Universiteit Leiden
The Netherlands

BSc Bioinformatics

Localized Information Comparison and Analysis for MycoDiversity Database

Lena ten Haaf (s2566818)

Supervisors:
Prof. Dr. Ir. Fons Verbeek
Dr. Rutger Vos (Naturalis)

BACHELOR THESIS
Computer Science, Specialization Bioinformatics

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/07/2023

Abstract

The MycoDiversity Database (MDDB) is developed and designed by LIACS in collaboration with Naturalis. This resource represents a lot of fungal species that have been sampled from the soil. In order to compare species in the database with "new" samples, a phylogenetic reference tree has been computed. This reference tree serves as a valuable resource for assessing the fungal biodiversity in a particular area. Phylogenetic placement is a technique that enables this analysis, and various algorithms are available to facilitate such placement. We propose a two-step approach that uses `pplacer` to perform the placement: i) Determine the correct subtree in the reference tree using BLAST, ii) Use `pplacer` to determine the position of the query within the selected subtree. The method is validated by comparing the difference in the number of nodes between two leaves in the original tree and the tree with the query placed by `pplacer`. This method will enable researchers to gain insight into the biodiversity of a sample. Phylogenetic biodiversity takes into account the evolutionary relationship and genetic variation among species. This makes the estimation of biodiversity more robust compared to an estimation based on a taxonomic profile. Furthermore, it can be used to compare different samples or perform longitudinal research which is very important to assess the biodiversity of one particular area over time.

Contents

1 Introduction

This bachelor thesis is written as part of the bioinformatics program at Leiden Institute of Advanced Computer Science (LIACS), at Leiden University.

The overall aim of this thesis is to find a way to do phylogenetic placement on a reference tree for ITS fungi data. To be able to motivate this research, first, an introduction to computational biodiversity, metabarcoding, and phylogeny is given. This section ends with the research questions and an overview of the rest of the thesis.

1.1 Computational biodiversity

The diversity of living organisms on earth is referred to as biodiversity [GM22]. The conventional approach to studying biodiversity is through the lens of species diversity, focusing on understanding evolutionary and ecological processes and quantifying patterns among species. However, biodiversity encompasses more than just species diversity. While two regions may have the same number of species, their biodiversity can differ due to variations in the phylogenetic background of those species. Moreover, species names alone lack comprehensive information about their evolution or functional roles within ecosystems. That is why biodiversity is composed of three interconnected components [Swe11] (See Figure 1).

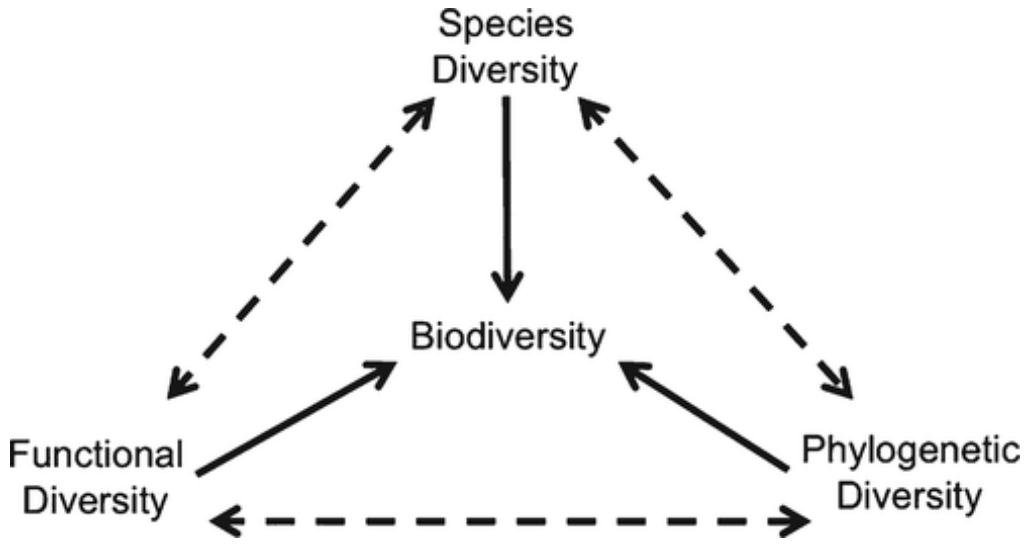


Figure 1: The relations between three important components of biodiversity. Source: American Journal of Botany

Although species diversity is the most familiar component, it is not the sole determinant. Various traits, such as body size, shape, physiological tolerance, or nutritional needs, can be used to represent species diversity. However, the choice of traits influences the similarity degree as certain traits tend to be associated with specific types of organisms [VCMFM11]. Thus, the significance of the other two components and their relationships with the overall biodiversity should not be

overlooked. The other two components are functional diversity, a facet of biodiversity that primarily focuses on the breadth of activities and roles performed by organisms within communities and ecosystems [PG06]. And phylogenetic diversity that takes into account the evolution of the species. These two components are related to each other. When in a region all species are closely related to each other, it means that there is a low phylogenetic diversity. However, this increases the likelihood of low functional diversity since closely related species resemble each other.

The addition of phylogenetic diversity to the measures to determine biodiversity gives rise to new computational ways to calculate biodiversity. Taxonomic and phylogenetic information provides valuable insights into the evolutionary relationships among species [VCMFM11]. By examining the placement of different species within a phylogenetic tree, it becomes possible to draw conclusions about the biodiversity within a specific area. Several algorithms have been developed to assign samples to a reference tree to be able to analyze their positions. In Section 3, we will explore and elaborate on these algorithms in detail.

1.2 Metabarcoding

The Tree of Life aims to represent the evolutionary relationships among organisms. This is an ongoing process with the discovery of new species. Life on earth is divided into three different domains: eubacteria, archaebacteria, and eukaryotes [WKW90]. Fungi are among the largest groups in the domain of the eukaryotes. This makes fungi an important part of the biodiversity, however, only about two to eight percent of the fungi species are discovered right now [OPJ+05]. Fungi are important for the biodiversity of other organisms. The fungi that live in the ground, break down waste and convert it into metabolites that can be used by other organisms. Reduction of the biodiversity of fungi is reduction of soil life and everything that grows on it. Fungi are everywhere, in the soil, in water, and in the air. Since they are everywhere, they play a big role in the ecosystems of the Earth and they are an important field of research.

An important, and recently started research project is the ARISE project. Which is done by Naturalis[1]. They make a mapping of all Dutch species (including fungi) and create an infrastructure to recognize them. To be able to use fungi in this infrastructure, they make use of metabarcoding for fungi.

DNA metabarcoding is a technique to describe the composition of species in a sample. In this case, we look at the metabarcoding of soil samples. DNA barcoding is a sequence-based technique that is designed for rapid species identification, which will accelerate the identification of new species. The technique makes use of a barcode gene. This gene must meet two requirements: i) at the level of species, the variability and divergence of genetics should exhibit substantially ii) feature conserved flanking sites suitable for the development of universal PCR primers with broad taxonomic applicability [KE08]. The first aspect is important since enough variability between marker genes is needed to map a sequence to only one species and allow accurate identification. The second aspect is needed to extract the barcode gene from the sample. Besides these two aspects, it is important that the gene is present in every cell of the organism and that it is easy to detect, even in small DNA quantities. The difference between DNA barcoding and DNA metabarcoding is that DNA barcoding is used to sample individuals and DNA metabarcoding is used to sample large collections [PH20]. DNA metabarcoding uses high-throughput sequencing (HTS) to identify

[1] https://www.naturalis.nl/en/science/arise-knowing-nature-in-the-netherlands

the species in the sample [LCB+20]. Figure 2 shows the workflow from DNA metabarcoding, from collecting the sample to the identification of the species in this sample.

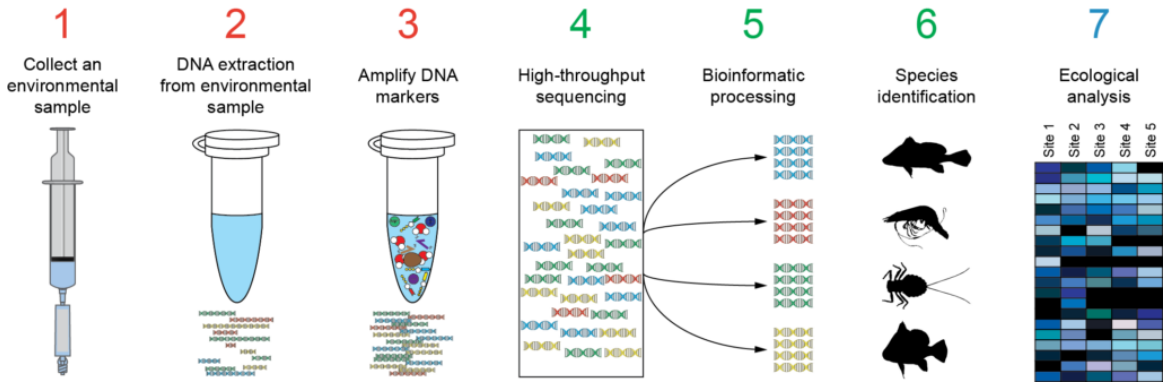


Figure 2: Workflow for DNA metabarcoding [Source: Nature Metrics]

1.2.1 ITS

For fungi, the most suitable marker gene is the internal transcribed spacer (ITS) region, a result of posttranscriptional processing of rRNA cistron. The probability for successful identification is the highest for this marker gene compared to others [SSH+12]. In the past, before the metabarcoding technique was developed, the composition of fungi was determined using morphological traits. With this new technique, also species invisible to the eye can be identified. This improves research into the biodiversity and living environment of fungi. In addition to the identification of more species, DNA also tells more about the origin of the fungi and their evolutionary relationships to other fungi and organisms.

Since the metabarcoding technique, many studies have generated ITS data of fungi. To make them accessible for everyone, different databases are built. One of these databases is the UNITE database[2], which contains all eukaryotic ITS sequences. In this database, sequences with a specific similarity percentage are combined into species hypothesis (SH) [ANL+10].

Another database that contains ITS data is the MycoDiversity DataBase[3] (MDDB). A joint project between Naturalis and Leiden Institute of Advanced Computer Science (LIACS). In this database, the sequence is combined with a location, taxonomic information, and literature written about this sequence [MHK+20]. The database management system behind MDDB is MonetDB. In MonetDB the data is stored in columns instead of rows, which speeds up the retrieval of data significantly compared to row-stored systems [NK12]. Functions defined in SQL, Python, and R can be integrated into the database, allowing analysis of the data. The integration of Python functions allows for data mining and data visualization techniques. Instead of species hypothesis, MDDB uses Zero-radius Operational Taxonomic Units (ZOTUs). Operational Taxonomic Units (OTUs) are clusters of reads with similar sequence similarity. Usually, these clusters are created using a cluster threshold of 97% [SG94]. For ZOTUs, this threshold is 100%, which means that if two sequences are not exactly similar, they are considered to be two different, unique taxonomic units [MHK+20].

