Karlsruhe Institute of Technology

Automated Plausibility Analysis of Large Phylogenies

Bachelor Thesis of

David Dao

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Alexandros Stamatakis
	Dr. Emmanuel Müller
Advisors:	Dr. Tomáš Flouri

Time Period: 30th September 2013 – 30th January 2014

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 23rd January 2014

Abstract

Megaphylogeny approaches in biological studies are becoming increasingly important in modern data-driven biology. However, our ability to infer large phylogenetic trees gives rise to novel challenges in computational phylogenetics. As we add more and more taxa while keeping the number of sites in an alignment fixed, phylogenetic accuracy decreases. Furthermore, it is almost impossible to visually determine whether such large trees 'make sense' or not. Facing these issues, how can we nevertheless assess the biological plausibility of a huge phylogeny? In this thesis, we develop an automated plausibility assessment approach and introduce the algorithm Plausibility-Check which tries to determine whether a comprehensive phylogenetic tree T is plausible or not given a set of substantially smaller reference trees that contain a proper subset of the taxa in T. This is done by comparing the large tree with the smaller and hence more trustworthy, reference trees and by calculating as well as averaging over all topological distances between both. The most compute-intensive operation of the algorithm is the extraction of the induced subtrees from the large tree with respect to the taxon set of the small reference tree which is a prerequisite to compute the topological distance. Therefore, we introduce and implement an effective algorithm for extracting induced subtrees. We denote the number of nodes in the large tree with $L = 2|T| - 2$. The effective algorithm improves the running time of the original, naïve implementation of Plausibility-Check by a factor of $\mathcal{O}(L^2)$. In addition, we prove that the resulting algorithm has an asympotic time complexity of $\mathcal{O}(km)$, where k is the number of reference trees and m is the average reference tree size. Finally, our experiments on simulated and real data using the STBase tree database show an overall running time improvement by up to five orders of magnitude compared to the naïve algorithm. The implementation of Plausibility-Check is available as part of the RAxML open source code that is available for download at `https://github.com/stamatak/standard-RAxML`

Acknowledgements

I would like to express my sincere gratitude to my advisors Prof. Dr. Alexandros Stamatakis and Dr. Tomáš Flouri for their support, guidance and patience, and for the interesting discussions and helpful corrections which played a key role in the development of this work. I would like to extend my appreciation to Prof. 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.

Contents

1. Introduction

25 years ago, researchers began to think about sequencing the human genome at the time, this seemed nearly impossible [LLB+01]. What happened since then is remarkable. It has become possible to sequence DNA one million times faster and cheaper then before. The molecular revolution [RUN98] has led to an unprecedented accumulation of relevant biological raw data and to the emerging importance of a young interdisciplinary field in computer science and biology: Bioinformatics. A key objective of Bioinformatics is to develop methods and software for storing, retrieving, organizing and analyzing this enormous amount of biological data and thereby enable new insights into the secrets of life. There is little doubt that the future of biology will more and more be shaped by this scientific discipline.

A major area of research in computational biology which benefits the most from the exponential growth of available DNA is evolutionary biology. Disentangling the evolutionary history and diversity of species has preoccupied mankind for centuries. Ever since Darwin's work on evolutionary theory [Dar36], evolutionary trees or phylogenies are typically used to represent evolutionary relationships between species. Biologists estimate that there are about 5 to 100 million species living on earth today. There is evidence from morphological, biochemical, and genetic sequence data, that suggests that all organisms on earth are genetically related. Therefore, the construction of a tree of life compromising all living and extinct organisms on earth is not only a fascinating but also a challenging idea and is often referred to as one of the grand challenges of Bioinformatics in our days.

However, the analysis of phylogenetic trees does not only serve human curiosity but also has practical applications in different fields of science. Phylogenetics help to address biological problems like drug design [BW98] as well as multiple sequence alignment [Edg04], protein structure [SKS94], gene function prediction [JE02], or studying the evolution of infectious diseases [FBB+00].

In the past decade, public databases such as GenBank [BKML+10] have grown exponentially (see Fig 1.1), which, in conjunction with scalable software, allows for computing extremely large phylogenies that contain thousands or even tens of thousands of species (see [GCM+09, SASD11], 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 [Fel78]. For example, there are three distinct unrooted trees for four

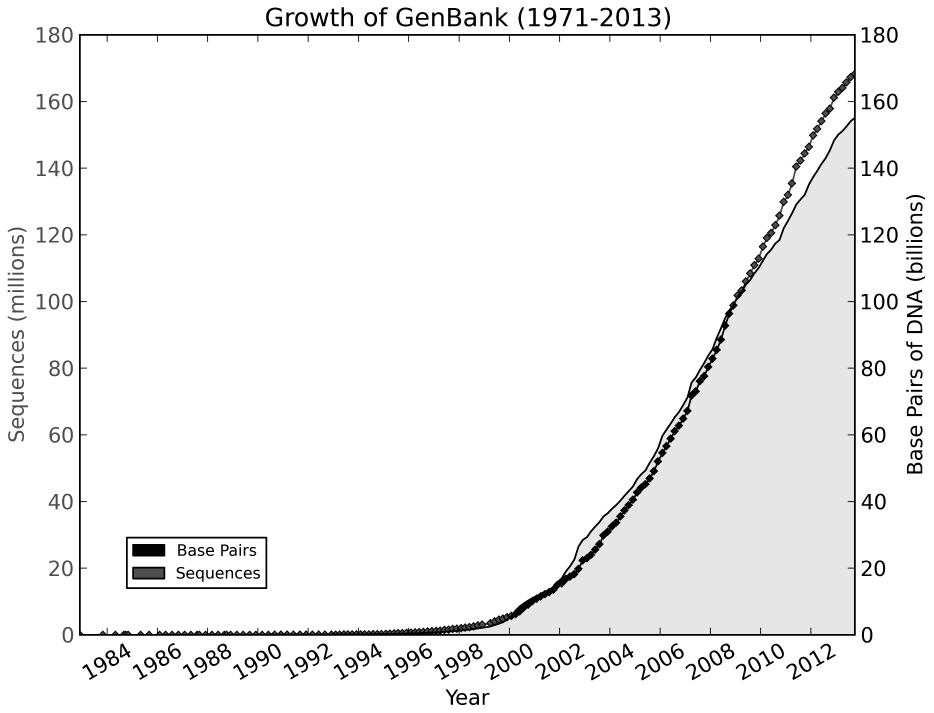


Figure 1.1: Growth of sequence data in GenBank, taken from `http://www.benjaminwicks.com/portfolio/images/01.png`

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 [BC11] and parallel processing techniques [OHL+08] as well as phylogenetic analysis software such as RAxML [Sta06] and MrBayes [RTvdM+12], 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 [MRW02, RC05].

Another problem is the non-uniform sampling of data. Although genomic data is becoming available at an increasing rate, many species are under-sampled because it is difficult to obtain or collect the 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 [RBP12], which leads to potentially biased results, and, therefore, to decreased phylogenetic accuracy [Wie03]. Hence, 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. In this work, we focus on the latter idea.

1.1 Scientific Contribution

In this thesis we introduce a new approach to assess the plausibility of large phylogenies by computing all pairwise topological Robinson-Foulds (RF) distances [RF81] between a large phylogeny (for instance, a phylogeny consisting of 55 000 species [SASD11]) and a set containing a large number of substantially smaller reference trees. These small reference trees are available in curated databases, such as, STBase [MDFB$^+$13] and comprise a strict subset of taxa of the large tree. The underlying assumption is that, the small trees are substantially more accurate. Hence, the lower the average RF distance, or any other reasonable topological distance, between all small reference trees and the large tree is, the more plausible the large tree will be. While we use a reference database containing a billion of small reference trees one could also use reference trees from the literature or from the treebase database [PCD$^+$09].

