exists a node v' in t_3 or t_2 that is an LCA and belongs to V_2. This node must have degree 3 in the rooted tree, and hence connect three subtrees t'_1, t'_2 and t'_3. By definition, two of the subtrees must contain leaves from L' and the third subtree must not (and must contain the root), such that node v' is in V_2. However, this is a contradiction as the third subtree is either the subtree that contains t_1, t_2 and v (in case v' is in t_3) or t_1, t_3 and v (in case v' is in t_2).

□

To generate the induced tree from the induced subgraph $G(T)$, we remove all nodes from V_2 and replace all edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ formed by paths $v_1, v_2, \ldots v_n$ such that $v_1, v_n \notin V_2$ and $v_i \in V_2$, for all $1 < i < n$, is represented by a single edge (v_1, v_n). We have already shown that it is sufficient to compute set V_3 which, together with L', can generate the induced tree. Therefore, computing $|L'| - 2$ LCAs is sufficient to induce the tree for

leaf-set L'. However, there exist $\frac{|L'|(|L'|-1)}{2}$ pairs of leaves. The main question now is how to choose the pairs for computing the unique $|L'|-2$ LCAs.

Let C denote the set of all LCAs of all pairs of a leaf-set L', that is,

$$C = \{\, p \mid p = lca(u,v), u \neq v, u, v \in L' \,\}.$$

The following lemma proves a fundamental property of LCA computation using the preorder notation. For three nodes u, v and w, where u appears before v and v before w in the preorder traversal of a rooted tree T, we show a transitive property which dictates that knowing the LCAs p of (u,v) and q of (v,w) is sufficient to determine the LCA of (u,w). In fact, the LCA of (u,w) is the node that appears first in the preorder traversal of T between node p and node q.

This property allows us to formulate the main theorem which proves that there exist at most $L'-1$ unique LCAs for the leaf-set L' (and a rooted tree), and states which pairs of leaves to use for obtaining the required LCAs. In the following we denote the preorder identifier of a leaf v as $pid(v)$.

Lemma 1.4 *Let $u, v, w \in L'$ such that $u < v < w$. It holds that $lca(u,w) = v'$, such that $pid(v') = \min(pid(p), pid(q))$ where $p = lca(u,v)$ and $q = lca(v,w)$.*

Proof: Proof by contradiction. Let r be the root of the tree. Let us assume that $p = lca(u,v)$ and $q = lca(v,w)$. By definition of the LCA, q may appear only along the path $r \rightsquigarrow p$ or $p \rightsquigarrow v$, that is, along the path $r \rightsquigarrow v$. We split the proof in two cases: Node q appears on either $r \rightsquigarrow p$ or $p \rightsquigarrow v$.

- In case q appears along $p \rightsquigarrow v$. Let us assume that $v' = lca(u,w)$ is not p. It can then appear along the path $r \rightsquigarrow u$. If it appears anywhere except p we have a cycle. Therefore, $lca(u,w) = p$ and it holds that $pid(p) = \min(pid(p), pid(q))$.
- In case q appears along $r \rightsquigarrow p$. Lets assume that $v' = lca(u,w)$ is not q. It can appear along the path $r \rightsquigarrow u$. If it appears anywhere except p we have a cycle. Therefore, $lca(u,w) = q$ and it holds that $pid(q) = \min(pid(p), pid(q))$.

The lemma holds. □

Specifically, with the next theorem we show that computing the set

$$C' = \{\, p \mid p = lca(u,v), u < v, u, v \in L', \nexists w : u < w < v \,\}$$

is not only sufficient, but that $C' = C$.

Theorem 1.1 *Given leaves $v_1, v_2, \ldots, v_n$ such that $v_i < v_{i+1}$ for $1 \leq i < n$, it holds that $lca(v_j, v_k) = u$ and $pid(u) = \min(pid(u_j), pid(u_{j+1}), \ldots, pid(u_{k-1}))$ for $1 \leq j < k \leq n$ and $u_i = lca(v_i, v_{i+1})$ for $j \leq i < k$.*

Proof: By strong induction on the range

- Let $k - j = 2$. The claim holds as shown by Lemma 1.4 and this forms our base case.
- Let m be a positive integer greater than 2, and let us assume that the claim holds for $k - j \leq m$. This forms our induction hypothesis and we must now prove that the claim holds for $m + 1$.
- Let $k - j = m + 1$. From this interval let us consider nodes v_j, v_{k-1} and v_k. From the induction hypothesis we obtain that $u_\ell = lca(v_j, u_{k-1})$ such that $pid(u_\ell) = \min \bigcup_{i=j}^{k-1} (pid(u_i))$. We also have that $lca(v_{k-1}, v_k) = u_{k-1}$. From Lemma 1.4 we can easily obtain the desired proof that $lca(v_j, v_k)$ is the node that has the smallest preorder identifier between u_ℓ and u_{k-1} and hence our claim holds.

□

Theorem 1.1 implies that it is sufficient to sort the leaf-set in ascending order according to the preorder identifiers of the corresponding leaves in the rooted tree. Then, for the sequence $u_1, u_2, \ldots, u_{|L'|}$ of leaves, one can compute the LCA of $|L'| - 1$ pairs (u_i, u_{i+1}), for $i \leq 1 < |L'|$, to obtain the desired $|L'| - 1$ unique LCAs. Note that, in case the selected root node is in V_3, it will appear twice; once as the LCA of two leaves from t_1 and t_2 (see Fig. 1.2), and once as the LCA of two leaves from t_2 and t_3. On the other hand, if the root node is not in V_3, then we obtain one additional LCA from V_2, which will not appear in the induced tree.

So far, we have proven that, given a leaf-set L', we can compute all common ancestors of all pairs of leaves in L' by simply computing the LCA of only $|L'| - 1$ pairs. Each pair consists of two vertices u_i and u_j so that there is no vertex u_k such that $u_i < u_k < u_j$. Moreover, we have shown that there are exactly $|L'| - 2$ common ancestors of degree 3 which will be present in the induced tree. In case the root node (when rooting the unrooted tree) is not part of these nodes, we will obtain one additional common ancestor which will have degree 2 in the induced subgraph, but that will not appear in the induced tree.