[2] https://unite.ut.ee/#main
[3] https://mycodiversity.liacs.nl/

1.3 Phylogeny

Phylogeny depicts the evolutionary relationship between species. In the past, morphological traits were used for this. Nowadays, besides morphological traits DNA sequences are used, because evolution can be clearly seen in the genetic sequence [ZJ12]. To convey the phylogenetic information, it is represented in a branching diagram, also called a phylogenetic tree [BL01]. A phylogenetic tree serves as a valuable tool for gaining insights into both biodiversity and the evolutionary events that have taken place over time [Bau08].

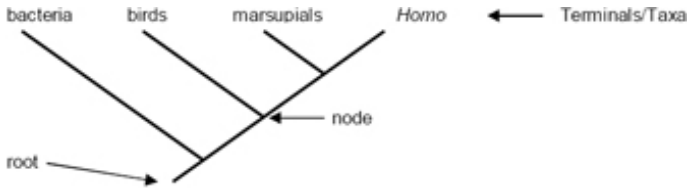


Figure 3: Representation of a phylogenetic tree [Source: Nature Education]

Figure 3 is an example of a rooted phylogenetic tree. Rooted means that all species in the tree descend from that one root, which is the common ancestor of all organisms in the tree. Such a tree consists of different parts [Bau08]. The leaves of the tree represent the different species in the tree. They are also called taxa. A node is a branching point in the tree. The children of the node are descendants with the node as the common ancestor. A branch is a connection between two nodes, this is called an internal branch, or between a node and a leaf, this is called an external branch. In certain phylogenetic trees, the length of branches corresponds to the level of interspecies diversity. Longer branch lengths indicate greater genetic or evolutionary distance between species, indicating they are more distantly related [BL01]. A clade is a distinct segment of a phylogenetic tree that encompasses both an ancestral lineage and all its subsequent descendants. A phylogenetic tree can be represented in different ways. If you can turn a tree into another tree by twisting or rotating branches, then these two trees have the same topology and represent the same evolutionary relationship. This is illustrated in Figure 4.

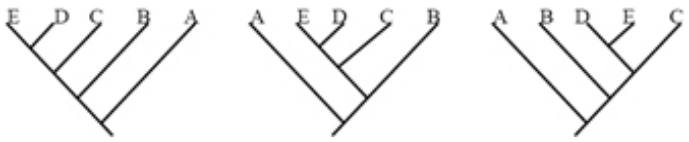


Figure 4: Different trees that show the same relationship [Source: Nature Education]

One way of using phylogenetic trees for biodiversity determination is via phylogenetic placement (Section 3 goes more into detail about phylogenetic placement). This is a more accurate way than a taxonomic read assignment [BM21]. UniFrac is a measure that can be used to compare environmental samples and say something about biodiversity [LLK+11].

1.4 Research questions

The aim of this research is to develop a method to do phylogenetic placement on a reference tree for ITS fungi data. This method can then be used in biodiversity research. This approach can provide valuable insights into the differences in biodiversity between distinct locations or for longitudinal studies.

There already exist different algorithms that can perform phylogenetic placement. However, they are not all suitable for ITS data. That is why first an analysis of these algorithms needs to be performed. After that, a method will be constructed for the placement. Finally, it is required to make a visualization of the results. From this overarching problem statement, the following research question can be formulated:

RQ1: How can the phylogenetic tree be a measure to determine biodiversity?

The above-stated research question can be separated into two subquestions:

RQ1.1: Which suitable algorithms already exist for phylogenetic placement?

RQ1.2: What is an efficient visualization for the outcome?

The method we will construct uses an algorithm for phylogenetic placement. By using this, we expect that if we remove leaves from a tree this algorithm places them back at the correct position. If the method we propose does not work, the leaves will be placed back at the wrong position.

1.5 Thesis overview

Chapter 2 contains the data, software, and hardware that is used for this project. In chapter 3 the different placement algorithms are explained and analyzed. Chapter 4 contains the method that is proposed and the experiments done with this method. A conclusion will be given in chapter 5. This section also contains a discussion and future work.

2 Material & methods

This section describes the data, software, and hardware that is used for this research.

2.1 Data

The data that is used for this research comes from UNITE and MDDB. Luuk Romeijn en Casper Carton created a method to create a reference tree using this data, which is made up of several chunks [CR22]. How the data is retrieved and what it looks like is described in this subsection.

2.1.1 Reference tree MDDB

The reference tree for MDDB is created using ITS data from UNITE and MDDB (See section 1.2). The algorithm that is created has different parameters that the user should specify. The research specifies several different options, and they give two recommendations. The first recommendation: `10.2_s3_4_1500_o1.0_a0_constr_localpair` is used for this research [CR22]. This setting is chosen because it has the most sequences in the tree when looking at the recommendations. The original data that is used for the creation of the reference tree can be found here. The data that is used for this research is downloaded from the github page of Luuk Romeijn. The folder `supertree` contains the data of the whole reference tree. From this folder, only the `backbone.fasta` file is used. This file contains all the sequences and their names and it is used for the BLAST search. There is a total of 23237 sequences in the file.

The folder `chunks` contains the aligned and unaligned FASTA files of the different chunks and the associated tree files. The aligned FASTA files, which contain all the sequences in the chunk, and the corresponding tree files of all chunks are used for this research. These files are used for the BLAST search and the experiments with `pplacer`.

Some details about the data: There are 229 chunks in the reference tree. The length of the chunk, or the number of sequences in the chunk, varies between 4 and 2249 sequences. There are 103 unique lengths for the chunks. The average length of a chunk is 102.472, and the standard deviation is 244.581. The median is 20 and the length that occurs the most is 5, with a frequency of 23. Figure 5 is a histogram of the frequency of the lengths of the chunks.

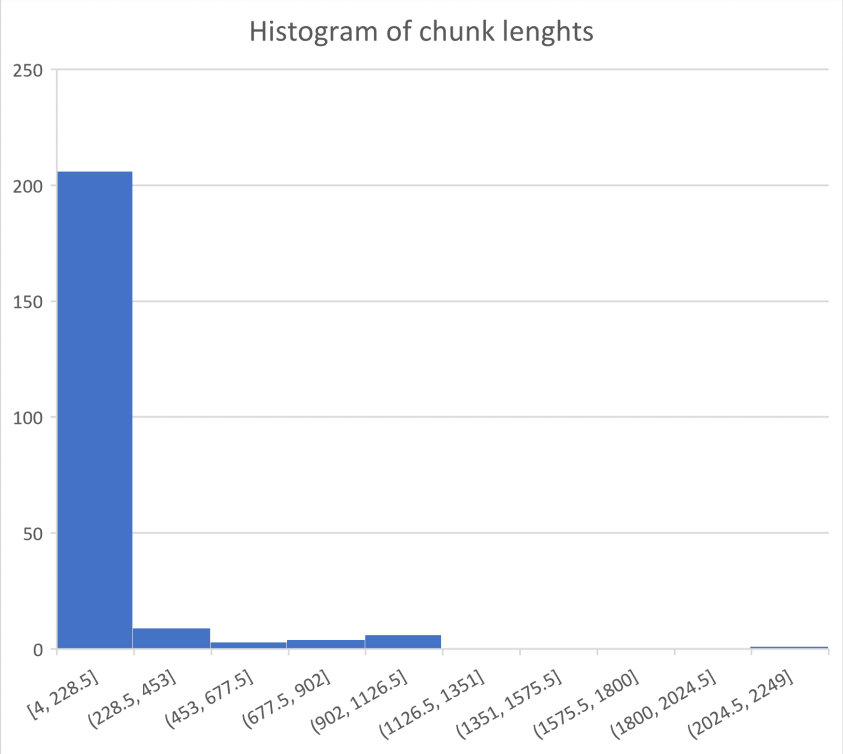


Figure 5: Histogram of the length of the chunks in the reference tree

2.2 Software

For the implementation of the analysis Python version 3.8.2 is used. This allowed the use of different Python packages, such as Biopython, and ete3 for the reading and writing of fasta and Newick tree files and taxtastic for the creation of reference packages.

2.2.1 BLAST

Basic Local Alignment Search Tool (BLAST) is a tool that can be used to find similarities between different sequences. It can compare protein or nucleotide sequences to a general database or a self-created database. For this research BLAST+ version 2.3.0 is used [CCA+09] in the command line.

2.2.2 RAxML

To be able to use `pplacer`, the query should be removed from the existing chunk, and the tree of the chunk should be rebuilt. For this, RAxML (**R**andomized **Ax**elerated **M**aximum **L**ikelihood) version 8.2.12, is used [Sta14]. This is a Maximum Likelihood algorithm that can analyze and generate phylogenetic trees. It generates different trees, each with a likelihood score. The generation of these trees and the calculation is very time-consuming, especially when the number of sequences is large. To create reference packages, the Python package taxtastic version 0.9.3, is used [Bio].

2.2.3 Pplacer

The key component of this research is the phylogenetic placement. This placement is performed by the `pplacer` algorithm version v1.1.alpha19-0-g807f6f3 which is freely available [MKA10]. A large collection of query sequences can be placed on a reference tree using likelihood-based methodology. `pplacer` has two different modes: Maximum Likelihood (ML) mode and Bayesian mode. In ML mode, the likelihood weight ratio (LWR) is computed. To create such a LWR, all the likelihood values from all the different placements are normalized, such that they sum to one. In Bayesian mode, the posterior probability of a placement is calculated. Both modes are used in this research.

`pplacer` requires a fixed reference tree, a reference alignment, and the query sequence(s) as input. The reference tree and the reference alignment are combined into a reference package made with taxtastic (See section: 2.2.2). It gives a single `.jplace` file as output, with the reference tree with numbered branches and the placement(s) of the query sequence(s). For every query sequence, there might be multiple placements when the algorithm was uncertain. The placement with the highest LWR score is the best possible location for the query sequence. The results of `pplacer` can be analyzed and visualized using guppy [MKA14].

2.3 Hardware

The BLAST search, the generation of trees with RAxML, and the phylogenetic placement with `pplacer` are all performed on the MDDB's server by LIACS. This server runs on an Intel(R) Xeon(R) x5355 CPU (8 cores) with 32GB memory. The analysis of the BLAST results, the reference package generation, and the analysis of the `pplacer` output are performed on a laptop. This laptop runs on an Intel(R) Core(TM) i5-8265U CPU (4 cores) with 8GB memory.