Our main contribution is the design and production-level implementation in RAxML[1] of an efficient 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.

Parts of this thesis have been derived from the following book chapter: D. Dao, T. Flouri, A. Stamatakis: "Automated Plausibility Analysis of Large Phylogenies", that has already been submitted for review.

A preprint is available at: `http://sco.h-its.org/exelixis/pubs/Exelixis-RRDR-2013-6.pdf`

1.2 Structure of the thesis

The rest of this chapter is organized as follows: Initially, we discuss some preliminaries and present the formal problem description. Subsequently, we 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 [MDFB$^+$13] and studies by Smith *et al.* [SASD11]. Finally, we present a brief summary and conclusion.

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

2. Preliminaries

In this chapter, we cover some basic notions and terminology of a phylogenetic tree. Furthermore we will discuss the required preliminaries and present a formal problem description.

2.1 Phylogenetic Trees: Representation and Terminology

Over the years, many different systems have been suggested for classifying living organisms. Aristotle originally suggested a system that he called the ladder of nature or the great chain of being, where all organisms are organized on a linear scale from lower to higher animals. However, the first really successful system that was devised was suggested by Carl Linnaeus. His system is based on a nested hierarchy of groups and was published in his book Systema Naturae in 1735. Due to its success, people quickly suggested that Linnaeus' system was realistic. However, only about 100 years later Darwin published his Origin of Species in 1859 and provided a scientific explanation. According to Darwin, Linneaus' system of nested hierarchies worked so well, was that, in fact, all life on earth, from the smallest microorganism to the largest vertebrate, is related and can be traced back to a common ancestor. Therefore, taxonomic groups are simply groups of the tree-shaped evolutionary history. Although Darwin's theory encountered heavy opposition within the scientific community in the beginning, it is now broadly accepted. Nowadays, evolutionary relationships among organisms are typically represented by an evolutionary tree (also called phylogeny).

Fig 2.1 outlines such a typical representation of a phylogeny. In evolutionary biology, a phylogenetic tree consists of *nodes* and *branches* connecting the nodes. We refer to terminal nodes as *leaves (or tips)* which correspond to present day living organisms. Nodes can also be *internal* (located inside the tree) and thereby correspond to *hypothetical common ancestors*. The common ancestor of all organism in the tree is called the *root*.

Trees can be *bifurcating* (or fully resolved), meaning that every internal node has two outgoing branches. In evolutionary theory this corresponds to a population of species splitting into two and thereby forming (due to genetic drift or adaptation) new species. Bifurcation is how speciation typically occurs. Trees can also be *multifurcating* (or partially resolved). This is not a reflection of the actual biology but rather the result of not having enough data to resolve on which branch the actual split occured. Phylogenetic trees are represented in two variants and can either be rooted or unrooted.

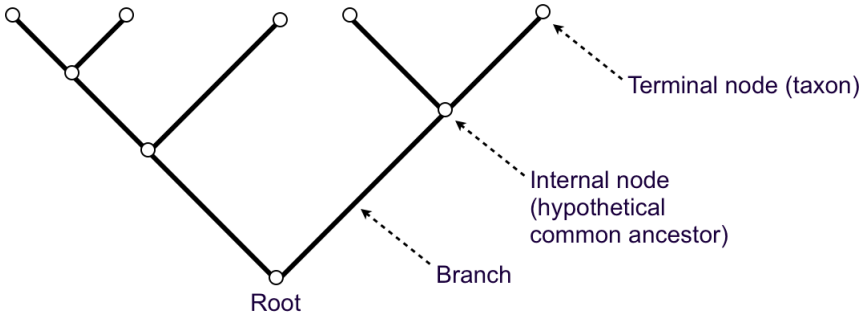


Figure 2.1: Rooted phylogeny

A rooted tree has a single node (the root) that represents an earlier point in time than any other node in the tree. Therefore, a rooted tree has directionality, meaning nodes can be ordered in terms of 'earlier' and 'later'. In unrooted trees there is no directionality. This is not a reflection of reality, but rather due to the fact that we simply do not know where the root is. For further information on phylogenetic trees, especially their history and biological significance, please refer to [Gre08].

The most important information contained in a phylogenetic tree is its branching pattern or so called *tree topology*. Finding a comprehensive tree topology for the tree of life is often considered to be the holy grail of phylogenetics. Unfortunately, the inference of such a phylogeny is NP-hard [CT06] and computationally extremely expensive. It has already been mentioned that the number of possible phylogenies increases super-exponentially with the number of tips. To be precise, the number of possible unrooted trees is $\prod_{i=3}^{n}(2i-5)$ where n denotes the number of tips of the tree. Thus, state of the art phylogenetic analysis software such as RAxML and MrBayes depend on good heuristics to search for the best tree. However, these programs rarely find the best tree due to local optima. Furthermore, numbers like $2.84 * 10^{76}$ possible unrooted trees containing only 50 tips underpin the fact that this search might be even more difficult than finding the famous needle in a haystack.

As a final remark, it should be stated that evolutionary relationships do not have to be represented as trees. There exists alternative forms of representation like for instance phylogenetic networks [Gus05].

2.2 Definitions

Although phylogenies have been used for over 150 years, statistical, computational, and algorithmic work on phylogenies - often referred to as computational phylogenetics - is barely 50 years old. Given the biological background and meaning of different parts of a phylogeny, we can now present a formal description of a tree and its properties.

Definition 2.1 (Degree of a node)**.** *We say a node r has a* degree $\deg(r)$ *if the number of adjacent branches is exactly* $\deg(r)$*.*

Definition 2.2 (Rooted Tree)**.** *We denote a* graph T *as a tree if it is* connected *and* acyclic*. In addition, a tree is called a* rooted tree *if one* internal vertex *or* node r *(i.e.* $\deg(r) > 2$*) has been designated the* root *in which case the* edges *are* directed *towards or away from* r*.*

Definition 2.3 (Height of a node in a tree)**.** *We say a node v of a tree T has a* height $h(v)$ *if the length of the longest path from v to any leaf is exactly* $h(v)$*.*

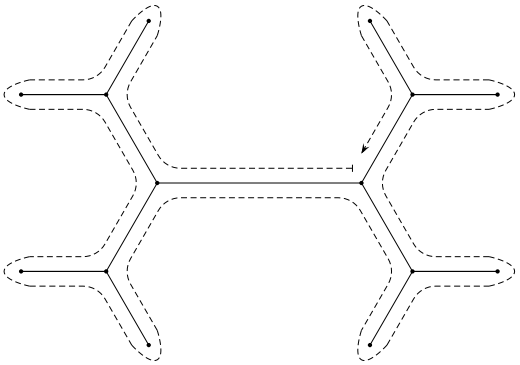


Figure 2.2: Euler traversal of an unrooted tree

Definition 2.4 (Lowest common ancestor)**.** *A 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 $h(w') \leq h(w)$.*

Definition 2.5 (Unrooted tree)**.** *We further say a connected, undirected and acyclic graph T is a* unrooted tree *if T has no root node.*

In the following, we will denote the LCA of two nodes u and v as $lca(u, v)$. We further denote the path from node u to v in a tree T by $u \rightsquigarrow v$. Note that the notion of height and LCA is not defined in unrooted trees. When we use the terms node height or LCA, we therefore imply that we are dealing with rooted trees. Furthermore, we only consider rooted binary trees and unrooted trees that only consists of inner nodes of degree 3 and leaves (degree 1), that is, strictly bifurcating trees. We further know that a rooted strictly binary tree consists of $2n - 1$ nodes where n is the number of leaves in the rooted tree. Trivially, an unrooted bifurcating tree has $2n - 2$ nodes due to the abscence of a root.