1.5 IMPROVED ALGORITHM

We are now in a position to present and explain the actual algorithm for inducing a tree given a leaf-set L' and an unrooted tree T. We divide the algorithm into two steps — preprocessing and inducing the trees. In the preprocessing step we use a dedicated data structure created from tree T that we can query for LCAs in constant time. In the second step, we show how to query for LCAs and build the resulting induced tree.

Moreover, if we are given several leaf-sets at once, we can implement the algorithm with one of the following two variants, each of which has different asymptotic space and time complexity. The first requires loading all leaf-sets in memory prior to computing the induced subtrees, and hence runs in

$\Theta(n + km)$ time and space. The second variant does not require pre-loading, uses only $\Theta(n)$ space and runs in $\mathcal{O}(n + m \log m)$ time.

1.5.1 Preprocessing

For fast LCA queries, it is necessary to root T at an arbitrary point and then create a data structure that allows for $\mathcal{O}(1)$ queries. To achieve this we consider the close relation between LCA computation and *Range Minimum Queries* (RMQ). As shown in [2], we construct a sequence of node identifiers which correspond to an Euler tour of the rooted tree. Identifiers are assigned to nodes upon the first visit and in increasing order. We then preprocess this sequence for RMQs. The size of the succinct preprocessed data structure for a tree of n nodes is at most $2n + o(n)$ bits [11]. Algorithm 4 lists the preprocessing phase. We omit the details of constructing a RMQ structure and rather point the interested reader to the available literature [10, 11].

Algorithm 4: PREPROCESS-ROOTED-TREE

Input : Tree T of n nodes
Output: Preprocessed data structure $RMQ(T)$

```
▷ Root tree T
if T is unrooted then
    Root T at an arbitrary node
end
▷ Build Euler tour of T
Let E(T) = s_1, s_2, ..., s_{2n-1} be the Euler tour of T
▷ Prepare a RMQ data structure
Let P(T) = pid(s_1), pid(s_2), ..., pid(s_{2n-1}) be the list of preorder
identifiers of E(T)
Let RMQ(T) = RANGE-MINIMUM-QUERY-PREPROCESS(P(T))
```

1.5.2 Computing lowest common ancestors

Once we have constructed the preprocessed data structure via the Euler tour, it is possible to compute the LCA of two leaf nodes in time $\mathcal{O}(1)$. To do so, we will need one additional data structure. Let L be the set of leaves of the rooted tree T of n nodes which we preprocessed. The new data structure represents the mapping

$$f : L \to \langle 1, 2(n-1) \rangle$$

of the position where each leaf appears for the first time in the Euler tour. We can now compute the LCA of two leaves $u, v \in L$ by finding the node with the lowest preorder identifier in the Euler tour in the range $\langle i, j \rangle$, where $i = \min(f(u), f(v))$ and $j = \max(f(u), f(v))$. With the preprocessed Euler

tour, we can compute this minimum value, and hence the ancestor of u and v, in time $\mathcal{O}(1)$ using RMQs.

1.5.3 Constructing the induced tree

Given a list L' sorted (in ascending order) by the preorder identifiers of the nodes in tree T, we determine the common ancestor of every pair of adjacent nodes. Let $v_1, v_2, \ldots, v_{|L'|}$ be the list of nodes in L' such that $v_i < v_{i+1}$ for $1 \leq i < |L'|$. We compute the LCA of every pair (v_i, v_{i+1}) and construct a new sequence $I = v_1, lca(v_1, v_2), v_2, lca(v_2, v_3), v_3, \ldots, v_{|L'|-1}, lca(v_{|L'|-1}, v_{|L|}), v_{|L'|}$ of size $2|L'| - 1$ by interleaving the LCA node between each pair of adjacent nodes. These steps are described in Algorithm 5.

Algorithm 5: COMPUTE-INDUCED-TREE

Input : Preprocessed RMQ structure $RMQ(T)$
Leaf-set L'
Mapping $f : L' \to \langle 1, \ldots, n \rangle$
Output: Induced unrooted tree $T|t_i$

```
▷ Sort leaf-set according to preorder traversal identifiers of T
Let L'_S = (u_1, u_2, ... u_|L'|) such that u_{i-1} < u_i < u_{i+1}, for 1 < i < |L'|
▷ Compute common ancestors
for i ← 2 to |L'| do
    c_i ← lca(u_{i-1}, u_i)
end
▷ Sort the resulting nodes and construct the induced tree
Let V' = L'_S ∪ ⋃_{i=2}^{|L'|} (c_i) and name the nodes as u_{1,2}, ..., u_|V'|
Let V'_S = (u_1, u_2, ... u_|V'|) such that u_{i-1} < u_i < u_{i+1}, for 1 < i < |L'|
T|t_i ← BUILD-INDUCED-TREE(V'_S)
```

The resulting sequence corresponds to the inorder notation of the induced rooted tree (see Lemma 1.1). While we can construct the induced tree in $\mathcal{O}(|L'|)$ time directly from the inorder sequence, we will show a different approach that requires an additional sorting step and is substantially simpler to explain.

By sorting the inorder sequence in ascending order we obtain the preorder notation of the induced rooted tree. The first node in the sequence is the root node, and we can build the induced tree by applying Algorithm 6 on the preorder sequence, which we include for completeness. The algorithm is simple and builds the induced rooted tree in a depth-first order. After building the tree and in case the root node is of degree 2 (see Lemma 1.3), we remove the root and connect the two children by an edge.

Note that, as an alternative, it is possible to extract the required bipartitions after sorting the sequence without building the actual tree. Using Algorithm 7 we can calculate the non-trivial bipartitions based upon the pre-

Algorithm 6: BUILD-INDUCED-TREE

Input : Sorted list of nodes $(u_1, u_2, \ldots, u_n)$
Output: Induced unrooted tree

```
▷ Check whether the root is of degree 2 or 3
r ← New-Node
push r; push r
if u_1 = u_2 then
    push r
    start ← 3
    deg(r) ← 3
else
    start ← 2
    deg(r) ← 2
for i ← start to n do
    pop p
    q ← New-Node
    Append-Child(p, q)
    if u_i is a leaf then
        push q; push q
        deg(q) ← 3
    else
        deg(q) ← 1
if deg(r) = 2 then
    Connect the two children of r with an edge and remove r
```

order sequence. The algorithm determines all subtrees of the induced tree $T|t_i$ and extracts the corresponding bipartitions by separating these subtrees from $T|t_i$.