3 Placement Methods

There already exist different methods that can perform phylogenetic placement. In phylogenetic placement, a new, unknown sequence, called a query sequence (QS), is placed on a fixed phylogenetic reference tree (RT). This will give researchers insight into how diverse a sample is. For each QS, the branches in the tree that are most related to the sequence are determined. Keeping in mind that the reference tree is fixed, the new QSs are not placed in the tree, they are rather mapped onto it [CSDB22a]. A sample can contain multiple queries. These queries are only aligned against the reference tree and not against each other. Placement of these queries only resolves the phylogenetic relationship between the query and the RT, but not between the different queries [CSDB22b]. The algorithms can be divided into three categories: maximum likelihood methods, distance methods, and other methods. This subsection provides information about all methods and explains several algorithms.

3.1 Maximum likelihood methods

Maximum likelihood methods use maximum likelihood to determine the placement of the query sequence. All methods are computationally intensive. For this research three different algorithms are compared:

3.1.1 pplacer

`pplacer` is a phylogenetic placement algorithm that is written in OCaml, C, and Python. It takes as input a reference tree (RT), a reference alignment (RA) of the reference tree, and a collection of query sequences (QSs). `pplacer` has two different modes: maximum likelihood (ML) mode and Bayesian mode. In ML mode, the QSs are placed on the RT using ML criteria. The output is a file with the RT and the placements of the QSs. These placements consist of the edge number where the QS will be placed, the likelihood score, and the likelihood weight ratio. For this ratio, all likelihood values for the placements are normalized such that they sum to one. In Bayesian mode, the output also contains the posterior probability of the query sequence. `pplacer` has one disadvantage, it fails on reference trees with more than 5000 leaves. [MKA10]

3.1.2 pplacerDC

`pplacerDC` is a phylogenetic placement algorithm that is written in Python. It uses divide and conquer to be able to process bigger trees. First, it divides the reference tree into different subtrees with a bound size. Secondly, it uses `pplacer` to place the query sequences into the subtrees. This gives a candidate tree with the QS placed in each subtree. Each placement is scored with a maximum likelihood score given by RAxML. The best placement is then returned by `pplacerDC` as placement on the reference tree. The input for the algorithm is the same as for `pplacer`. Due to the use of divide and conquer, `pplacerDC` can process reference trees with up to 100.000 leaves. [KPW21]

3.1.3 EPA-NG

`EPA-NG` is a phylogenetic placement algorithm that is written in `C++`. It works in two phases. First, it determines a set of candidate branches where the QS can be placed, this is called preplacement.

In the second phase, the candidate branches are scored more thoroughly with a likelihood score. `EPA-NG` can be parallelized to speed up the process. As input, it requires a multiple sequence alignment (MSA), a reference tree, and QSs aligned against the fixed MSA. As output, it gives back a set of placements on the reference tree and each placement is associated with a LWR. [BKC+19]

3.2 Distance methods

Distance methods use a distance metric to determine the placement of the query sequence. For this research two different algorithms are compared:

3.2.1 APPLES

`APPLES` is a distance-based phylogenetic placement algorithm that is written in Python. `APPLES` can be used on assembly-free and aligned datasets. For assembly-free datasets the distance is calculated using Skmer [SBPG+19]. This model computes the genomic distances. The distances are then corrected using the JC69 model. For aligned datasets, the distance is calculated using JC69. The input of the algorithm is the RT, a QS, and a vector with the distances between the QS and all other species in the RT. The output is a tree that adds the QS to the RT. [BSM20]

3.2.2 App-SpaM

`App-SpaM` (Alignment-free phylogenetic placement algorithm based on Spaced-word Matches) is a distance-based algorithm that is written in `C++`. It requires a RT and the corresponding reference sequences and a set of QSs as input. The distance is calculated using spaced-word matches. For these matches, a binary pattern that describes the match and don't care positions is determined at forehand. This pattern is then used to estimate the distance. Finding spaced-word matches between the RT and a QS is the first step of the algorithm. In the second step, a subset of these spaced-word matches is used to calculate the distances. This distance is then used to determine the position of the QS on the RT. The output is a jplace file with the placements of the QSs. [BSM20]

3.3 Other methods

There exist other methods that do not use distance or maximum likelihood as metrics, but that use something else, like ancestral reconstruction.

3.3.1 RAPPAS

`RAPPAS` (Rapid Alignment-free Phylogenetic Placement via Ancestral Sequences) is an algorithm that is written in JAVA. It uses ancestral reconstruction to determine the placement. Ancestral reconstruction is a way to reason back in time from observed traits of individuals or populations to infer the characteristics of their common ancestors [JLM+16]. The algorithm uses phylogenetically informed k-mers (phylo-k-mers) to determine the placement. As input a RT and reference alignment are required, and the algorithm then builds a database with the phylo-k-mers. By matching the k-mer of the QS to the database, the query can be placed on the tree. A jplace file with the QSs and their LWR is generated as output. [LSP19]

3.4 Analysis

This section contains an analysis of the different methods that can be used for phylogenetic placement.
`pplacer` is a computationally intensive algorithm that fails on reference trees with more than 5000 leaves. It does return an LWR for each query and the involved branch lengths [MKA10]. `pplacerDC` is an improvement of `pplacer`. It uses divide and conquer to be able to process bigger reference trees, but it is still computationally intensive. It is slower than `APPLES`, but the accuracy is better and the algorithm has good scalability [KPW21].

`EPA-NG` is better than `pplacer` when preplacement and premasking are both enabled. It is also faster than `pplacer` and `pplacerDC` on a RT with 1000 leaves. When the number of leaves increases, `pplacerDC` becomes faster. The accuracy of `EPA-NG` is lower than the accuracy of `pplacerDC`, but higher than `APPLES` [KPW21]. The algorithm fails on reference trees with more than 50,000 leaves and it is computationally intensive. On a smaller RT, the algorithm has good scalability for multiple queries [BKC+19]. It returns a LWR and the branch lengths for all queries.

`APPLES` can run on large RT and is fast when placing one query. It has a lower accuracy than maximum likelihood methods and it only returns the position and the branch lengths. There is no uncertainty reported about the placements. `APPLES` can handle assembly-free and aligned datasets [BSM20].

The accuracy of `RAPPAS` is similar to the accuracy of maximum likelihood methods, however, this accuracy is highly influenced by the k-mer size. The first step is building a database for the RT that can be used multiple times. The algorithm can be used with metabarcoding and it returns the LWR of the placements [LSP19].

The last algorithm that is analyzed is `App-SpaM`. This algorithm can use short sequencing reads and returns the placement of the queries without uncertainty but with the branch lengths. It has the fastest runtime compared to `RAPPAS, APPLES, EPA-NG, and pplacer` placing 100,000 queries. A difficulty for the algorithm are partial homologies, which may cause over-estimation of the distance [BM21].

Besides algorithms that perform phylogenetic placement, there are also methods that use k-mers to identify sequences present in a sample [GZR22] [VESB23]. Instead of the reference tree, these methods use a reference database that contains the k-mer profiles specific to each taxonomic unit. However, these methods are not considered in this research.

Table 1 provides a summary of all methods described in sections 3.1, 3.2, 3.3, and 3.4.

3.4.1 Conclusion

There exist several phylogenetic placement methods that are based on different measurement methods. Maximum likelihood methods are computationally expensive but work better for ITS data because they can better handle the gaps. Because the method will be integrated into MDDB, it is important that it is compatible with MonetDB. As stated in section 1.2.1, functions written in Python, R, and SQL can be integrated. That is why for this research `pplacer` is chosen in combination with BLAST. The proposed method is explained in section 4.

Method	ML/ distance/ other	Advantages	Disadvantages	Input	Output	Language
pplacer	ML	X	Computationally intensive. Fails on RT with >5000 leaves.	RT, RA and a collection of QSs	RT with QS and their LWR	OCaml, C, Python
pplacerDC	ML	Uses D&C for big trees. Slower than APPLES, but higher accuracy and good scalability	Computationally intensive	RT, RA and a collection of QSs	RT with the placed QS	Python
EPA-NG	ML	Better than pplacer when preplacement and premasking are used	Fails on RT with >50.000 leaves. Computationally intensive.	MSA, RT, and QSs aligned against fixed MSA	Set of placements on the RT, each with an LWR	C++
APPLES	Kmer-based distance	Can run on large RT. Scalable method.	Lower accuracy than ML methods	RT, QS, and vector with distances between QS and species in RT	RT with the placed QS	Python
App-SpaM	Phylogenetic distance	Can use short sequencing reads	Partial homologies may cause over-estimation of distances.	A set of RSs, an RT, and a set of QSs	JPlace file containing all the placements of the QSs	C++
RAPPAS	Ancestral reconstruction	Builds a DB that is used for placement. Can be used with metabarcoding.	Accuracy is highly influenced by the k-mer size.	Input for DB: RA and RT	JPlace file with placements of QSs (with LWR)	JAVA

Table 1: Summary of the different phylogenetic placement algorithms. ML: Maximum Likelihood, RT: Reference Tree, QS: Query Sequence, LWR: Likelihood Weight Ratio, RA: Reference Alignment, RS: Reference Sequence, DC: Divide and Conquer, MSA: Multiple Sequence Alignment

4 Experiments and Results

Figure 6 is an overview of the method that is tested for the phylogenetic placement. The method is divided into two parts: BLAST and `pplacer`. BLAST is used to determine the correct subtree and `pplacer` is then executed on the smaller subtree. The red and blue rectangles are shown in more detail in the corresponding subsections. The method is validated by calculating the distances and the number of nodes.
The method is tested with three different subsets of the data, with each 25 randomly selected queries. The third test contains next to the 25 new samples, also three samples from the first (denoted with a * in the results) and three samples from the second test (denoted with a + in the results). The `pplacer` part of the method is run three times for each test to determine if the algorithm is stochastic, or deterministic.

This section contains an in-depth explanation of the different parts of the method.