Definition 2.6 (Euler tour of a rooted tree)**.** *The Euler tour of a rooted tree is a sequence of nodes generated 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: call* Step *with the left child as parameter, then call* Step *with the right child as parameter and finally list/print v.*

Note that, in an arbitrary graph, an Euler tour is a path that visits every edge exactly once. However, because trees are acyclic graphs, we say that each edge is bidirectional. Hence, we define the Euler tour of a rooted tree as the path through the tree that begins at the root and ends at the root, traversing each edge exactly twice - once to enter a subtree, and once to exit it. The Euler tour of a tree is essentially a depth-first traversal of a tree that returns to the root at the end as outlined in Fig 2.2.

The length of the Euler tour for a rooted binary tree of n leaves is exactly $4n - 3$ because we visit each leaf only once and each inner node (including the root) three times. Hence this sums to exactly $4n - 3$ elements. Analogously to the rooted binary tree, the length of the Euler tour of a unrooted bifurcating tree is $4n - 5$. We can arbitrarily root the tree at an inner node v and traverse the resulting tree starting from v. Note that because deg(v) = 3, we traverse v four times.

Definition 2.7 (Inorder traversal of a rooted tree)**.** *The inorder traversal of an 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.*

Trivially, the length of the inorder traversal sequence for a rooted tree with n leaves is exactly $2n - 1$, because we visit every node once.

Lemma 2.8. *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. First we show that our claim holds for a binary tree with three nodes which is our base case. Next, under the assumption that our claim holds for all trees with up to m nodes, we prove that it also holds for trees with $m + 1$ nodes.

- Let T be a tree with root node u and two child nodes v and w. By definition, the inorder traversal of T yields the sequence v, u, w and hence the base case holds.
- We now assume that our claim holds for every tree with up to m nodes and show that this is also the case for $m + 1$ 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. 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 2.9 (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 $2n - 1$ elements long.

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

Definition 2.10 (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 2.11 (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 2.12 (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.

There are exactly as many non-trivial bipartitions as there are inner branches in a tree. Therefore we can compute $n-3$ bipartitions for unrooted trees and $n-2$ bipartitions for rooted trees.

Definition 2.13 (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 shown to be a metric [RF81] and we can compute it in linear time [PGM07].*

3. Automated Plausibility Analysis

In the following we introduce the algorithm Plausibility-Check.

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 a proper subset of taxa of T. Our underlying assumption is that these smaller phylogenies are more accurate than the large tree. Therefore, a low topological distance would underpin the biological plausibility of the large tree.

Furthermore, we will present two different versions of Plausibility-Check:

- a naïve approach called Naive-Plausibility-Check
- an improved version called Fast-Plausibility-Check

3.1 A naïve approach

In this section, we will present a formal description of Naive-Plausibility-Check. In addition, we will discuss its time complexity and provide an example.

3.1.1 Overview

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. An overview of the naïve approach which we denote as Naive-Plausibility-Check is outlined in Alg 3.1

We take as input parameters a large phylogeny T and a set F of small reference trees. As a important prerequisite, we ensure that trees in F only contain proper subsets of the taxa in T. Next, Naive-Plausibility-Check extracts all bipartitions of T, which we further denote as $B(T)$ and stores them in a hash table. Then, we loop over all given small trees t_i in F and extract for every small tree t_i its corresponding leaf-set L'_i. Given L'_i, Naive-Plausibility-Check can now compute the induced subtree $T|t_i$, its bipartitions $B(T|t_i)$, and hence the Robinson-Foulds distance between $T|t_i$ and t_i. When all small trees have been processed, the algorithm finishes by returning a list of m pairwise RF distances. Averaging these distances will give us a metric which tells us how plausible our large tree is in respect to the small reference trees.

Algorithm 3.1: NAIVE-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))
Let RF(T, F) = r_1, r_2, ..., r_m
```

Algorithm 3.2: EXTRACT-NON-TRIVIAL-BIPARTITIONS

Input : Tree T of n nodes

Output : List of all non-trivial bipartitions $B(T)$ of T

```
▷ Root the tree T
▷ Traverse the tree bottom-up towards the root with a depth-first traversal
▷ Recursivley extract all bipartitions from each inner branch of T
Let B(T) = b_1, b_2, ...b_{n-3} be the list of non-trivial bipartitions of T
```

3.1.2 Extracting Bipartitions

By definition, the RF distance compares bipartitions as a method to calculate the topological distance between different trees with the same taxon set. While we will only consider the RF distance, one could also use a different and more advanced topological distance, such as the quartet-distance [BTKL00] for instance.

EXTRACT-NON-TRIVIAL-BIPARTITIONS calculates all nontrivial bipartitions of T.

In the naïve approach, we will first extract all non-trivial bipartition b_i at each inner branch of the large tree and store them in a hash table. We will then use this hash table to generate the induced non-trivial bipartitions b'_i in a later step.

The algorithm relies on a rooted view of the otherwise unrooted large tree. Therefore, we initially root the tree at an arbitrary branch and recursively compute all bipartitions bottom-up towards the virtual root via a depth-first traversal.

The implementation requires $\mathcal{O}(n^2)$ time for traversing the tree and storing all bipartitions in a suitable data structure, typically, a hash table. For further implementation details see Chapter 4.1 in [PABE+09].

However, there are also alternative ways for extracting bipartitions. In section 3.3.5 we will introduce a novel approach to compute bipartitions given only the preorder array representation of the tree.

3.1.3 Computing Induced Bipartitions

In the previous step, we efficiently extracted all non-trivial bipartitions from the large tree. Given $B(T)$, we can now filter our required induced subtree bipartitions. In other words, instead of constructing an instance of the induced subtree with respect to the taxon set of the small reference tree, we can skip this step and directly compute the induced bipartitions $B(T|t_i)$.

Algorithm 3.3: 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 $T|t_i$