■ **EXAMPLE 1.1**

Compute the induced tree, given the query tree in Fig. 1.3 and a leaf-set L' that consists of the leaves marked in blue.

First, we transform the unrooted tree into a rooted one by designating one node as root. We then assign a preorder traversal identifier to each node as shown in Fig. 1.4, starting from 0. The numbers at each node in the figure indicate the preorder traversal identifiers assigned to that particular node. For this example, the Euler traversal is the sequence

Algorithm 7: EXTRACT-BIPARTITIONS-PREFIX

Input : Sorted list of nodes $(u_1, u_2, \ldots, u_n)$
Output: Non-trivial bipartitions from induced subtree $B(T|t_i) = b_1, b_2, \ldots, b_{n-3}$

```
k = 0
for i ← n to 1 do
    if k < number of splits then
        V[i] ← 1
        if u_i is not a leaf then
            for j ← 1 to deg(u_i) do
                V[i] ← V[i] + V[i + V[i]]
            for j ← 1 to V[i] do
                if u_{i+j} is a leaf then
                    Add u_{i+j} into b_i
                else
                    Add all nodes from b_{i+j} into b_i
                    j ← j + V[i + j]
            k = k + 1
```

0	1	2	3	4	3	5	3	2	6	7	6	8	6	2	1
9	10	11	10	12	10	9	13	14	13	15	13	9	1	0	16
17	18	19	18	20	18	17	21	22	21	23	21	17	16	24	25
26	25	27	25	24	28	29	28	30	28	24	16	0	31	32	33
34	33	35	33	32	36	37	36	38	36	32	31	39	40	41	40
42	40	39	43	44	43	45	43	39	31						

which is preprocessed for RMQ queries. We then sort the sequence of leaves of L' in ascending order, that is,

$$8, 20, 22, 23, 26, 38, 41$$

Then, we compute the LCAs of node pairs

$$(8, 20), (20, 22), (22, 23), (23, 26), (26, 38), (38, 41)$$

and obtain the sequence

$$8, 0, 20, 17, 22, 21, 23, 16, 26, 0, 38, 31, 41$$

which represents the inorder notation of the induced tree. We can now build the induced tree directly from this inorder notation, or sort the sequence and build the tree using Algorithm 6. Fig. 1.4 depicts the induced tree.

1.5.4 Final remarks

As stated earlier, it is possible to implement the algorithm in two different ways, depending on the amount of available memory. The difference between

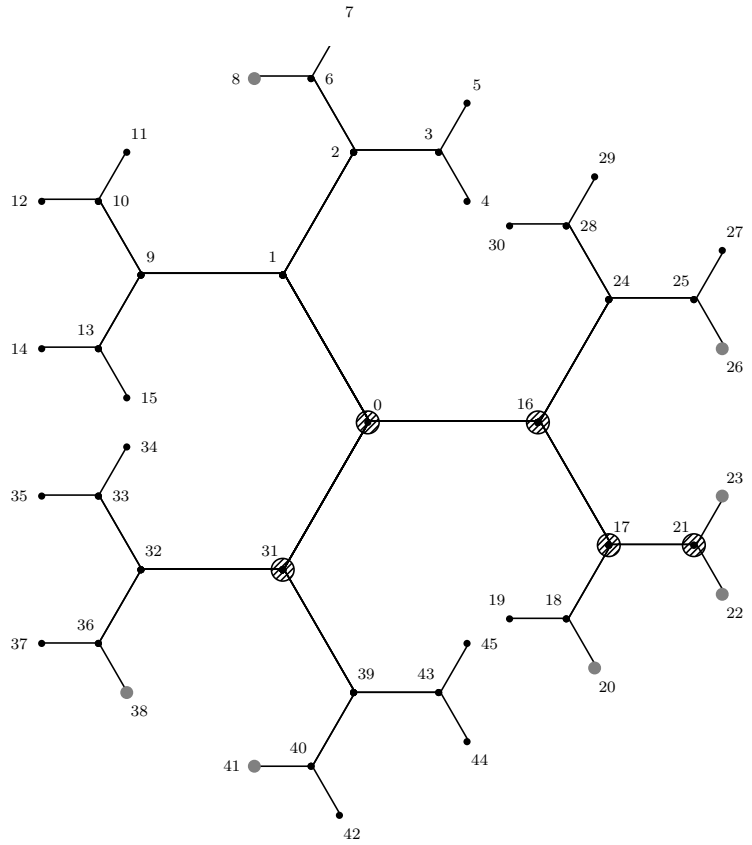

Figure 1.3 Unrooted phylogeny of 24 taxa (leaves). The taxa for which we induce a subtree are marked in gray. Inner nodes which represent the common ancestors and hence the minimum amount of nodes needed to maintain the evolutionary relationships among the selected taxa are shown dashed. The numbers denote the order of each node in the preorder traversal of the tree assuming we root it at node 0.

the two variants is in the way how the initial sorting of each query leaf-set is done.

Let T be a large tree of n nodes and let $L'_1, L'_2, \ldots, L'_k$ be k leaf sets with an average size of m. One can now sort each leaf-set, compute the LCAs from the sorted sequence (and the already preprocessed Euler tour of the query tree) using Algorithm 5, and then apply Algorithm 6 to construct the induced tree. The asymptotic time and space complexity for this variant is $\mathcal{O}(n)$ time and space for preprocessing T and $\mathcal{O}(km \log m)$ time for inducing k trees.