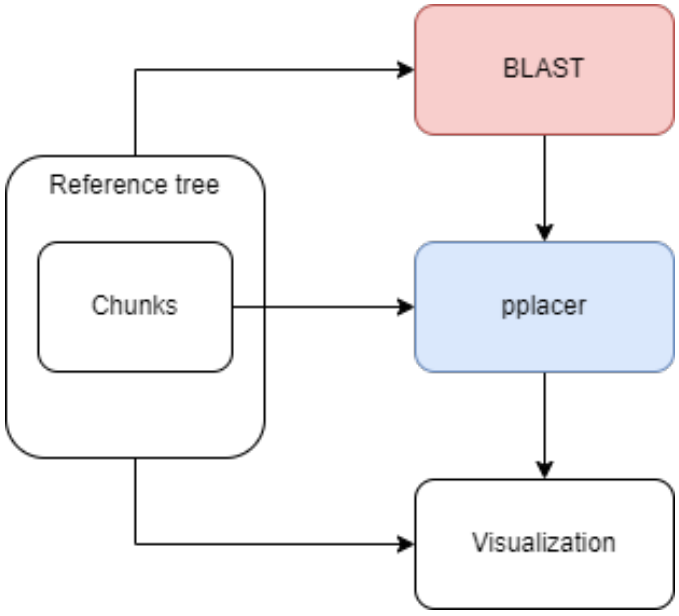


Figure 6: Overview of the method for phylogenetic placement. First BLAST is used to identify the subtree. Then `pplacer` is used to do the phylogenetic placement.

4.1 BLAST

Determining the correct subtree is an important step for the phylogenetic placement. Since `pplacer` can not handle trees with more than 5000 leaves, we need to find a way to use `pplacer` on a smaller tree. The reference tree is constructed using different chunks. How BLAST is used to determine the correct subtree and the results of this are described in this subsection.

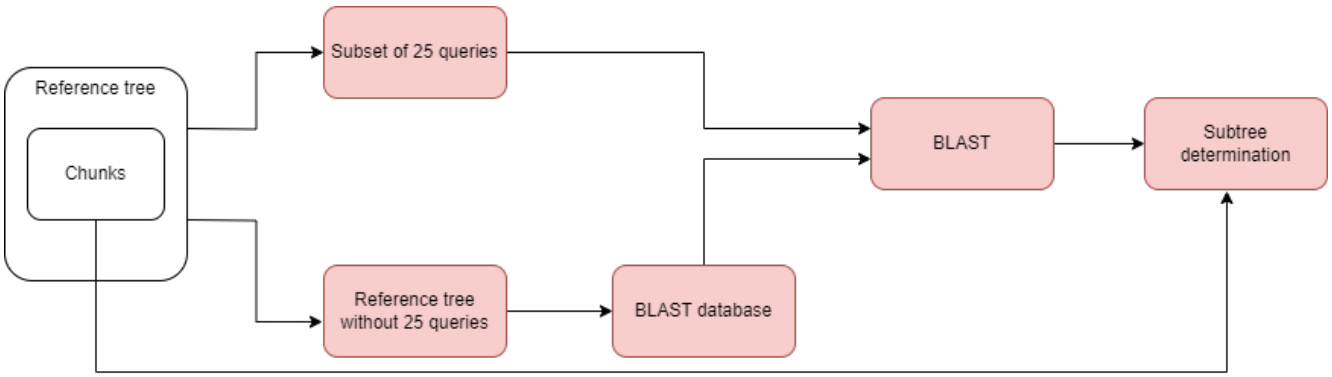


Figure 7: A detailed representation of the first part of the method, using BLAST.

4.1.1 Method BLAST

Figure 7 is a graphical representation of the first part of the method. It is a detailed representation of the red rectangle in figure 6. The reference tree is divided into two parts. A subset of 25 queries that will be used for testing, and the remaining reference tree without the 25 queries. A BLAST database is made from the remaining reference tree. The following command is used for this:

```
makeblastdb -in data.faa -out database_name -dbtype nucl -title 'database title'
-parse_seqids,
```

where *data.faa* is the fasta file that contains all sequences for the database. `-parse_seqids` is specified in order to keep the original sequence ids.

After that, the query sequences are blasted against the created BLAST database. The command that is used for this is as follows:

```
blastn -query test.faa -db database_name -out output.csv -outfmt '7 qseqid
sseqid pident evalue length qcovhsp' -max_target_seqs 10,
```

where *test.faa* is the fasta file that contains all the query sequences, *output.csv* is a CSV file that is generated as output and it has 6 columns: query id, subject id, % identity, ϵ-value, alignment length and % query coverage per HSP. For each query sequence, the top ten BLAST hits are saved in the output file.

4.1.2 Results BLAST

The determination of the original subtree is done for every BLAST hit for every query. The number of hits in a subtree is counted and after the ten hits, the subtree with the most hits is compared to the original subtree where the query was in to see if the correct subtree was found. Table 3 is the result of test one. It denotes the query id, the original subtree, in which subtree(s) there were BLAST hits, the number of BLAST hits in these subtree(s), and if the original subtree is the same tree as the subtree with the most BLAST hits. For this test for only one of the 25 queries not the correct subtree was chosen. This is denoted with an orange colour in the table. Table 4 is the result of test two and table 5 is the result of test three.

Table 2 is a summary of the results of all the tests. In test one and test three, there is only one query where we did not end up in the correct subtree after the BLAST search. In test one, 77% of the queries had all hits in the same, correct tree. For test two and test three this is 88% and 77% respectively.

	All hits in 1 subtree	Multiple subtrees	Correct	Wrong	Number of queries
Test 1	19	6	24	1	25
Test 2	22	3	25	0	25
Test 3	24	7	30	1	31

Table 2: Summary of the results of all tests. Test three contains 25 random queries and three randomly selected queries from test one and three from test two

Query id	Original tree	Subtree(s)	Hits BLAST	Correct
SH1237664.08FU_JN685254_reps	Glomerales	Glomerales	10	Yes
SH1140508.08FU_AB634264_refs	Thelephorales	Thelephorales	10	Yes
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	Hygrophoraceae	10	Yes
SH1170869.08FU_UDB015275_refs	Tricholomataceae	Tricholomataceae	10	Yes
SH1235509.08FU_KU663969_reps	Hymenogastraceae	Hymenogastraceae	10	Yes
*SH1145193.08FU_EU222979_reps	Russulales	Russulales	10	Yes
SH1218065.08FU_UDB011825_refs	Geastrales	Geastrales	10	Yes
SH2605620.08FU_UDB0454851_reps	Cystobasidiales	Cystobasidiomycetes_ord_Incertae_sedis, Erythrobasidiales, Cystobasidiales	1, 1, 8	Yes
SH3220109.08FU_UDB0625009_reps	Entylomatales	Hymenochaetales, Entylomatales, Microstromatales	1, 3, 6	No
SH1184486.08FU_UDB0244450_reps	Geminibasidiales	Tremellales, Cystofilobasidiales, Tritirachiales, Geminibasidiales	1, 1, 1, 7	Yes
SH2613479.08FU_UDB0408767_reps	Capnodiales	Capnodiales	10	Yes
SH2601717.08FU_UDB0237225_reps	Pleosporales	Pleosporales	10	Yes
SH1152427.08FU_MG029166_reps	Helotiales	Helotiales	10	Yes
*SH1189121.08FU_MG920147_reps	Helotiales	Helotiales	10	Yes
SH1168071.08FU_KU945909_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1166278.08FU_MK841471_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1150620.08FU_MF278325_reps	Trichocomaceae	Aspergillaceae, Trichocomaceae	1, 9	Yes
SH1190485.08FU_KX828139_reps	Xylariales	Diaporthales, Xylariales	2, 8	Yes
SH1149544.08FU_AY425628_reps	Lecanorales	Lecanorales	10	Yes
SH1215154.08FU_LN810821_reps	Acarosporales	Acarosporales	10	Yes
*SH1152267.08FU_KR017144_reps	Trapeliales	Trapeliales	10	Yes
SH1160442.08FU_HQ650658_reps	Lecideales	Lecanorales, Lecideales	1, 9	Yes
SH1160493.08FU_UDB0260931_reps	Saccharomycetales	Saccharomycetales	10	Yes
SH1177296.08FU_UDB0231445_reps	Mortierellales	Mortierellales	10	Yes
SH1180624.08FU_KP744428_reps	Mortierellales	Mortierellales	10	Yes

Table 3: The results of the subtree determination of the first test. Query is the id of the query which is blasted against the database. Original tree is the original subtree of the query. Subtree(s) are the trees with BLAST hits of the specific query. Hits BLAST are the number of BLAST hits in the subtree(s) and correct is if the original tree is the same as the subtree with the most BLAST hits. The orange colour denotes the query that did not return the same subtree as the original subtree. The asterisk (*) highlights the queries that are tested again in test three.

Query id	Original tree	Subtree(s)	Hits BLAST	Correct
SH1159594.08FU_UDB0335978_reps	Thelephorales	Thelephorales	10	Yes
SH1159486.08FU_KF476884_reps	Thelephorales	Thelephorales	10	Yes
SH1140827.08FU_AY969759_reps	Thelephorales	Thelephorales	10	Yes
SH1245769.08FU_MN007027_reps	Polyporales	Polyporales	10	Yes
SH1238102.08FU_JN105716_reps	Polyporales	Polyporales	10	Yes
+SH1190303.08FU_MH926037_reps	Polyporales	Polyporales	10	Yes
SH1145476.08FU_UDB007576_reps	Boletales	Boletales	10	Yes
SH1184430.08FU_UDB0767606_reps	Russulales	Russulales	10	Yes
SH1191708.08FU_MF496149_reps	Russulales	Russulales	10	Yes
+SH2614834.08FU_UDB0680020_reps	Tremellales	Tremellales	10	Yes
SH1238165.08FU_MH861181_refs	Pleosporales	Pleosporales	10	Yes
SH1150688.08FU_KF800188_reps	Pleosporales	Pleosporales	10	Yes
SH1156865.08FU_AF486126_refs	Helotiales	Helotiales	10	Yes
SH1166814.08FU_EF669705_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1162773.08FU_KT310978_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1162466.08FU_HQ891869_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1166968.08FU_KY978354_reps	Aspergillaceae	Aspergillaceae	10	Yes
SH1235707.08FU_KY051892_reps	Diaporthales	Diaporthales	10	Yes
+SH1191717.08FU_MF488989_refs	Xylariales	Xylariales	11	Yes
SH1142770.08FU_KT949894_reps	Xylariales	Sordariales, Xylariales	1, 9	Yes
SH1183141.08FU_GU903288_reps	Magnaporthales	Hypocreales, Magnaporthales	1, 9	Yes
SH1228780.08FU_KY940500_reps	Magnaporthales	Glomerellales, Xylariales, Magnaporthales	1, 1, 8	Yes
SH1172583.08FU_KR912053_reps	Teloschistales	Teloschistales	10	Yes
SH1183613.08FU_KJ707198_reps	Saccharomycetales	Saccharomycetales	10	Yes
SH1172915.08FU_JQ906769_refs	Saccharomycetales	Saccharomycetales	10	Yes

Table 4: The results of the subtree determination of the second test. Query is the id of the query which is blasted against the database. Original tree is the original subtree of the query. Subtree(s) are the trees with BLAST hits of the specific query. Hits BLAST are the number of BLAST hits in the subtree(s) and correct is if the original tree is the same as the subtree with the most BLAST hits. The plus (+) highlights the queries that are tested again in test three.