```
1 ▷ Iterate through all bipartitions B(T)
2 for i ← 1 to |B(T)| do
3     b' ← ▷ Filter bipartition b with L'
4     if b' is non-trivial then
5         ▷ Rehash b'
```

As a prerequisite, we first need to extract the leaf-set L' of the reference tree.

Next, COMPUTE-INDUCED-SUBTREE-NBP extracts the induced bipartitions $B(T|t_i)$ by iterating through all bipartitions b_i of T which we previously stored in a hash table. It generates the induced non-trivial bipartitions b'_i for $T|t_i$ by deleting the leaves which are not present in the leaf-set L'. The resulting induced bipartitions are then re-hashed. Hence, 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).

Example 3.1.1. Given the query tree T in Fig. 3.1 and the smaller reference tree t_i in Fig. 3.2, we initially compute the induced non trivial bipartitions $B(T|t_i)$ and then the RF distance. First we extract all non-trivial bipartitions of T, which we represent as bitvectors, and hash them into a hashtable h. We represent all bipartitions in a canonical way as bitvectors:

$$B(T) = 110000, 110010, 110011$$

By extracting the leaf-set L' of t_i, we know that taxon E is missing in the reference tree. Therefore we have to iterate through all bitvectors in h and set the bits to zero, which is not represented in L'. In our example, the particular bit can be found at index 5. After filtering all bitvectors, the resulting sequence is

$$B(T|t_i) = 110000, 110000, 110001 = 110000, 110001$$

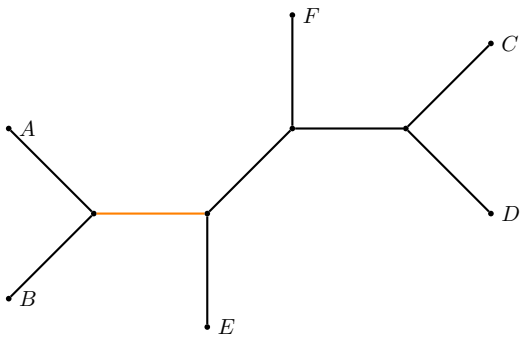
We need to pay attention to potential duplicates. Next, we will rehash all bipartitions in $B(T|t_i)$ and look how many bipartitions the reference tree t_i shares with $T|t_i$.

$$B(t_i) = 110000, 110100$$

In this example, we can find one shared bipartition (110000). Therefore the RF distance between $T|t_i$ and t_i is 1.

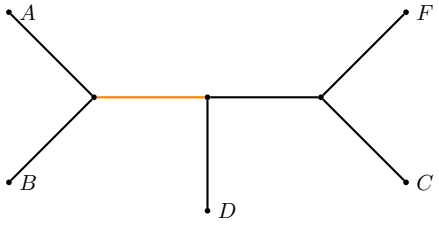
3.1.4 Discussion

In summary, NAIVE-PLAUSIBILITY-CHECK has an overall complexity of $\mathcal{O}(n^2mk)$, 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. However, as mentioned earlier, we can reduce the complexity using bitvectors. Thus, the reduced time complexity is $\mathcal{O}(\frac{n^2mk}{w})$. Nonetheless, this results in serious performance issues for the standard use case of our algorithm as outlined in Section 5.2. Automated Plausibility Analysis is designed to deal with large phylogenies and millions of substantially smaller reference trees. Thus, for each small reference tree,



$B(T)$	Bitvector
AB|CDEF	110000
ABE|CDF	110010
ABEF|CD	110011

Figure 3.1: Unrooted phylogeny T of 6 taxa. We represent its non-trivial bipartitions as bitvectors



$B(t_i)$	Bitvector
AB|CDF	110000
ABD|CF	110100

Figure 3.2: Small reference tree t_i and its non-trivial bipartitions $B(t_i)$ for example 3.1.1. t_i shares with T from figure 3.1 one non trivial bipartition (orange)

Naive-Plausibility-Check needs to iterate through the whole taxon set of the large tree to extract the required induced bipartitions. Therefore, we will discuss how to improve the algorithm in the next section and introduce a significantly faster algorithm for extracing induced subtrees.

3.2 Towards a faster method

In this section we present a novel method for speeding up the computation of induced subtrees for a given leaf-set. We will further discuss the theory behind this method and outline the benefits with respect to the previous naïve approach.

The key idea is to root the large tree at an inner node and build the induced subtree by computing 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 it is possible to preprocess the large tree in a way that we can compute the required LCAs of each pair in $\mathcal{O}(1)$ time. We will get back to this in Section 3.3. In addition, it is sufficient to compute the LCAs of $|L'| - 1$ particular pairs of leaves which is a rather surprising finding. The rest of this Section focuses on proving this statement.

For an arbitrary leaf-set L' of three and more taxa, 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 $V_i = \{\, v \mid v \in V,\ deg(v) = i \,\}$.

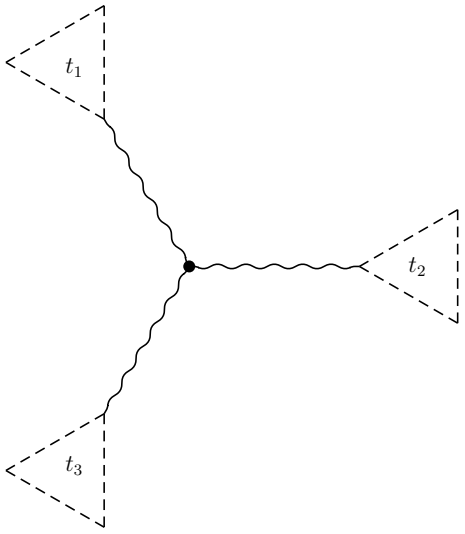


Figure 3.3: Node from V_3

Note that, for a leaf-set with size two, the induced subgraph is a path of nodes and edges from one taxon to another and therefore the induced tree is a simple line with the two elements of the leaf-set as endpoints. Given an unrooted binary tree, we know from section 2 that, the size of V_3 is exactly $|L'| - 2$.

Next, we show that all nodes in V_3 are LCAs of some pairs of leafs in the leaf-set L'. In fact, V_3 consists of all LCAs of all possible pairs of leaves in L' except possibly at most one LCA (which will be in V_2). We deal with these special cases in Lemma 3.2.

Lemma 3.1. *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. We will prove the lemma by contradiction.

- Let us assume that there exists a node v in V_3 that is not the lowest common ancestor of any two nodes from L'. Because $deg(v) = 3$, in the rooted tree there exist exactly two paths $v \rightsquigarrow u$ and $v \rightsquigarrow w$, where $u, w \in L'$. Now let $p = lca(u, w)$ 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 $r \rightsquigarrow p$ are 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. 3.3 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 3.2. *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.*

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

- An internal node v (as depicted in Fig. 3.3) 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,

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

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.

Lemma 3.3. *In the following, we denote the preorder id of a nodev as pid(v). Let $u, v, w \in L'$ such that $u < v < w$. It holds that $\mathit{lca}(u, w) = v'$, such that $\mathit{pid}(v') = \min(\mathit{pid}(p), \mathit{pid}(q))$ where $p = \mathit{lca}(u, v)$ and $q = \mathit{lca}(v, w)$.*

Proof. Proof by contradiction. Let r be the root of the tree. Let us assume that $p = \mathit{lca}(u, v)$ and $q = \mathit{lca}(v, w)$. 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 into 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 $v' = \mathit{lca}(u, w)$ is not p. It can then appear along the path $r \rightsquigarrow u$. If it appears anywhere except p we have a cycle. Therefore, $\mathit{lca}(u, w) = p$ and it holds that $\mathit{pid}(p) = \min(\mathit{pid}(p), \mathit{pid}(q))$.
- In case q appears along $r \rightsquigarrow p$. Let us assume that $v' = \mathit{lca}(u, w)$ is not q. It can appear along the path $r \rightsquigarrow u$. If it appears anywhere except p we have a cycle. Therefore, $\mathit{lca}(u, w) = q$ and it holds that $\mathit{pid}(q) = \min(\mathit{pid}(p), \mathit{pid}(q))$.

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 3.4. *Given the sequence of 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 that $pid(u) = \min(pid(u_j), pid(u_{j+1}), \ldots, pid(u_{k-1}))$ 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 3.3 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 3.3 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 3.4 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. 3.3), 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, 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.

3.3 An improved algorithm

Based on the theoretical insights from the previous Section, we are now in a position to introduce and implement an effective algorithm for inducing a subtree $T|t_i$ with respect to a given large tree T and the leaf-set L' of the smaller reference tree t_i. The algorithm is an important part of the improved version of Plausibility-Check which we denote as Fast-Plausibility-Check.

Here we discuss the required preprocessing of the large tree T. We create a dedicated data structure from T so that we can query for LCAs in $\mathcal{O}(1)$ time. Moreover, we explain the actual inducing step and how to query for LCAs to build the induced tree $T|t_i$. Finally, we present two ways for computing the resulting bipartitions from $T|t_i$.

Algorithm 3.4: FAST-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