The alternative variant is to avoid sorting each of the k leaf-sets individually. Instead, one can store all of them in memory at the same time and sort them using a bucket sort method. Since the range of values in the k leaf-sets is $\langle 1, n \rangle$, we can sort them all in a single pass in conjunction with the prepro-

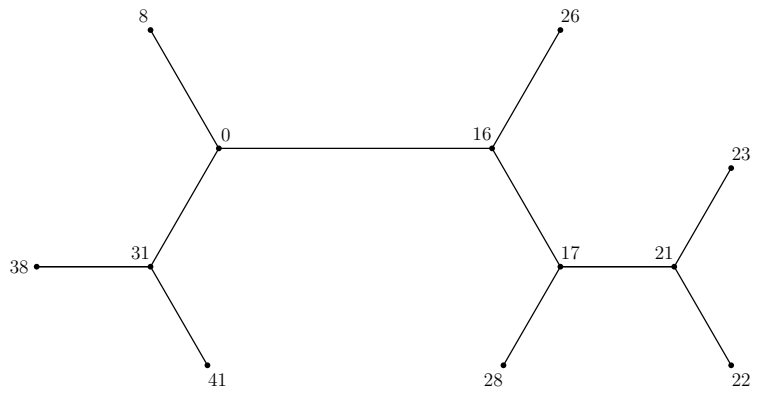

Figure 1.4 Induced tree for Example 1.1

cessing step in $\mathcal{O}(\max(n, km))$ time and space. Thereafter, we can build the k induced trees in $\mathcal{O}(km)$ time, assuming that we construct the induced tree directly from the inorder notation.

1.6 IMPLEMENTATION

In the following, we present a straightforward implementation of the PLAUSIBILITY-CHECK algorithm. We have implemented the algorithm in C as part of RAxML. Furthermore, we address how to efficiently implement the fast method from Section 1.5 for bifurcating unrooted trees.

1.6.1 Preprocessing

First of all, we need to preprocess the large phylogenetic tree by assigning preorder identifiers to every node. Therefore, we root the tree at an arbitrary inner node and traverse it to assign preorder identifiers and store them in an array. We will use this array in the following steps to efficiently look up preorder identifiers for every node.

We now traverse our tree for a second time via an Euler traversal. We can also avoid this second tree traversal by assigning preorder identifiers on the fly during the Euler traversal. However, this method requires additional memory for marking already visited nodes. Note that, the resulting array consists of $4|L| - 5$ elements because the Euler traversal visits $|L| - 3$ inner nodes (all inner nodes except for the root) three times, all other $|L|$ nodes once and the root four times. To further optimize the induced tree reconstruction phase, we use an additional array, which we denote by FASTLOOKUP, that stores the index of the first appearance of each taxon during the Euler tour. This information allows us to speed up RMQ queries in the reconstruction phase and we can also compute it on-the-fly during the Euler traversal.

While we choose to use arrays for storing node information such as preorder identifiers or Euler labels, one could also use hash tables to reduce memory storage or list data structures, for instance.

Based on the Euler tour, we can now construct a RMQ data structure. For this, we use source code developed by Fischer *et al.* [11] which we modify and adapt to our purposes.

1.6.2 Reconstruction

Initially, we extract the leaf set from our small reference tree by traversing the small tree and storing its taxon set in an auxiliary array called SmallTreeTaxa. As before, we denote the number of taxa in the small reference tree by m. In the following, we use SmallTreeTaxa each time we need to iterate through the leaf set of the small tree.

Now, for every taxon in the reference tree we look up at which index position it first appeared in the Euler tour using the FastLookUp array. Because of the auxiliary FastLookUp array, this procedure has a time complexity of $\mathcal{O}(m)$. Without this additional array, we would have to search through the entire Euler tour to find the corresponding indices, which would require $\mathcal{O}(nm)$ time. Thereafter, we sort all resulting indices in ascending order using quicksort. Note that, this is analogous to sorting the preorder identifiers, which is necessary for computing the induced tree as outlined in Section 1.5. By querying the RMQ data structure, we can now find the least common ancestor of two taxa in constant time and reconstruct the induced tree using Algorithm 6.

1.6.3 Extracting Bipartitions

To finally compute the RF distance, we extract all non-trivial bipartitions by traversing the small reference tree and the induced tree using the bipartition hash function which has been thoroughly discussed by Pattengale *et al.* [18].

To reduce memory consumption and to improve running times, we store bipartitions in bit vectors with m instead of n bits. We achieve this, by consistently using the taxon indices from SmallTreeTaxa instead of the original taxon index in the large tree. Bit vectors are well suited for storing sets with a pre-defined number of m elements such as bipartitions. They only need $\Theta(m)$ bits of space and can be copied efficiently with C functions like `memcpy()`. These bit vectors are then hashed to a hash table and can be looked up efficiently.

As stated earlier in Section 1.5, it is possible to extract all non-trivial bipartitions directly from the preorder sequence without relying on an instance of a tree structure, as outlined in Algorithm 7. We deploy this approach because it does not require building the induced tree at all.

However, for both implementation options, we need a mechanism to avoid storing (and thus checking for) complementary bipartitions. To avoid distinct, yet identical representations of one and the same bipartition (the bipartition

and its bit-wise complement), we hash bipartitions in a canonical way. We only hash a bipartition if it contains a specific taxon (in our case the first taxon in SmallTreeTaxa). If our bit vector does not contain the specific taxon, we compute and then hash its complement instead.

1.7 EVALUATION

In the following we describe the experimental setup and the results.

1.7.1 Test Datasets

1.7.1.1 Real-world Datasets For real-world data tests, we used the mega-phylogeny of 55 473 plant species by Smith *et al.* [28]. To obtain a reference tree set, we queried all trees in STBase [15] which are proper subsets of the large tree. Our reference tree set consists of 175 830 trees containing 4 up to 2 065 taxa and is available for download at `http://www.exelixis-lab.org/material/plausibilityChecker.tar.bz2`.

1.7.1.2 Simulated Datasets As large trees, we used 15 trees with 150 up to 2 554 taxa from [20] that are available for download at `http://lcbb.epfl.ch/BS.tar.bz2`. For each large tree, we generated 30 000 corresponding reference trees containing 64 taxa. We used the following procedure to simulate and build the reference trees: First we extract the taxon labels of the large tree. Thereafter, we randomly select a proper subset of these taxa and construct the trees using an algorithm that is similar to Algorithm 6.