Query id	Original tree	Subtree(s)	Hits BLAST	Correct
SH2591562.08FU_MK348933_reps	Glomerales	Glomerales	10	Yes
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	Hygrophoraceae	10	Yes
SH1180966.08FU_KT429785_refs	Hygrophoraceae	Hygrophoraceae	10	Yes
SH1235601.08FU_FJ943242_reps	Hygrophoraceae	Hygrophoraceae	10	Yes
SH3233617.08FU_MT883671_reps	Amanitaceae	Cortinariaceae, Amanitaceae, Tricholomataceae, Callistosporiaceae, Lyophyllaceae	1, 1, 3, 3, 2	No
SH1168537.08FU_LC056769_reps	Amanitaceae	Amanitaceae	10	Yes
SH2593743.08FU_MW077527_reps	Pluteaceae	Pluteaceae, Lyophyllaceae, Hymenochaetales	5, 3, 2	Yes
+SH1190303.08FU_MH926037_reps	Polyporales	Polyporales	10	Yes
SH1174831.08FU_LC368018_reps	Russulales	Russulales	10	Yes
*SH1145193.08FU_EU222979_reps	Russulales	Russulales	10	Yes
SH2602692.08FU_UDB0613842_reps	Auriculariales	Auriculariales	10	Yes
SH1173571.08FU_UDB0764892_reps	Auriculariales	Auriculariales	10	Yes
+SH2614834.08FU_UDB0680020_reps	Tremellales	Tremellales	10	Yes
SH1235583.08FU_HQ874756_reps	Botryosphaeriales	Botryosphaeriales	10	Yes
SH1145889.08FU_UDB028521_reps	Helotiales	Helotiales	10	Yes
SH1155345.08FU_HM230882_reps	Helotiales	Helotiales, Phacidiales	9, 1	Yes
*SH1189121.08FU_MG920147_reps	Helotiales	Helotiales	10	Yes
SH1162906.08FU_AF272574_refs	Aspergillaceae	Aspergillaceae	10	Yes
SH1194552.08FU_UDB092932_reps	Sordariales	Sordariales	10	Yes
SH1173602.08FU_HM484543_refs	Hypocreales	Hypocreales	10	Yes
SH1187931.08FU_MH858276_reps	Hypocreales	Hypocreales	10	Yes
+SH1191717.08FU_MF488989_refs	Xylariales	Xylariales	11	Yes
SH1153681.08FU_MT185508_reps	Chaetosphaeriales	Chaetosphaeriales, Phomatosporales	10, 1	Yes
SH2604690.08FU_UDB090068_reps	Pleurotheciales	Pleurotheciales	10	Yes
SH1144564.08FU_HQ889706_reps	Boliniales	Coniochaetales, Sordariales, Boliniales	1, 3, 6	Yes
SH1169414.08FU_AJ458289_reps	Ostropales	Ostropales, Lecanorales, Umbilicariales, Pertusariales	6, 1, 2, 1	Yes
*SH1152267.08FU_KR017144_reps	Trapeliales	Trapeliales	10	Yes
SH1174889.08FU_KR902678_reps	Teloschistales	Teloschistales	10	Yes
SH2608872.08FU_UDB0209524_reps	GS36	Capnodiales, Lecanoromycetes_ord_Incertae_sedis, GS36	4, 1, 5	Yes
SH1178035.08FU_EU836707_refs	Saccharomycetales	Saccharomycetales	10	Yes
SH1155115.08FU_KC489498_refs	Umbelopsidales	Umbelopsidales	10	Yes

Table 5: The results of the subtree determination of the third test. Query is the id of the query which is blasted against the database. Original tree is the original subtree of the query. Subtree(s) are the trees with BLAST hits of the specific query. Hits BLAST are the number of BLAST hits in the subtree(s) and correct is if the original tree is the same as the subtree with the most BLAST hits. The orange colour denotes the query that did not return the same subtree as the original subtree. The asterisk (*) and the plus (+) highlight the sequences that are also tested in tests one and two respectively.

4.2 PPLACER

The second part is the actual placement of the query sequences. The reference tree is reduced in size by selecting a subtree, so `pplacer` can be used for the placement. Before `pplacer` can be run, there are some steps to be taken. Figure 8 is a graphical representation of the second part of the method. It is the detailed representation of the blue `pplacer` rectangle in figure 6. The preprocessing steps for `pplacer`, the use of `pplacer`, and the analysis of the result are described in this section.

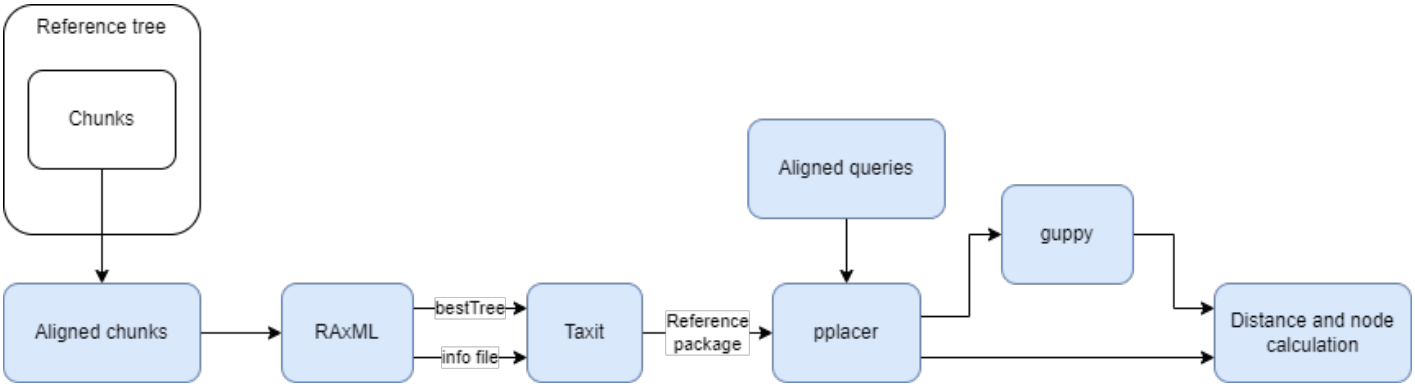


Figure 8: A detailed representation of the second part of the method, using `pplacer`

4.2.1 Method pplacer

The query sequences are removed from the aligned fasta files of the chunks. They are also removed from the corresponding tree files. However, when these modified files were given as input to `pplacer`, a parsing error was encountered that was unsolvable. As a result of this unresolved issue, the decision was made to employ RAxML for the purpose of reconstructing the subtree. The modified fasta files are the input for RAxML. `pplacer` requires a reference package as input, so that needs to be created first. RAxML is used to rebuild the chunk and to generate an information file about this tree. RAxML requires a file with aligned sequences as input and returns multiple files. The *RAxML_info* and *RAxML_bestTree* files are of importance for this research. The *RAxML_info* file contains the final GAMMA-based likelihoods and the α shape parameters. It also denotes how RAxML was called and some information about the algorithm. The *RAxML_bestTree* file contains the best phylogenetic tree with the highest maximum likelihood score. These two files are used to create a reference package for `pplacer`. For this research, RAxML is called as follows:

```
./raxmlHPC-PTHREADS-SSE3 -s input_file -n file_name.out -w output_
dir -m GTRGAMMA -p 12345 -T 8,
```

where *input_file* is the name of the input file, *file_name* is the name of the output file, and *output_dir* is the name of the output directory. GTRGAMMA is the model that is used for the substitution, `-p 12345` is the parsimony random seed and `-T` is the number of threads that should be used.

The reference packages are created using taxtastic. This package requires three files as input and returns a reference package. The package is called as follows:

```
taxit create -l its -P package_name.refpkg --aln-fasta input_file.fasta
--tree-stats RAxML_info.file_name.out --tree-file RAxML_bestTree.file
_name.out
```

After the preprocessing steps, the next step is to run `pplacer`. It is called as follows:

```
pplacer -c refpkg_file query_file -o query_name.jplace
```

refpkg_file is the reference package including the reference tree and alignment, *query_file* is the file that contains the aligned query or queries, and *query_name* is the output file as `.jplace`. To run `pplacer` in Bayesian mode, a `-p` is added to the end of the command.

For analyzing the performance of `pplacer`, a tree with the query sequence placed is needed. This tree can be generated using the `guppy tog` command [MKA14]. This takes the best placement and puts this in the tree. The command that is used is as follows:

```
guppy tog -o file_name.tog.tre file_name.jplace,
```

where *file_name* is the name of the corresponding `.jplace` file generated by `pplacer`. Both the .jplace and the .tog.tre files are used for the distance and nodes calculation. The results are shown in the next section.

4.2.2 Results pplacer

Figure 9 shows the output of `pplacer` for one of the chunks. The first element of the output is the tree with the edges numbered. The second part is the placement of the query sequence(s). For this tree there was one query sequence, that could be placed in five different locations. The placement with the highest likelihood weight ratio is the best possible location. The metadata part denotes how `pplacer` was called and what the field names are.

```
{"tree":
  "(SH1184487.08FU_JX242877_refs:0.0223892{0},(SH2611405.08FU_UDB0564822
_reps:0.0218262{1},((SH2617718.08FU_UDB0672727
_reps:0.0561952{2},(SH2618257.08FU_UDB0663474
_reps:0.0367245{3},(SH1179440.08FU_KC751417
_refs:0.0425551{4},SH1179439.08FU_JX242863
_refs:0.0198067{5}):0.263801{6}):0.0171428{7}):1e-06{8},OUTGROUP:1.09924
{9}):0.00741618{10}):0.00756909{11},SH1184488.08FU_JX242880
_refs:0.0268434{12}):0{13};",
  "placements":
  [
    {"p":
      [[0.0128232450226, 7, 0.349640615308, -2186.5528836,
0.0219904662511],
        [0.0531899486064, 2, 0.27234741903, -2186.80271089,
0.0218130156369],
        [5.00000250015e-07, 8, 0.187544424428, -2187.17577367,
          0.0244054943917
        ],
        [7.24236660387e-06, 10, 0.18679592165, -2187.17977272,
          0.0244080779167
        ],
        [0.263706116437, 6, 0.00367161958372, -2191.10915655,
0.0201511074504
        ]
      ], "nm": [["SH1184486.08FU_UDB0244450_reps", 1]]
    }
  ], "metadata":
  {"invocation":
    "pplacer -c \/home\/s2566818\/packages\/query_100.refpkg \/home
\/s2566818\/queries\/queries\/query_100.fasta -o query_100.jplace"
  }, "version": 3, "fields":
  ["distal_length", "edge_num", "like_weight_ratio", "likelihood",
    "pendant_length"
  ]
}
```


Figure 9: Output of `pplacer`, as jplace file, for one of the chunks. The tree part contains the tree with edges numbered. The placements part contains information about the placed query sequence(s) and the metadata part contains information about how `pplacer` is called.