```
1  RMQ(T) ← PREPROCESS-ROOTED-TREE(T)
2  ▷ Generate mapping f : L → ⟨1, 2(n − 1)⟩
3  for i ← 1 to m do
4      ▷ Extract leaf-set L'_i from t_i
5      Let I(T|t_i) be the inorder notation of T|t_i
6      I(T|t_i) ← COMPUTE-INDUCED-TREE(RMQ(T), L'_i, f)
7      ▷ Sort I(T|t_i) to get P(T|t_i)
8      B(T|t_i) ← EXTRACT-BIPARTITIONS-PREFIX(P(T|t_i))
9      B(t_i) ← EXTRACT-NON-TRIVIAL-BIPARTITIONS(t_i)
10     r_i ← RF-DISTANCE(B(T|t_i), B(t_i))
11 Let RF(T, F) = r_1, r_2, ..., r_m
```

3.3.1 Overview

An overview of the fast approach of PLAUSIBILITYCHECK is outlined in Alg 3.4

Just as the naïve approach, FAST-PLAUSIBILITY-CHECK takes as input parameters a large phylogeny T and a set F of small reference trees. For better understanding, we further divide the algorithm into three steps - preprocessing, inducing and bipartition extraction. In the first step, we preprocess T and create a succint Range Minimum Queries datastructure, which we denote as $RMQ(T)$, to query for LCAs in constant time. Furthermore, we generate a mapping f which allows us to quickly look up particular pairs of leaves of L in $RMQ(T)$. Next, we iterate through all small trees t_i in F and extract its leaf-sets L'_i. Given our preprocessed datastructure $RMQ(T)$, a mapping f and the extracted leaf-set L'_i, we can now compute the actual induced subtree $T|t_i$. Finally, given the resulting preorder representation of $T|t_i$, we compute the induced bipartitions $B(T|t_i)$ and hence the Robinson-Foulds distance between $T|t_i$ and t_i.

3.3.2 Preprocessing

To allow fast LCA queries, we first have to root T at an arbitrary inner branch and then create a data structure that allows for queries in constant time. To achieve this we consider the close relation between LCA computation and *Range Minimum Queries* (RMQ) that was first published in [BV93]. For a static array $A[1, n]$ of n elements, a Range Minimum Query $\text{RMQ}_A(i, j)$ for $i < j$ returns the position of the minimum element in the subarray $A[i, j]$. This problem was shown to be linearly equivalent to the LCA-problem by Gabow *et al.* [GBT84], in the sense that both problems can be transformed into each other in time linear in the size of the input. As outlined in [FH07], the size of the succint preprocessed data structure for a tree with n nodes is at most $2n + o(n)$ bits. In addition, queries for the minimum element in between two array indices can be done in constant time. To benefit from these insights, we first have to reduce our LCA queries to RMQ queries. Therefore, we build a sequence of node identifiers which correspond to an Euler tour of the rooted tree. These identifiers are assigned to each node during a preorder traversal of T. Algorithm 3.5 lists the required preprocessing steps. For further implementation details on how to construct a specific RMQ structure we rather point the interested reader to the available literature [FH06, FH07].

Algorithm 3.5: PREPROCESS-ROOTED-TREE

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

1 ▷ Root tree T
2 ▷ Build Euler tour of T
3 Let $E(T) = s_1, s_2, \ldots, s_{2n-1}$ be the Euler tour of T
4 ▷ Prepare a RMQ data structure
5 Let $P(T) = pid(s_1), pid(s_2), \ldots, pid(s_{2n-1})$ be the list of preorder identifiers of $E(T)$
6 Let $RMQ(T) =$ RANGE-MINIMUM-QUERY-PREPROCESS$(P(T))$

3.3.3 Computing Lowest Common Ancestors

Before we can finally compute the LCA of two leaf nodes of L' in constant time, we require one additional data structure. Although we built the RMQ datastructure in the previous step, we still lack the information of the particular input indices for each leaf. Let L be the leaf-set of T and n the number of taxa in the large tree. Our additional data structure represents the mapping

$$f : L \to \langle 1, 4n - 5 \rangle$$

By mapping each leaf of the rooted tree T to the position where it appears for the first time in the Euler tour, we can easily look up the required indices. Given the mapping f , we are now in a position to compute the LCA of two leaves $u, v \in L$ in time $\mathcal{O}(1)$ using RMQs. Therefore we query for the node w with the lowest preorder identifier in the range $\langle i, j \rangle$, where $i = \min(f(u), f(v))$ and $j = \max(f(u), f(v))$. Since we reduced LCA to RMQ, w is not only the minimum element in the range $\langle i, j \rangle$ but also the lowest common ancestor of u and v.

3.3.4 Constructing Induced tree

We can now compute the inorder sequence of the induced subtree. COMPUTE-INDUCED-TREE takes as input parameters the extracted leaf-set L', the RMQ datastructure $RMQ(T)$ and the mapping f. First, we sort L' according to the preorder traversal identifiers of T and denote our sorted leaf-set as $L'_S = u_1, u_2, \ldots, u_{|L'|}$. Next, we compute the LCA of every pair (u_i, u_{i+1}) and hence 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 $2|L'| - 1$. As stated earlier in Lemma 2.8, the resulting sequence corresponds to the inorder notation of the induced rooted tree.

Algorithm 3.6: COMPUTE-INDUCED-TREE

Input : Preprocessed RMQ structure $RMQ(T)$
Leaf-set L'
Mapping $f : L' \to \langle 1, \ldots, n \rangle$
Output : Inorder notation of induced tree T'

1 ▷ Sort leaf-set according to preorder traversal identifiers of T
2 Let $L'_S = (u_1, u_2, \ldots u_{|L'|})$ such that $u_{i-1} < u_i < u_{i+1}$, for $1 < i < |L'|$
3 ▷ Compute common ancestors
4 **for** $i \leftarrow 2$ **to** $|L'|$ **do**
5 $\quad c_i \leftarrow lca(u_{i-1}, u_i)$

Algorithm 3.7: 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
```

3.3.5 Computing Induced Bipartitions

By sorting the inorder sequence in ascending order we obtain the preorder notation of the induced rooted tree. A direct approach to compute the induced bipartitions would be to build the induced tree $T|t_i$ in $\mathcal{O}(|L'|)$ directly from the preorder sequence using algorithm 3.7. BUILD-INDUCED-TREE constructs the induced rooted tree in depth-first-order using a stack. Note that, according to Lemma 3.2, we might have the case that the resulting root node is of degree two. In this case, we remove the root and connect its two children by an edge. Given $T|t_i$, we can further compute $B(T|t_i)$ with EXTRACT-NON-TRIVIAL-BIPARTITIONS which we already introduced in Section 3.1.

However building an induced tree requires additional storage, therefore we present EXTRACT-BIPARTITIONS-PREFIX(Alg 3.8) which is an alternative method for extracting non-trivial bipartitions without relying on an instance of a tree structure. The algorithm determines all subtrees of $T|t_i$ and computes the corresponding bipartitions by seperating these subtrees from $T|t_i$.

Example 3.3.1. Compute the induced tree $T|t_i$ and its bipartitions, given the query tree T in Fig. 3.4 and a leaf-set L' that consists of the leaves in Fig. 3.2.

First, we transform the unrooted tree into a rooted one by designating one branch as root. We then assign a preorder traversal identifier to each node as shown in Fig. 3.5, 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

0 1 0 2 3 2 4 5 4 6 4 2 0 7 8 9 8 10 8 7 11 12 11 13 11 7 0

Algorithm 3.8: EXTRACT-BIPARTITIONS-PREFIX

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

```
1  k = 0
2  for i ← n to 1 do
3      if k < number of splits then
4          V[i] ← 1
5          if u_i is not a leaf then
6              for j ← 1 to deg(u_i) do
7                  V[i] ← V[i] + V[i + V[i]]
8              for j ← 1 to V[i] do
9                  if u_{i+j} is a leaf then
10                     Add u_{i+j} into b_i
11                 else
12                     Add all nodes from b_{i+j} into b_i
13                     j ← j + V[i + j]
14             k = k + 1
```