Moreover, we also want to assess how long it will take our algorithm to run on a very large number of reference trees for a mega-phylogeny. To this end, we extracted 1 million reference trees with 128 taxa each from the empirical mega-phylogeny with 55 000 taxa.

1.7.2 Experimental Results

All experiments were conducted on a 2.2 GHz AMD Opteron 6174 CPU running 64-bit Linux Ubuntu. We invoked the plausibility check algorithm as implemented in standard RAxML with following command:

```
raxmlHPC-AVX -f R -m GTRCAT -t largetree -z referencetrees -n T1
```

In all experiments, we verified that both algorithms yield exactly identical results.

1.7.2.1 Mega-phylogeny For the mega-phylogeny, we obtain an average relative RF distance of 0.318 (see Table 1.1) between the large tree and the reference trees from STBase. We consider this average topological distance of approximately 32% to be rather low, because of the substantially larger

tree search space for the 55K taxon tree. For a tree with 2 000 taxa there are about $3.00 \times 10^{6\,328}$ possible unrooted tree topologies, whereas for 55 000 taxa there exist approximately $2.94 \times 10^{253\,380}$ possible unrooted tree topologies. In other words, the tree search space of the 55K taxon tree is about $10^{247\,052}$ times larger than for the 2 000 taxon tree. Taking into account that, different procedures were used to automatically construct the corresponding alignments, and that, the trees have also partially been constructed from different genes, an average error of around 30% appears to be low. However, the interpretation of these results is subject to an in-depth empirical analysis which is beyond the scope of this paper. Fig. 1.5 illustrates the overall distribution of RF distances, whereas Fig. 1.6 shows the corresponding distribution for the 20 000 largest reference trees. Using our improved algorithm, we can process the 175 830 small reference trees by five orders of magnitude faster than with the naïve algorithm. In total, the naïve algorithm required 67 644s for all reference trees, while the effective algorithm required less than 7.14 s, after a preprocessing time of 0.042 s. If we only consider the inducing steps and ignore the time for parsing every single tree, the naïve algorithm needs 67 640 s for reconstructing the induced tree whereas the effective approach only takes 3.11 s. Hence, the effective algorithm is five orders of magnitude faster than the naïve version.

1.7.2.2 Simulated data The naïve algorithm needs more time for larger phylogenies as discussed in Section 1.3 because it iterates over all taxa of the large tree for each small tree. In contrast to this, our new approach only preprocesses the large tree once. As we show in Fig. 1.7, the run-time of the effective algorithm is independent of the input tree size. It induces the subtree in time that is proportional to the size of each small tree. This yields a significant run-time improvement for our new algorithm (see Table 1.2). In the following, we calculated the speedup by comparing the run times for the inducing step in both algorithms. Fig. 1.8 shows the speedup for the optimized induced subtree version of Plausibility-Check compared to the naïve approach. As theoretically expected, the speedup improves with increasing size of the input phylogeny T. For example, on the large tree with 2 458 tips, the effective approach is about 19 times faster than the naïve algorithm which is consistent with our theory. In each run, the naïve algorithm has to traverse the large tree which is about 40 times the size of the small tree (64 tips),

Average Robinson-Foulds distance	0.318
Total time for inducing (naïve)	67 640.00 s
Total time for inducing (improved)	3.11 s
Total execution time (naïve)	67 643.00 s
Total execution time (improved)	7.14 s

Table 1.1 Test results for a mega-phylogeny of 55 473 taxa

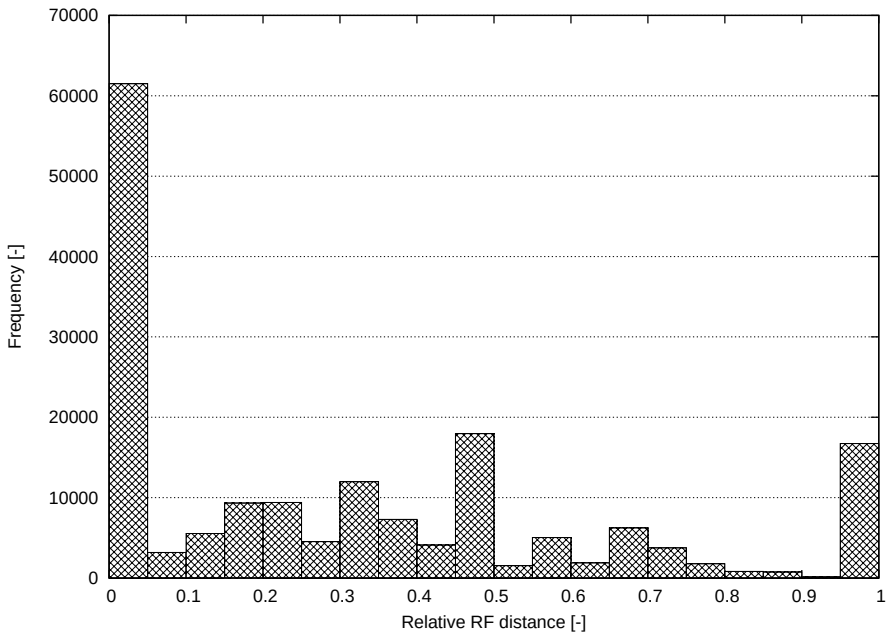

Figure 1.5 Distribution of all relative RF distances between the large megaphylogeny and the reference trees from STBase.

whereas the efficient method only traverses the small reference tree. However, due to additional sorting and traversing of the small tree, we suffer a loss in run-time performance which explains the resulting speedup. If the difference between the size of the large tree and the small reference tree is small, both algorithms will have approximately the same run-time. However, this is not the standard use case for our algorithm. Fig. 1.9 shows the overall execution times for both algorithms, while Fig. 1.10 shows the preprocessing time for the effective algorithm which depends on the size of T. The preprocessing time is negligible compared to the overall execution time. Table 1.3 illustrates the huge differences between the effective and the naïve algorithm on extremely large data sets. For one million reference trees, our naïve algorithm required 113 hours (ca. five days) whereas the effective algorithm required less than 8 minutes.