Table 6 is the summarized output of `pplacer` for test one. It contains the query id of the placed sequence, the subtree in which it is placed, the likelihood weight ratio (LWR) for the best placement, the number of possible positions for the query sequence and the posterior probability for the best placement. The gray columns are the output of `pplacer` in maximum likelihood mode, the green column is added when `pplacer` is run in Bayesian mode. Table 7 is the summarized output of `pplacer` for test two and Table 8 is the summarized output of `pplacer` for test three.

Query id	Subtree	LWR	Number of placements	Posterior probability
SH1237664.08FU_JN685254_reps	Glomerales	0.996557	3	0.997709
SH1140508.08FU_AB634264_refs	Thelephorales	1.0	1	1.0
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	0.323651	9	0.388831
SH1170869.08FU_UDB015275_refs	Tricholomataceae	0.926914	8	0.923368
SH1235509.08FU_KU663969_reps	Hymenogastraceae	1.0	1	1.0
*SH1145193.08FU_EU222979_reps	Russulales	0.997733	2	0.994696
SH1218065.08FU_UDB011825_refs	Geastrales	0.543687	7	0.54309
SH2605620.08FU_UDB0454851_reps	Cystobasidiales	0.218069	10	0.223116
SH1184486.08FU_UDB0244450_reps	Geminibasidiales	0.29354	7	0.401973
SH2613479.08FU_UDB0408767_reps	Capnodiales	1.0	1	1.0
SH2601717.08FU_UDB0237225_reps	Pleosporales	0.161761	15	0.241216
*SH1189121.08FU_MG920147_reps	Helotiales	0.994604	4	0.995463
SH1152427.08FU_MG029166_reps	Helotiales	0.437745	3	0.539873
SH1168071.08FU_KU945909_reps	Aspergillaceae	0.05	20	0.05
SH1166278.08FU_MK841471_reps	Aspergillaceae	0.052317	20	0.055319
SH1150620.08FU_MF278325_reps	Trichocomaceae	0.348697	16	0.262837
SH1190485.08FU_KX828139_reps	Xylariales	1.0	1	1.0
SH1149544.08FU_AY425628_reps	Lecanorales	0.949805	3	0.967509
SH1215154.08FU_LN810821_reps	Acarosporales	0.401506	5	0.408583
*SH1152267.08FU_KR017144_reps	Trapeliales	0.948184	3	0.986195
SH1160442.08FU_HQ650658_reps	Lecideales	0.6031	10	0.644485
SH1160493.08FU_UDB0260931_reps	Saccharomycetales	1.0	1	1.0
SH1177296.08FU_UDB0231445_reps	Mortierellales	0.916382	3	0.955286
SH1180624.08FU_KP744428_reps	Mortierellales	1.0	1	1.0

Table 6: The result of `pplacer` for test one. Query id is the id of the query that is placed. The subtree is the subtree in which the query is placed. LWR is the likelihood weight ratio for the best placement. Number of placements is the number of positions `pplacer` determined for the query. Posterior probability is the posterior probability value for the best placement. Gray is the output of `pplacer` in maximum likelihood mode, the green column is added when `pplacer` is run in Bayesian mode. The asterisk (*) highlights the queries that are tested again in test three.

Query id	Subtree	LWR	Number of placements	Posterior probability
SH1159486.08FU_KF476884_reps	Thelephorales	0.923243	7	0.924643
SH1140827.08FU_AY969759_reps	Thelephorales	1.0	1	1.0
SH1159594.08FU_UDB0335978_reps	Thelephorales	0.160873	20	0.258652
SH1245769.08FU_MN007027_reps	Polyporales	0.893283	10	0.962162
+SH1190303.08FU_MH926037_reps	Polyporales	0.253029	9	0.236485
SH1238102.08FU_JN105716_reps	Polyporales	1.0	1	1.0
SH1145476.08FU_UDB007576_reps	Boletales	1.0	1	1.0
SH1184430.08FU_UDB0767606_reps	Russulales	0.439869	6	0.515159
SH1191708.08FU_MF496149_reps	Russulales	1.0	1	1.0
+SH2614834.08FU_UDB0680020_reps	Tremellales	1.0	1	1.0
SH1150688.08FU_KF800188_reps	Pleosporales	1.0	1	1.0
SH1238165.08FU_MH861181_refs	Pleosporales	0.606459	3	0.604346
SH1156865.08FU_AF486126_refs	Helotiales	1.0	1	1.0
SH1166968.08FU_KY978354_reps	Aspergillaceae	0.05	20	0.05
SH1162773.08FU_KT310978_reps	Aspergillaceae	0.870022	20	0.866402
SH1162466.08FU_HQ891869_reps	Aspergillaceae	0.05	20	0.05
SH1166814.08FU_EF669705_reps	Aspergillaceae	0.800474	3	0.34765
SH1235707.08FU_KY051892_reps	Diaporthales	0.451275	11	0.60628
SH1142770.08FU_KT949894_reps	Xylariales	1.0	1	1.0
+SH1191717.08FU_MF488989_refs	Xylariales	1.0	1	1.0
SH1183141.08FU_GU903288_reps	Magnaporthales	1.0	1	1.0
SH1228780.08FU_KY940500_reps	Magnaporthales	1.0	1	1.0
SH1172583.08FU_KR912053_reps	Teloschistales	0.881873	5	0.893487
SH1172915.08FU_JQ906769_refs	Saccharomycetales	0.427598	5	0.374877
SH1183613.08FU_KJ707198_reps	Saccharomycetales	0.482296	5	0.430322

Table 7: The result of `pplacer` for test two. Query id is the id of the query that is placed. The subtree is the subtree in which the query is placed. LWR is the likelihood weight ratio for the best placement. Number of placements is the number of positions `pplacer` determined for the query. Posterior probability is the posterior probability value for the best placement. Gray is the output of `pplacer` in maximum likelihood mode, the green column is added when `pplacer` is run in Bayesian mode. The plus (+) highlights the queries that are tested again in test three.

Query id	Subtree	LWR	Number of placements	Posterior probability
SH2591562.08FU_MK348933_reps	Glomerales	1.0	1	1.0
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	0.322088	8	0.386894
SH1235601.08FU_FJ943242_reps	Hygrophoraceae	1.0	1	1.0
SH1180966.08FU_KT429785_refs	Hygrophoraceae	1.0	1	1.0
SH1168537.08FU_LC056769_reps	Amanitaceae	1.0	1	1.0
SH2593743.08FU_MW077527_reps	Pluteaceae	1.0	1	1.0
+SH1190303.08FU_MH926037_reps	Polyporales	0.279693	9	0.26051
*SH1145193.08FU_EU222979_reps	Russulales	0.997893	2	0.99507
SH1174831.08FU_LC368018_reps	Russulales	1.0	1	1.0
SH2602692.08FU_UDB0613842_reps	Auriculariales	0.991303	2	0.991647
SH1173571.08FU_UDB0764892_reps	Auriculariales	1.0	1	1.0
+SH2614834.08FU_UDB0680020_reps	Tremellales	1.0	1	1.0
SH1235583.08FU_HQ874756_reps	Botryosphaeriales	0.616152	6	0.626686
SH1145889.08FU_UDB028521_reps	Helotiales	0.629805	3	0.62979
*SH1189121.08FU_MG920147_reps	Helotiales	0.944242	6	0.956278
SH1155345.08FU_HM230882_reps	Helotiales	0.682116	10	0.777626
SH1162906.08FU_AF272574_refs	Aspergillaceae	0.332917	5	0.399174
SH1194552.08FU_UDB092932_reps	Sordariales	0.785177	3	0.822465
SH1187931.08FU_MH858276_reps	Hypocreales	0.47998	13	0.471618
SH1173602.08FU_HM484543_refs	Hypocreales	0.977011	3	0.988748
+SH1191717.08FU_MF488989_refs	Xylariales	1.0	1	1.0
SH1153681.08FU_MT185508_reps	Chaetosphaeriales	0.590695	7	0.6278
SH2604690.08FU_UDB090068_reps	Pleurotheciales	0.666984	7	0.530907
SH1144564.08FU_HQ889706_reps	Boliniales	0.372726	3	0.374636
SH1169414.08FU_AJ458289_reps	Ostropales	0.464336	5	0.227834
*SH1152267.08FU_KR017144_reps	Trapeliales	0.948184	3	0.986195
SH1174889.08FU_KR902678_reps	Teloschistales	0.951633	4	0.93688
SH2608872.08FU_UDB0209524_reps	GS36	0.471723	4	0.559123
SH1178035.08FU_EU836707_refs	Saccharomycetales	0.481467	3	0.545292
SH1155115.08FU_KC489498_refs	Umbelopsidales	1.0	1	1.0

Table 8: The result of `pplacer` for test three. Query id is the id of the query that is placed. The subtree is the subtree in which the query is placed. LWR is the likelihood weight ratio for the best placement. Number of placements is the number of positions `pplacer` determined for the query. Posterior probability is the posterior probability value for the best placement. Gray is the output of `pplacer` in maximum likelihood mode, the green column is added when `pplacer` is run in Bayesian mode. The asterisk (*) and the plus (+) highlight the sequences that are also tested in tests one and two respectively.

4.2.3 Distance and nodes calculation

To make sure the method proposed gives reliable and reproducible results, the method must be validated. This is done using two different metrics: the distance between two leaves and the number of nodes between two leaves in the original tree and in the new `pplacer` tree. Both methods and the results of this are described in this section.
For each query sequence, the neighbour is searched in the original tree. For example, if leaf F is the query sequence, then the sisters of the leaf are searched. Sisters are the other children of the most recent common ancestor (MRCA) (green dot in Figure 10). This can return a leaf or a clade. In this example a clade is returned, so we have to search deeper for a neighbour. First, the left child of the clade is examined if it is a clade or a leaf. If it is a leaf, the neighbour is found. If it is a clade, the right child is examined in the same way. If left and right are both clades, the left clade is chosen and the search starts again until a neighbour is found. Since the rotation or twisting of the tree does not change topology (see section 1.3), the two sequences do not have to be real neighbours as long as they have the same MRCA. So, for this example, the neighbour of leaf F would be leaf I.