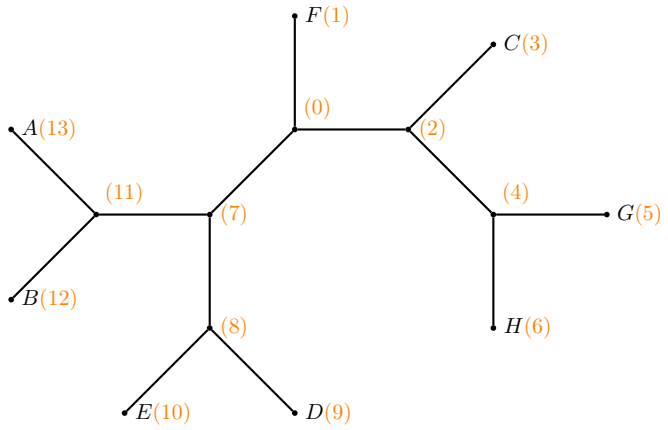


Figure 3.4: unrooted phylogeny T of 8 taxa with its preorder identifiers (orange)

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

$$1, 3, 9, 12, 13$$

Then, we compute the LCAs of node pairs

$$(1, 3), (3, 9), (9, 12), (12, 13)$$

and obtain the sequence

$$1, 0, 3, 0, 9, 7, 12, 11, 13$$

which represents the inorder notation of the induced tree. Next, we extract the induced non-trivial bipartitions using Algorithm 3.8. To this end, we sort the inorder sequence to retrieve the preorder notation $P(T|t_i)$ of $T|t_i$.

$$0, 0, 1, 3, 7, 9, 11, 12, 13$$

We are now able to compute the size of each subtree in $T|t_i$ with a bottom-up approach and store it in an array V. Take for example $V[4]$. After initialization and the first

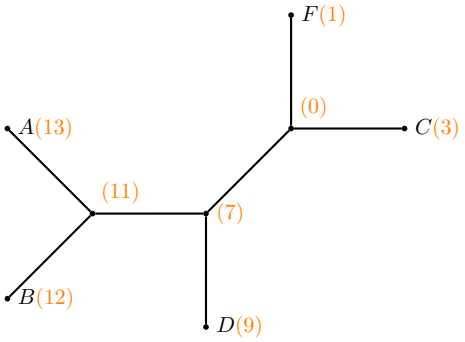


Figure 3.5: Induced phylogeny $T|t_i$ for Example 3.3.1

i	0	1	2	3	4	5	6	7	8
$P(T\|t)$	0	0	1	3	7	9	11	12	13
$V[i]$	-	8	1	1	5	1	3	1	1

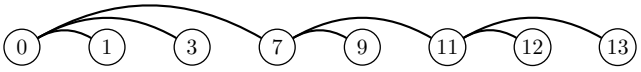


Figure 3.6: Induced subtree after using Extract-Bipartitions-Prefix

iteration $V[4]$ is $V[4] + V[4 + V[4]] = 1 + V[5] = 2$. In the second step, we get $V[4] = V[4] + V[4 + V[4]] = 2 + V[6] = 5$ and thus the number of nodes of the subtree which has 7 as root (see Fig.3.6). We can now easily continue to compute all non-trivial bipartitions via dynamic programming.

Note that as an alternative, we could also build the induced tree directly from the inorder notation, or sort the sequence and build the tree using Algorithm 3.7 and hence compute the bipartition with our familiar depth first approach (section 3.1). Fig. 3.5 depicts the induced tree.

3.3.6 Two variants

As stated earlier, it is possible to implement the algorithm in two different ways, depending on the amount of available memory. The difference between 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 3.6, and then apply Algorithm 3.7 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 preprocessing 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.

3.3.7 Discussion

We showed that Fast-Plausibility-Check appears in two variants and can have either a time complexity of $\mathcal{O}(km^2 \log m)$ or $\mathcal{O}(km^2)$. Furthermore, in the first variant, the

Algorithm	Time complexity	Space complexity
Naive-Plausibility-Check	$\mathcal{O}(n^2mk)$	$\mathcal{O}(n^2)$
Extract-Non-Trivial-Bipartitions	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Compute-Induced-Subtree-NBP	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Fast-Plausibility-Check	$\mathcal{O}(km^2 \log m)/\mathcal{O}(km^2)$	$\mathcal{O}(n+m^2)/\mathcal{O}(km+m^2)$
Preprocess-Rooted-Tree	$O(n)/\mathcal{O}(\max(n, km))$	$O(n)/\mathcal{O}(\max(n, km))$
Compute-Induced-Tree	$\mathcal{O}(m \log m)/\mathcal{O}(m)$	$\mathcal{O}(m)$
Extract-Bipartitions-Prefix	$\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$

Figure 3.7: Overview of the time and space complexities of our presented algorithms, where n is the number of taxa of the large tree, k the number of reference trees and m the average size of the reference trees

algorithm needs $\mathcal{O}(n+m^2)$ space while in the second variant it requires $\mathcal{O}(km+m^2)$ space. Although the second variant has a slightly better time complexity than the first one, we have to consider the fact, that the number of small reference trees k might be far larger than n. Therefore, the choice of algorithm strongly depends on the available amount of memory and the input data.

Nevertheless, both implementations of Fast-Plausibility-Check have one significant advantage compared to Naive-Plausibility-Check. Besides the better overall space complexity, the time complexity of the improved algorithm is independent of the size of the large tree. Thus, for our standard use case $n \ll m$, where n is the number of nodes in the large tree and m denotes the average size of the small reference trees, Fast-Plausibility-Check can achieve much better performance results than the naïve approach.

Figure 3.7 summarizes all presented algorithms and their asymptotic time and space complexities.

Note that, analogous to Section 3.1, we can also reduce the time complexity of Fast-Plausibility-Check by a factor of w (vector width of the target architecture) using a bit-parallel implementation.

4. Implementation details

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 3.3 for extracting induced subtrees from bifurcating unrooted trees.

4.1 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 branch 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 $4n - 5$ entries because the Euler traversal visits $n - 2$ inner nodes (all inner nodes except for the root) three times, all n leaves once and the root 4 times. To further optimize the induced tree reconstruction phase, we use an additional array, which we denote as 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 requirements or list data structures, for instance.

Based on the Euler tour, we can now construct a RMQ data structure. for this, we use the source code developed by Fischer *et al.* [FH07] which we modify and adapt to our purposes.

4.2 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.

As before, we denote the number of taxa in the small reference tree by m. 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 3.3. 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 3.7.

4.3 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.* [PABE+09].

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 3.3, 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 3.8. 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.

5. Evaluation

In the following we describe the experimental setup and provide a comparative performance analysis between Fast-Plausibility-Check and Naive-Plausibility-Check on simulated as well as real phylogenetic data.

5.1 Test Datasets

For conducting experiments we extracted real-world data and generated synthetic data. Both datasets are available for download at `http://www.exelixis-lab.org/material/plausibilityChecker.tar.bz2`.

5.1.1 Real-world Datasets

For real-world data tests, we used the mega-phylogeny of 55 473 plant species by Smith *et al.* [SASD11]. To obtain a reference tree set, we queried all trees in STBase [MDFB$^+$13] which are proper subsets of the large tree. Our reference tree set consists of 175 830 trees containing 4 up to 2 065 taxa.

5.1.2 Simulated Datasets

As large trees, we used 15 trees with 150 up to 2 554 taxa from [PSM10] 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 3.7.

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.

5.2 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.

5.2.1 Mega-phylogeny

For the mega-phylogeny, we obtain an average relative RF distance of 0.318 (see Table 5.1) 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 thesis.

Fig. 5.2 illustrates the overall distribution of RF distances, whereas Fig. 5.1 shows the corresponding distribution for the 20 000 largest reference trees. Using our improved algorithm, we can process the 175 830 small reference trees five orders of magnitude faster than with the naïve algorithm. In total, the naïve algorithm required 67 644 s 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.