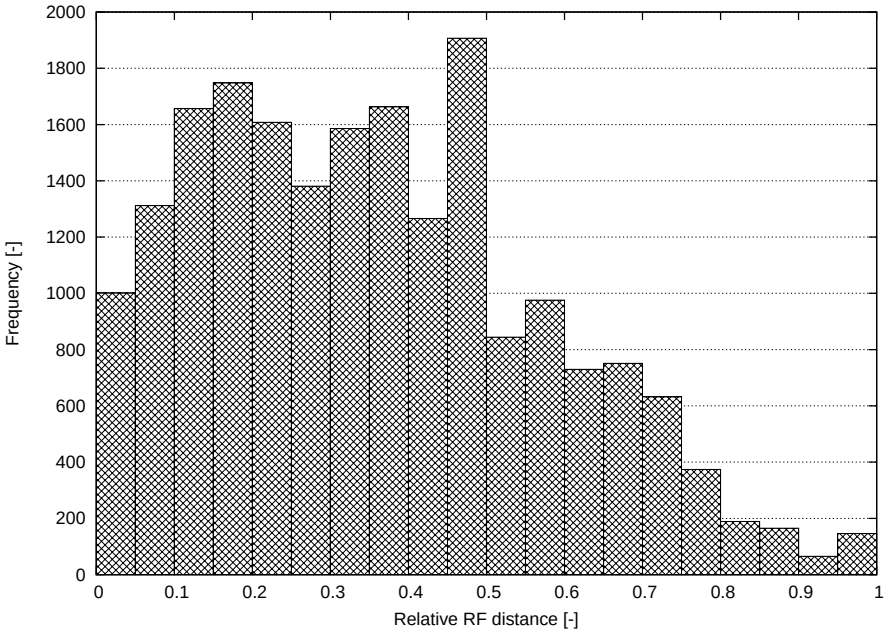

Figure 1.6 Distribution of relative RF distances for the 20 000 largest reference trees (30 up to 2 065 taxa)

# of taxa in large tree	Inducing time (naïve)	Inducing time (improved)	Preprocessing time	Overall execution time (naïve)	Overall execution time (effective)
150	1.805363	2.21387	0.00030	3.200785	3.959420
218	2.173332	2.27510	0.00031	3.614717	3.989973
354	3.318583	2.30837	0.00036	4.935320	4.178407
404	3.683192	2.42039	0.00037	4.904781	4.053480
500	4.318119	2.26976	0.00038	5.648583	3.990615
628	6.077749	2.36570	0.00046	7.312694	3.895842
714	7.063149	2.36753	0.00048	8.399326	3.897443
994	10.290771	2.35056	0.00056	11.840957	4.079138
1 288	16.531953	2.33238	0.00077	18.346817	4.078463
1 481	20.654801	2.44133	0.00080	22.444981	4.134798
1 604	23.317732	2.45706	0.00086	25.385845	4.269186
1 908	29.793863	2.44010	0.00100	31.903671	4.188301
2 000	30.726621	2.43945	0.00106	32.648712	4.050954
2 308	39.535349	2.39014	0.00119	41.739811	4.157518
2 554	46.642499	2.48903	0.00125	48.698793	4.498240

Table 1.2 Test results for different input tree sizes (150 - 2 554 taxa). We executed the algorithm on 30 000 small trees for each run. Each small tree contains exactly 64 taxa.

# of taxa in large tree	55 473
# of small trees	1 000 000
Total time for inducing (naïve)	406 159.00 s
Preprocessing time	0.045 s
Total time for inducing (improved)	238.37 s
Total execution time (naïve)	405 902.00 s
Total execution time (improved)	448.40 s

Table 1.3 Test results for one million simulated reference trees (each containing 128 taxa)

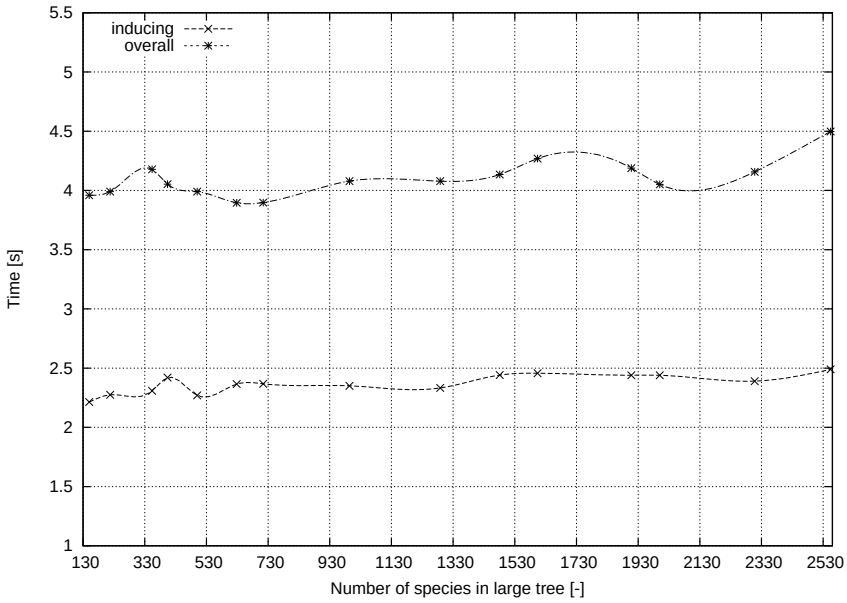

Figure 1.7 Running time of the effective inducing step (dashed) compared to the overall execution time of the effective algorithm (dotted)