A measure of calculating phylogenetic biodiversity (PD) is using Faith's PD [Fai92]. For this measure, the branch lengths from the leaf to the root of all species of interest are summed together. Changes in this number indicate changes in biodiversity. For this research, an adjusted version of Faith's PD is used. The distance used is the sum of the branch lengths of all branches between the two nodes. So, the distance between nodes C and B in figure 10 is $0.15 + 0.1 + 0.2 = 0.45$ (see the yellow lines in the figure). The distance is normalized by dividing it by the longest branch in the chunk. This allows for the comparison of distances between different chunks. The number of nodes between nodes C and H in figure 10 is four (see the red dots in the figure).

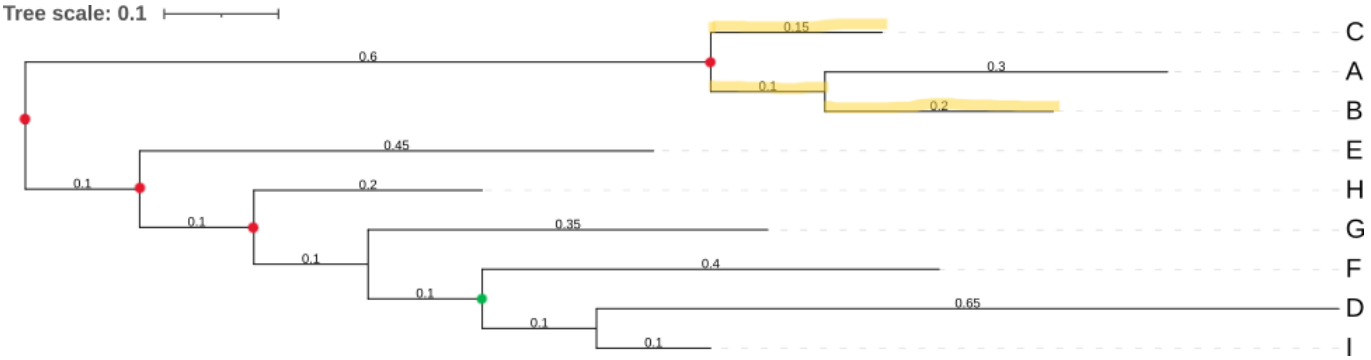


Figure 10: An example of how the distance and the number of nodes in a tree is calculated. The yellow lines are summed for the distance between leaf C and B. The red dots are the number of nodes between leaf C and H. The green dot is the common ancestor of leaf F, D, and I.

The distance and the number of nodes between the query sequence and its neighbour are calculated in the original tree and in the tree with the query sequence placed. The absolute difference in distance and number of nodes is shown in table 9. The colours of the row correspond with the colours of the leaves in figure 14. Each chunk has its own colour. For test two the results are shown in table 10 and the colours correspond with the leaves in figure 15. The results of test three are shown in table 11 and the colours correspond to the leaves in figure 16.

Query id	Subtree	Difference in nodes	Difference in distance
SH1237664.08FU_JN685254_reps	Glomerales	0	0.00949807
SH1160493.08FU_UDB0260931_reps	Saccharomycetales	0	0.00453032
*SH1152267.08FU_KR017144_reps	Trapeliales	0	0.00341981
SH1149544.08FU_AY425628_reps	Lecanorales	0	0.05772156
SH1190485.08FU_KX828139_reps	Xylariales	0	0.0121388
SH1150620.08FU_MF278325_reps	Trichocomaceae	0	0.00196334
*SH1189121.08FU_MG920147_reps	Helotiales	0	0.00032693
SH1177296.08FU_UDB0231445_reps	Mortierellales	0	0.00642654
SH2613479.08FU_UDB0408767_reps	Capnodiales	0	0.00476237
SH1180624.08FU_KP744428_reps	Mortierellales	0	0.01025783
SH2605620.08FU_UDB0454851_reps	Cystobasidiales	0	0.0341218
SH1218065.08FU_UDB011825_refs	Geastrales	0	0.01333243
*SH1145193.08FU_EU222979_reps	Russulales	0	0.00504073
SH1235509.08FU_KU663969_reps	Hymenogastraceae	0	0.0716281
SH1170869.08FU_UDB015275_refs	Tricholomataceae	0	0.01245505
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	0	0.038016
SH1140508.08FU_AB634264_refs	Thelephorales	0	0.01172754
SH1184486.08FU_UDB0244450_reps	Geminibasidiales	0	0.04084656
SH2601717.08FU_UDB0237225_reps	Pleosporales	1	0.00495125
SH1166278.08FU_MK841471_reps	Aspergillaceae	1	9.76e-05
SH1152427.08FU_MG029166_reps	Helotiales	1	0.01601629
SH1215154.08FU_LN810821_reps	Acarosporales	2	0.02562728
SH1160442.08FU_HQ650658_reps	Lecideales	3	0.30930186
SH1168071.08FU_KU945909_reps	Aspergillaceae	14	2.43e-05

Table 9: The difference in distance and the difference in the number of nodes between two leaves for the original tree and the `pplacer` tree for the first test. The table is sorted on the difference in number of nodes. The asterisk (*) highlights the queries that are tested again in test three.

Query id	Subtree	Difference in nodes	Difference in distance
SH1156865.08FU_AF486126_refs	Helotiales	0	0.00474857
SH1172583.08FU_KR912053_reps	Teloschistales	0	0.00038766
SH1228780.08FU_KY940500_reps	Magnaporthales	0	0.00719424
SH1183141.08FU_GU903288_reps	Magnaporthales	0	0.00516072
SH1142770.08FU_KT949894_reps	Xylariales	0	0.02904741
+SH1191717.08FU_MF488989_refs	Xylariales	0	0.00012276
SH1235707.08FU_KY051892_reps	Diaporthales	0	0.00596839
SH1162773.08FU_KT310978_reps	Aspergillaceae	0	0.00575541
SH1166814.08FU_EF669705_reps	Aspergillaceae	0	0.00145389
SH1183613.08FU_KJ707198_reps	Saccharomycetales	0	0.00262731
SH1150688.08FU_KF800188_reps	Pleosporales	0	0.01730934
SH1172915.08FU_JQ906769_refs	Saccharomycetales	0	0.00738778
+SH2614834.08FU_UDB0680020_reps	Tremellales	0	0.00309087
SH1191708.08FU_MF496149_reps	Russulales	0	0.01007196
SH1184430.08FU_UDB0767606_reps	Russulales	0	0.00124808
SH1145476.08FU_UDB007576_reps	Boletales	0	0.01763931
+SH1190303.08FU_MH926037_reps	Polyporales	0	0.01140133
SH1238102.08FU_JN105716_reps	Polyporales	0	0.01329597
SH1140827.08FU_AY969759_reps	Thelephorales	0	0.0022452
SH1159486.08FU_KF476884_reps	Thelephorales	0	0.00029567
SH1238165.08FU_MH861181_refs	Pleosporales	0	0.03525241
SH1245769.08FU_MN007027_reps	Polyporales	2	0.14336684
SH1166968.08FU_KY978354_reps	Aspergillaceae	3	0.01955677
SH1159594.08FU_UDB0335978_reps	Thelephorales	5	0.03630957
SH1162466.08FU_HQ891869_reps	Aspergillaceae	82	0.00493143

Table 10: The difference in distance and the difference in the number of nodes between two leaves for the original tree and the `pplacer` tree for the second test. The table is sorted on the difference in number of nodes. The plus (+) highlights the queries that are tested again in test three.

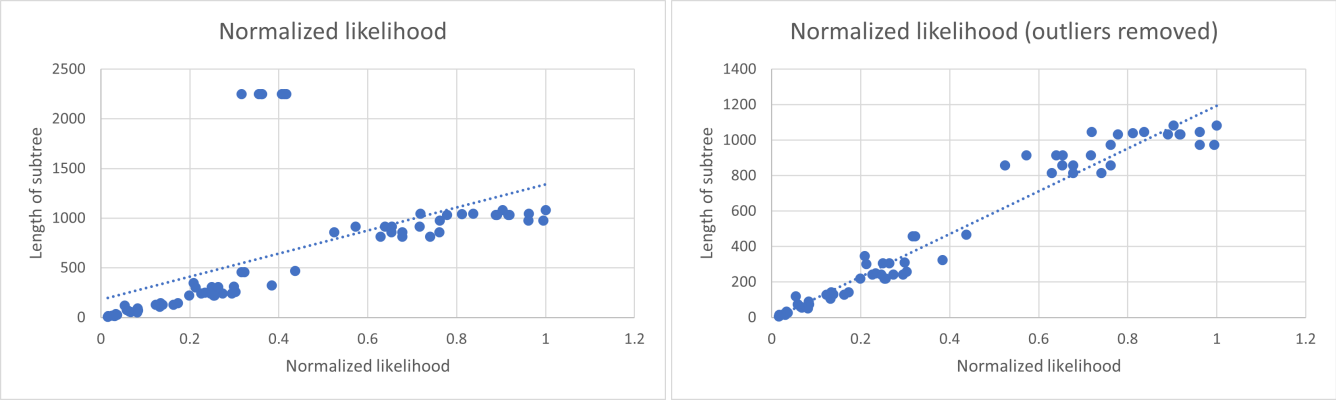
Query id	Subtree	Difference in nodes	Difference in distance
SH2591562.08FU_MK348933_reps	Glomerales	0	0.0033484
SH1174889.08FU_KR902678_reps	Teloschistales	0	0.00225512
*SH1152267.08FU_KR017144_reps	Trapeliales	0	0.00341981
SH1144564.08FU_HQ889706_reps	Boliniales	0	0.03117913
SH2604690.08FU_UDB090068_reps	Pleurotheciales	0	0.00166448
SH1153681.08FU_MT185508_reps	Chaetosphaeriales	0	0.00052149
+SH1191717.08FU_MF488989_refs	Xylariales	0	0.00515157
SH1173602.08FU_HM484543_refs	Hypocreales	0	0.00468159
SH1194552.08FU_UDB092932_reps	Sordariales	0	0.01459981
*SH1189121.08FU_MG920147_reps	Helotiales	0	0.00362057
SH1178035.08FU_EU836707_refs	Saccharomycetales	0	0.00648553
SH1155345.08FU_HM230882_reps	Helotiales	0	0.0115516
+SH1190303.08FU_MH926037_reps	Polyporales	0	0.01605295
SH1175403.08FU_KJ780631_refs	Hygrophoraceae	0	0.02279403
SH1180966.08FU_KT429785_refs	Hygrophoraceae	0	0.00411375
SH1235601.08FU_FJ943242_reps	Hygrophoraceae	0	0.03181313
SH1168537.08FU_LC056769_reps	Amanitaceae	0	0.00881615
SH2593743.08FU_MW077527_reps	Pluteaceae	0	0.02799038
+SH2614834.08FU_UDB0680020_reps	Tremellales	0	0.00309087
SH1155115.08FU_KC489498_refs	Umbelopsidales	0	0.00029826
SH1174831.08FU_LC368018_reps	Russulales	0	0.00015163
*SH1145193.08FU_EU222979_reps	Russulales	0	0.00897937
SH2602692.08FU_UDB0613842_reps	Auriculariales	0	0.00919327
SH1173571.08FU_UDB0764892_reps	Auriculariales	0	0.00313107
SH1162906.08FU_AF272574_refs	Aspergillaceae	1	8.99e-06
SH1169414.08FU_AJ458289_reps	Ostropales	1	0.06283601
SH1145889.08FU_UDB028521_reps	Helotiales	1	0.02181867
SH2608872.08FU_UDB0209524_reps	GS36	2	0.28178906
SH1235583.08FU_HQ874756_reps	Botryosphaeriales	2	0.00352666
SH1187931.08FU_MH858276_reps	Hypocreales	4	0.13445295