5.2.2 Simulated data

The naïve algorithm needs more time for larger phylogenies as discussed in Section 3.1 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. 5.5, 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 5.2). In the following, we calculated the speedup by comparing the run times for the inducing step in both algorithms. Fig. 5.3 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), whereas the efficient method only traverses the small reference trees. 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. 5.4 shows the overall execution times for both algorithms,

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 5.1: Test results for a mega-phylogeny of 55 473 taxa

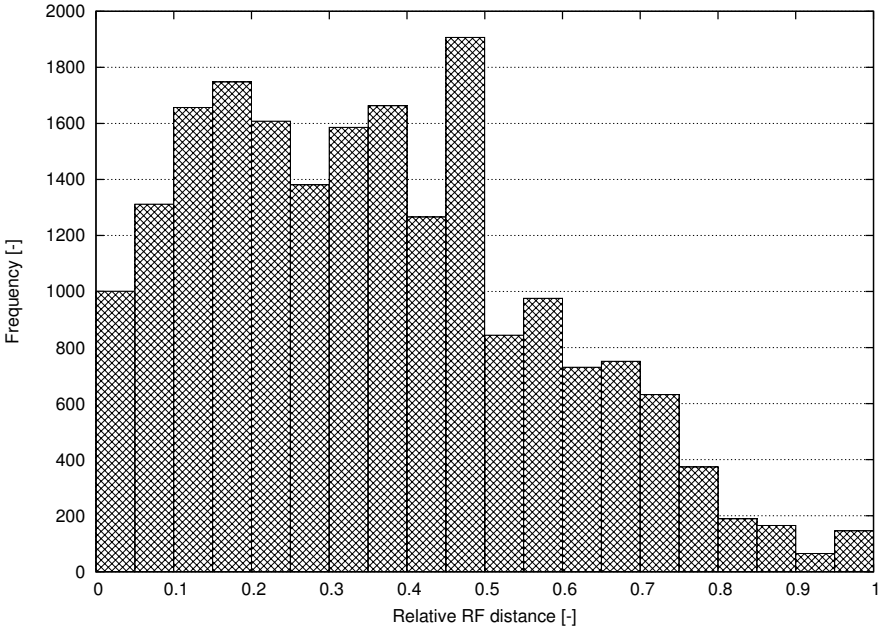


Figure 5.1: Distribution of relative RF distances for the 20 000 largest reference trees (30 up to 2 065 taxa)

while Fig. 3.5 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 5.3 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.

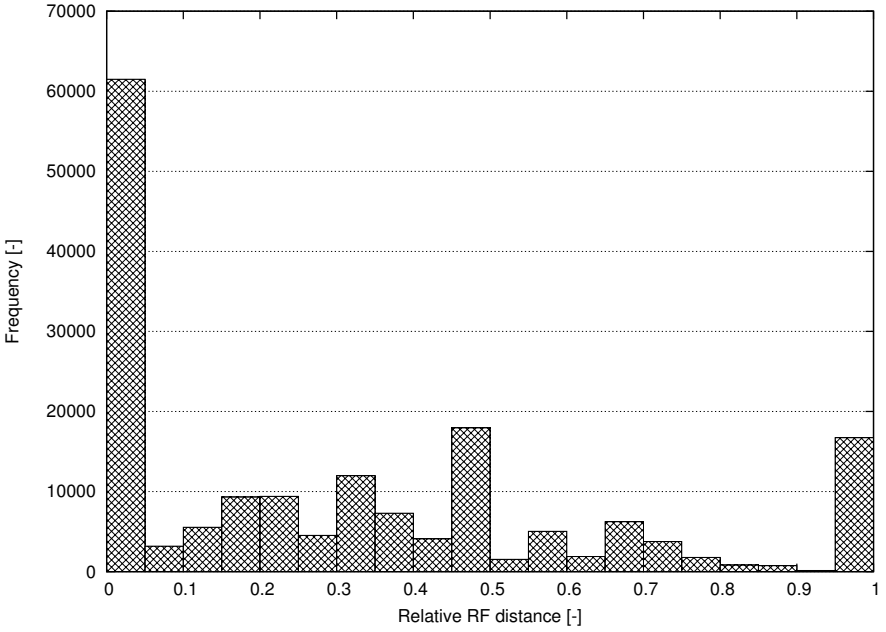


Figure 5.2: Distribution of all relative RF distances between the large mega-phylogeny and the reference trees from STBase.

# 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 5.2: 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 5.3: Test results for one million simulated reference trees (each containing 128 taxa)

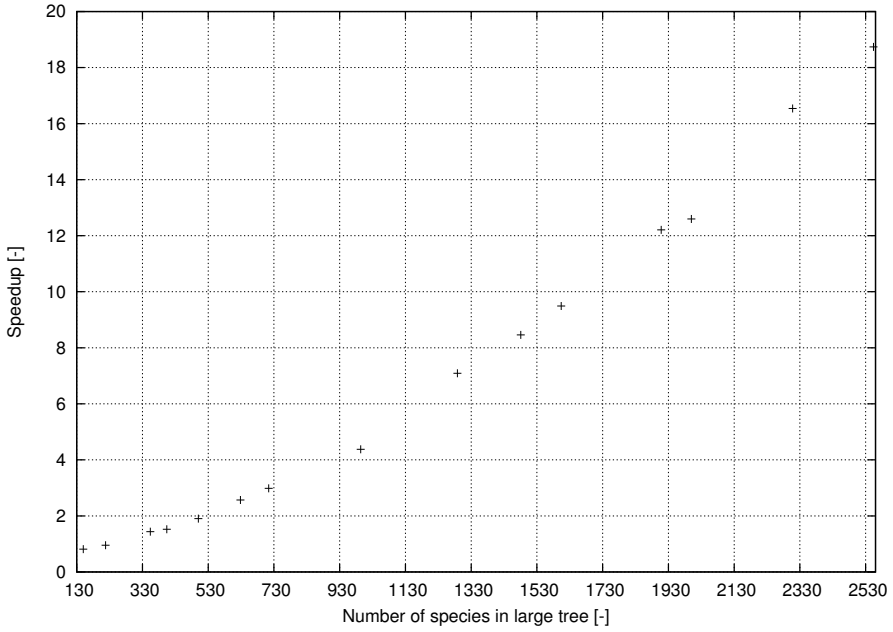


Figure 5.3: Speedup of the effective inducing tree approach. We calculate the speedup by dividing the overall naïve inducing time with the effective inducing time.

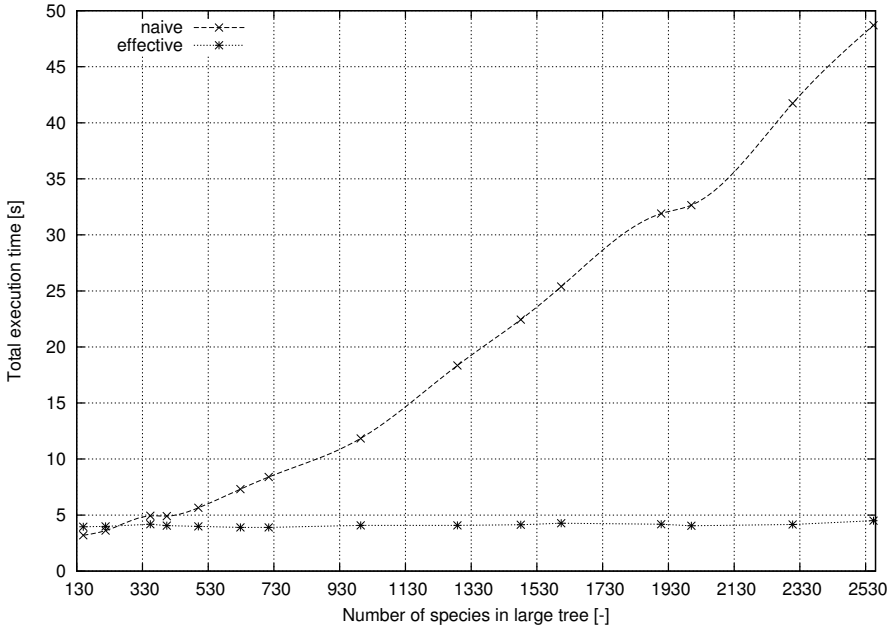


Figure 5.4: Total execution time of the naïve algorithm (dashed) compared to the effective approach (dotted)