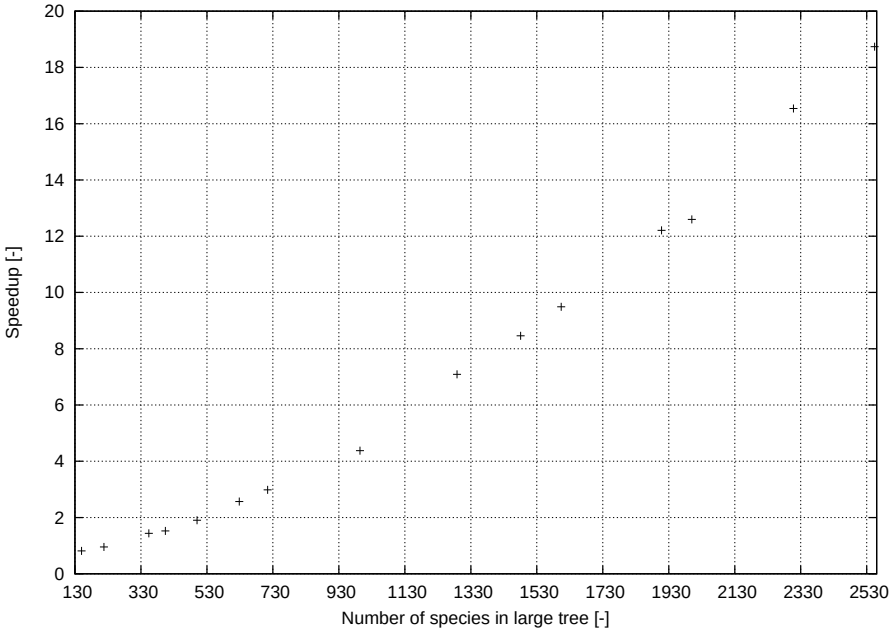

Figure 1.8 Speedup of the effective inducing tree approach. We calculate the speedup by dividing the overall naïve inducing time with the effective inducing time.

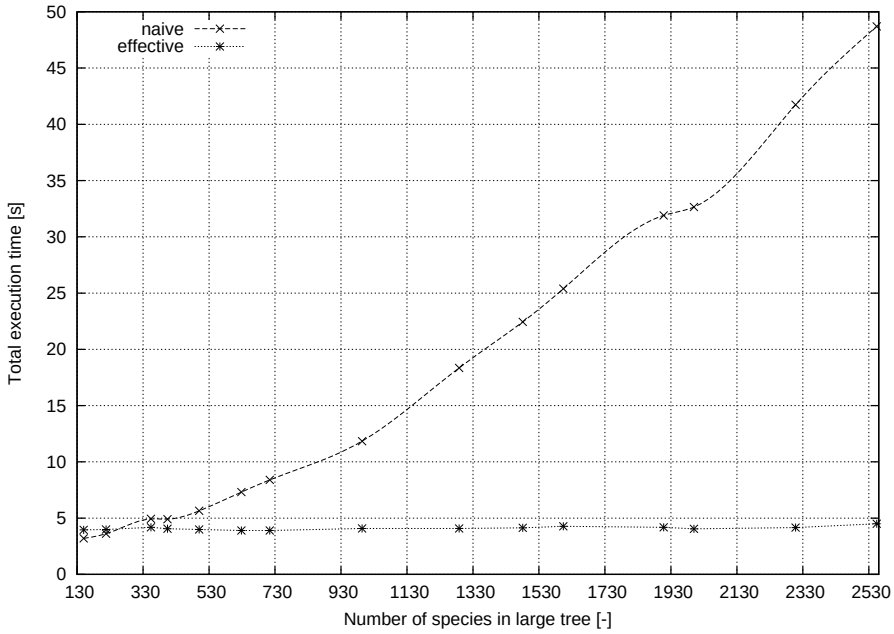

Figure 1.9 Total execution time of the naïve algorithm (dashed) compared to the effective approach (dotted)

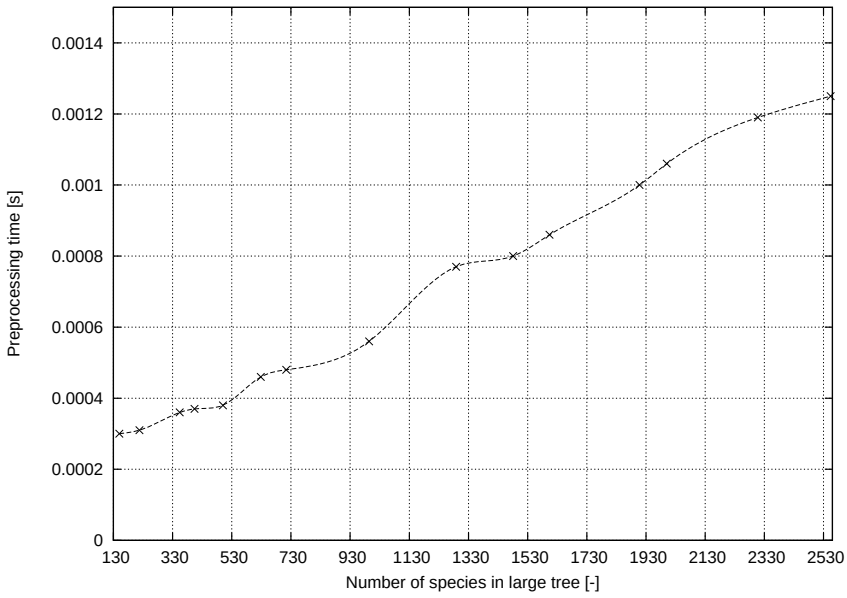

Figure 1.10 Time needed for the preprocessing step of the effective algorithm

1.8 CONCLUSION

In view of the increasing popularity of megaphylogeny approaches in biological studies [13, 28], one of the main challenges is to assess the plausibility of such large phylogenies. Because of the availability of a large number of curated smaller phylogenies, the methods and software we introduce here, allow to automatically assess and quantify the plausibility of such large trees. Moreover, they can be used to compare such large phylogenies to each other by means of their respective average RF distances. Here, we use the RF distance metric, but any other, potentially more advanced, topological distance metric, such as the quartet-distance [6] for instance, can be used.

We consider the average RF distance of 32% we obtained using empirical reference trees for the 55K taxon tree to be surprisingly small with respect to the size of the tree search space. The histograms with the distribution of the RF distances can be used to identify problematic clades in mega-phylogenies. One could also establish an iterative procedure that removes taxa from the mega-phylogeny in such a way that the average RF distance drops below a specific threshold. This may also give rise to novel optimization problems. For instance, one may consider the problem of finding the smallest set of taxa to prune, whose removal yields a 10% improvement of the average RF distance. This kind of optimization problems might also be connected to recent algorithms for rogue taxon identification [1]. Apart from these practical considerations, we showed that our method runs in $\mathcal{O}(km)$ or $\mathcal{O}(km \log m)$ time. This is an important finding because the time complexity, except for the preprocessing phase, is independent of the size of the mostly very large input phylogeny. Our experimental findings are in line with our theoretical results and the implementation exhibits a substantial speedup over the naïve algorithm. Nevertheless, there are still several open problems that need to be addressed. Is it possible to design an algorithm for our method which runs in linear time as a function of the leaf set of the small reference tree? Furthermore, our method examines the extent to which a large phylogeny corresponds to existing, smaller phylogenies. At present, the small trees have to contain a proper taxon subset of the large phylogeny. An open problem is how to handle small trees that contain taxa which do not form part of the large tree.

Finally, we simply do not know if large trees that attain high plausibility scores (low average RF distance) do indeed better represent the evolutionary history of the organisms at hand.

1.9 ACKNOWLEDGMENTS

We would like to thank Mike Sanderson, from the University of Arizona at Tucson, for sharing the problem statement with us and implementing an option to extract small reference trees in STBase.

References

1. Andre J. Aberer, Denis Krompass, and Alexandros Stamatakis. Pruning Rogue Taxa Improves Phylogenetic Accuracy: An Efficient Algorithm and Webservice. *Systematic Biology*, 62(1):162–166, 2013.
2. Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.
3. Dennis A. Benson, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and Eric W. Sayers. Genbank. *Nucleic Acids Research*, 38(suppl 1):D46–D51, 2010.
4. Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011.
5. James R Brown and Patrick V Warren. Antibiotic discovery: is it all in the genes? *Drug Discovery Today*, 3(12):564–566, 1998.
6. David Bryant, John Tsang, Paul Kearney, and Ming Li. Computing the quartet distance between evolutionary trees. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 285–286, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.
7. Charles Darwin. *The origin of species.* Everyman's library. Dent, 1936.
8. Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.
9. Joseph Felsenstein. The number of evolutionary trees. *Systematic Biology*, 27(1):27–33, 1978.

10. Johannes Fischer and Volker Heun. Theoretical and Practical Improvements on the RMQ-Problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *CPM*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006.

11. Johannes Fischer and Volker Heun. A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *ESCAPE*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

12. WM Fitch, RM Bush, CA Bender, K Subbarao, and NJ Cox. The Wilhelmine E Key 1999 invitational lecture. Predicting the evolution of human influenza A. *Journal of Heredity*, 91(3):183–185, 2000.

13. Pablo A. Goloboff, Santiago A. Catalano, J. Marcos Mirande, Claudia A. Szumik, J. Salvador Arias, Mari Källersjö, and James S. Farris. Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics*, 25:211–230, 2009.

14. M. J.A. Eisen, Wu. Phylogenetic analysis and gene functional predictions: phylogenomics in action. *Theoretical population biology*, 61(4):481–487, 2002.

15. Michelle M. McMahon, Akshay Deepak, David Fernández-Baca, Darren Boss, and Michael J. Sanderson. Stbase: One billion species trees for comparative biology. submitted ms. 2013.

16. BernardM.E. Moret, Usman Roshan, and Tandy Warnow. Sequence-length requirements for phylogenetic methods. In Roderic Guig and Dan Gusfield, editors, *Algorithms in Bioinformatics*, volume 2452 of *Lecture Notes in Computer Science*, pages 343–356. Springer Berlin Heidelberg, 2002.

17. J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

18. Nicholas D. Pattengale, Masoud Alipour, Olaf R. Bininda-Emonds, Bernard M. Moret, and Alexandros Stamatakis. How many bootstrap replicates are necessary? In *Proceedings of the 13th Annual International Conference on Research in Computational Molecular Biology*, RECOMB 2009, pages 184–200, Berlin, Heidelberg, 2009. Springer-Verlag.

19. Nicholas D. Pattengale, Eric J. Gottlieb, and Bernard M. E. Moret. Efficiently computing the robinson-foulds metric. *Journal of Computational Biology*, 14(6):724–735, 2007.

20. Nicholas D. Pattengale, Krister M. Swenson, and Bernard M. E. Moret. Uncovering hidden phylogenetic consensus. In Mark Borodovsky, Johann Peter Gogarten, Teresa M. Przytycka, and Sanguthevar Rajasekaran, editors, *ISBRA*, volume 6053 of *Lecture Notes in Computer Science*, pages 128–139. Springer, 2010.

21. William H. Piel, Lucie Chan, Mark J. Dominus, Jin Ruan, Rutger A. Vos, and Val Tannen. TreeBASE v. 2: A Database of Phylogenetic Knowledge. In *e-BioSphere 2009*, 2009.

22. D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

23. Antonis Rokas and Sean B. Carroll. More genes or more taxa? the relative contribution of gene number and taxon number to phylogenetic accuracy. *Molecular Biology and Evolution*, 22(5):1337–1344, 2005.

24. Mostafa Ronaghi, Mathias Uhlén, and Pål Nyrén. A sequencing method based on real-time pyrophosphate. *Science*, 281(5375):363–365, 1998.

25. Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L. Ayres, Aaron Darling, Sebastian Hhna, Bret Larget, Liang Liu, Marc A. Suchard, and John P. Huelsenbeck. Mrbayes 3.2: Efficient bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology*, 2012.

26. Béatrice Roure, Denis Baurain, and Hervé Philippe. Impact of missing data on phylogenies inferred from empirical phylogenomic datasets. *Molecular Biology and Evolution*, 2012.

27. I.N. Shindyalov, N.A. Kolchanov, and C. Sander. Can three-dimensional contacts in protein structures be predicted by analysis of correlated mutations? *Protein Engineering*, 7(3):349–358, 1994.

28. SA Smith, JM Beaulieu, A Stamatakis, and MJ Donoghue. Understanding angiosperm diversification using small and large phylogenetic trees. *American Journal of Botany*, 98(3):404–414, 2011.

29. Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

30. John J. Wiens. Missing data, incomplete taxa, and phylogenetic accuracy. *Systematic Biology*, 52(4):528–538, 2003.