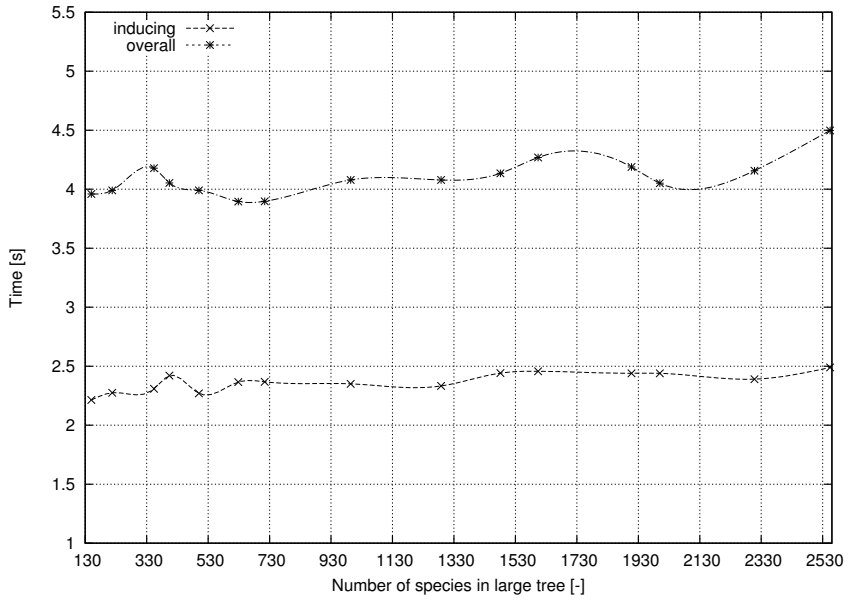


Figure 5.5: Running time of the effective inducing step (dashed) compared to the overall execution time of the effective algorithm (dotted)

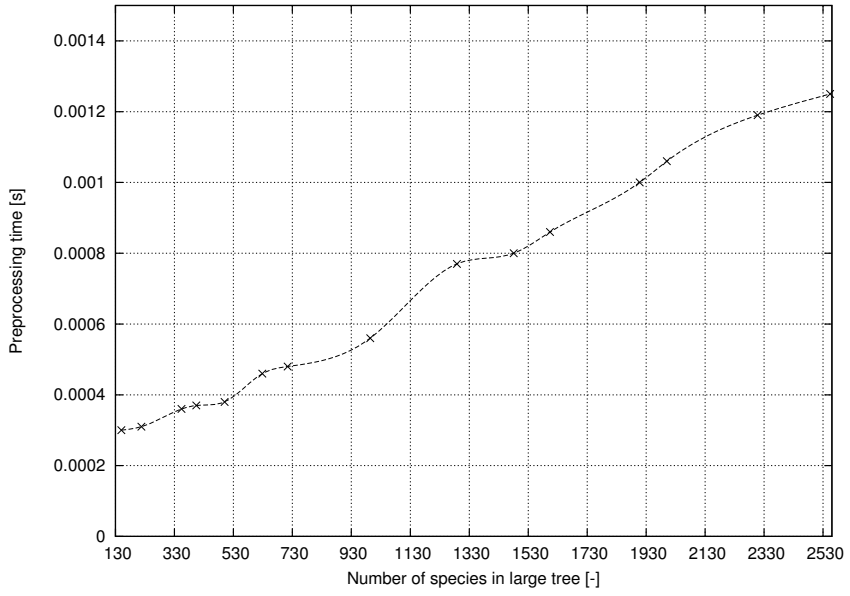


Figure 5.6: Time needed for the preprocessing step of the effective algorithm

6. Conclusion

In view of the increasing popularity of megaphylogeny approaches in biological studies [GCM+09, SASD11], 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 to the small trees. Here, we use the RF distance metric, but any other, potentially more advanced, topological distance metric, such as the quartet-distance [BTKL00] for instance, can be used.

Future Work in Phylogenetics. 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 [AKS13]. 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.

Future Work in Other Fields. Besides its application in phylogenetics, one could also consider adapting Plausibility-Check for other purposes such as information retrieval.

For instance, given a HTML or XML document, which can be represented as a tree structure, it is often useful to know whether or not a particular subtree structure is part of the document. Using an adapted version of our algorithm, one can think of comparing the document structure with thousands of other documents in an efficient way or ranking pages according to their similiarity. Furthermore, linguists may use the algorithm to parse trees and thus compute how similar a specific language is to a set of other languages given their parse trees.

Bibliography

[AKS13] 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.

[BC11] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011.

[BKML+10] 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.

[BTKL00] 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.

[BV93] Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, April 1993.

[BW98] James R Brown and Patrick V Warren. Antibiotic discovery: is it all in the genes? *Drug Discovery Today*, 3(12):564–566, 1998.

[CT06] Benny Chor and Tamir Tuller. Finding a maximum likelihood tree is hard. *J. ACM*, 53(5):722–744, September 2006.

[Dar36] Charles Darwin. *The origin of species*. Everyman's library. Dent, 1936.

[Edg04] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[FBB+00] 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.

[Fel78] Joseph Felsenstein. The number of evolutionary trees. *Systematic Biology*, 27(1):27–33, 1978.

[FH06] 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.

[FH07] 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.

[GBT84] Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 135–143, New York, NY, USA, 1984. ACM.

[GCM+09] 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.

[Gre08] T.Ryan Gregory. Understanding evolutionary trees. *Evolution: Education and Outreach*, 1(2):121–137, 2008.

[Gus05] Dan Gusfield. Optimal, efficient reconstruction of root-unknown phylogenetic networks with constrained and structured recombination. *Journal of Computer and System Sciences*, 70(3):381 – 398, 2005. Special Issue on Bioinformatics II.

[JE02] M. J.A. Eisen, Wu. Phylogenetic analysis and gene functional predictions: phylogenomics in action. *Theoretical population biology*, 61(4):481–487, 2002.

[LLB+01] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[MDFB+13] 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.

[MRW02] BernardM.E. Moret, Usman Roshan, and Tandy Warnow. Sequence-length requirements for phylogenetic methods. In Roderic Guigó and Dan Gusfield, editors, *Algorithms in Bioinformatics*, volume 2452 of *Lecture Notes in Computer Science*, pages 343–356. Springer Berlin Heidelberg, 2002.

[OHL+08] 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.

[PABE+09] 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.

[PCD+09] 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.

[PGM07] 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.

[PSM10] 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.

[RBP12] Béatrice Roure, Denis Baurain, and Hervé Philippe. Impact of missing data on phylogenies inferred from empirical phylogenomic datasets. *Molecular Biology and Evolution*, 2012.

[RC05] 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.

[RF81] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[RTvdM+12] Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L. Ayres, Aaron Darling, Sebastian Höhna, 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.

[RUN98] Mostafa Ronaghi, Mathias Uhlén, and Pål Nyrén. A sequencing method based on real-time pyrophosphate. *Science*, 281(5375):363–365, 1998.

[SASD11] Alexandros Stamatakis Stephen A. Smith, Jeremy M. Beaulieu and Michael J. Donoghue. Understanding angiosperm diversification using small and large phylogenetic trees. *American Journal of Botany*, 98(3):404–414, 2011.

[SKS94] 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.

[Sta06] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[Wie03] John J. Wiens. Missing data, incomplete taxa, and phylogenetic accuracy. *Systematic Biology*, 52(4):528–538, 2003.