Table 11: The difference in distance and the difference in the number of nodes between two leaves for the original tree and the `pplacer` tree for the third test. The table is sorted on the difference in number of nodes. The asterisk (*) and the plus (+) highlight the sequences that are also tested in tests one and two respectively.

4.2.4 Interpretation of results

When there is heterogeneity in the subtrees found by BLAST, one would expect to have a lower maximum likelihood (ML) score than when there is no heterogeneity. This is not the case in this research. The minimal normalized likelihood score with all hits in one tree is 0.0377, while the maximum normalized likelihood score for a query with BLAST hits in multiple trees is 0.7611.

There is however a relation between the likelihood score and the size of the subtree. Figure 11a shows the relation between the normalized likelihood and the length of the subtree. The outliers pull the trendline a bit higher, which is why in figure 11b the outlier are removed. All outliers came from the same subtree, namely the *Aspergillaceae* subtree. These outliers are removed because the branch lengths in the tree are very small. The average branch length is 0.0195, but the median and the mode are both 0.0. Such small branch lengths can mean that the sequences are highly similar or that there is not enough differentiation yet. This makes it harder for `pplacer` to determine the placement, so the likelihood scores are lower. When the tree is smaller, the ML score is lower. A reason for this is that larger trees have more branches and thus increased complexity. A larger tree has more branch lengths and parameters associated with topology, which allows more flexibility in placing the query on the tree.

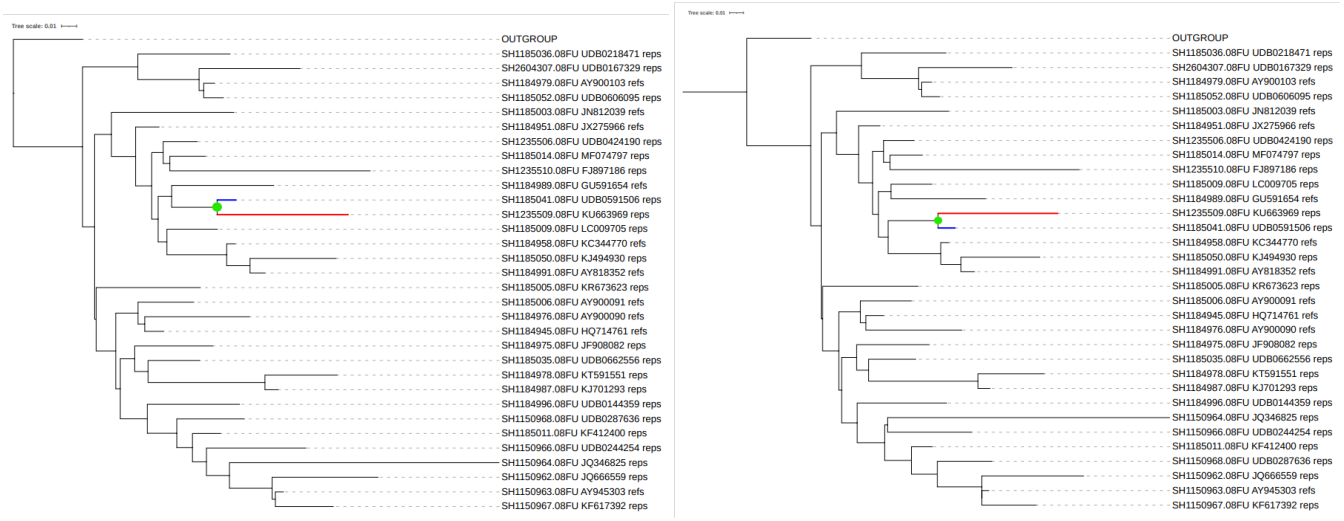


(a) Normalized likelihood plotted against the length of the subtree.

(b) Normalized likelihood plotted against the length of the subtree. Outliers are removed

Figure 11: Normalized likelihood plotted against the length of subtree.

If there is no difference in the number of nodes between two leaves in the different trees, it means that the query is placed in the same subtree as the neighbour. The exact location may differ, but the topology of the tree is not changed. The difference in distance can then be explained by the fact that the branch lengths of the branches in the tree have changed. There are no extra branches in between the two leaves. Figure 12 is an example of a subtree where the difference in number of nodes zero is. The red branch is the query and the blue branch is the neighbour. The green dot is the node between the two leaves. Figure 12a is the original tree and figure 12b is the tree where `pplacer` placed the query back. Only the rotation of the leaves is different in both trees.



(a) Difference in number of nodes in the original tree. The red branch is the query, the blue branch is its neighbour. The green dot is the node between the two branches.

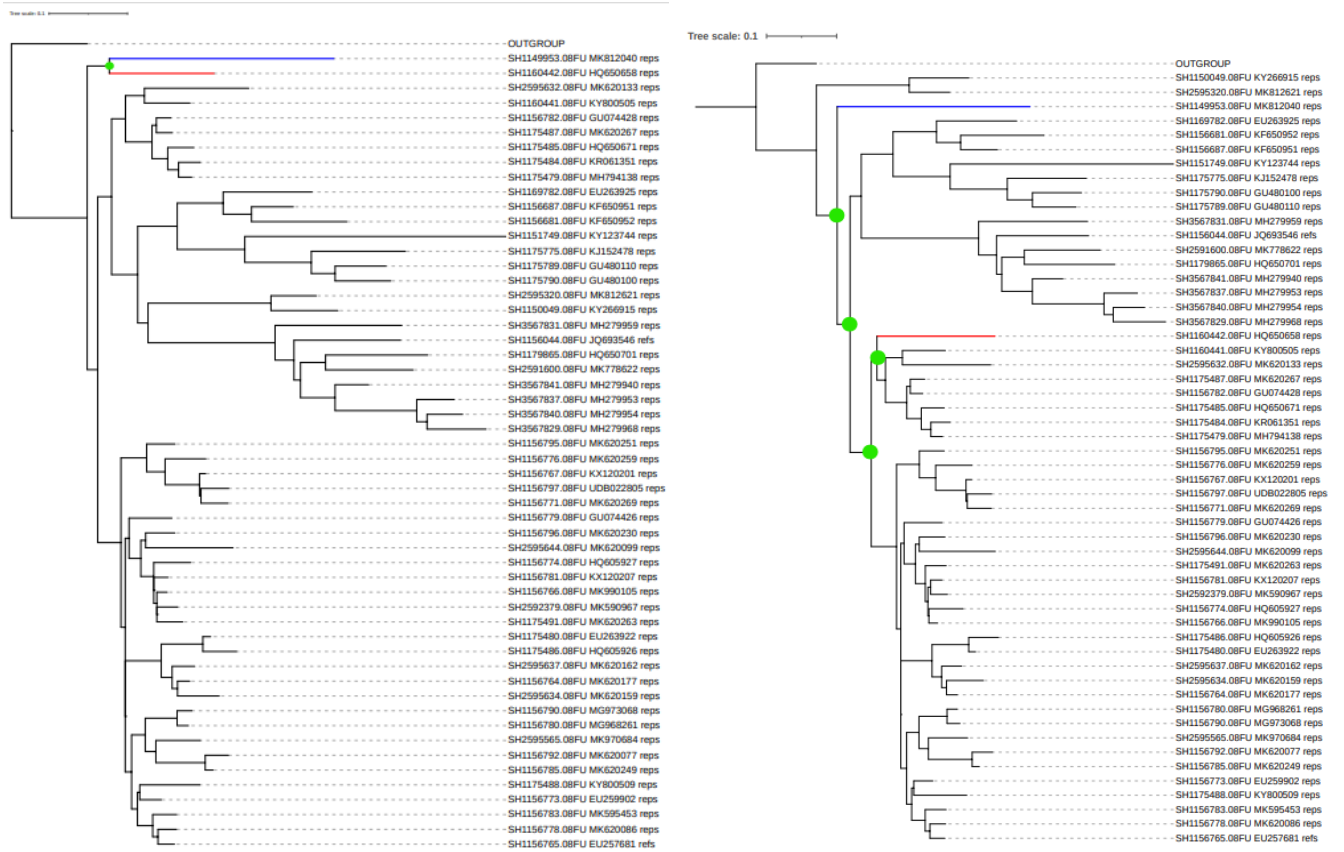
(b) Difference in number of nodes in the `pplacer` tree. The red branch is the query, the blue branch is its neighbour. The green dot is the node between the two branches.

Figure 12: Example where the difference in number of nodes between two leaves is zero. Result of subtree *Hymenogastraceae* in the first test. There is only a rotation of the leaves.

When the difference in the number of nodes is bigger than zero, it means that there are nodes added (or removed) between the two leaves. The difference in distance can then be explained by the fact that branches are added or removed between the leaves. Figure 13 is an example of a subtree where the difference in number of nodes is not zero, but three in this case. Figure 13a is the original tree and figure 13b is the tree with the query placed by `pplacer`. In the original tree, the two leaves are neighbours, but in the `pplacer` tree they are not anymore.

For each test BLAST and `pplacer` is run three times and the results of these tests are the same for all three runs. This indicates that BLAST and `pplacer` are deterministic algorithms. Because of time constraints, RAxML is not run three times. The results of these tests can be found in the zip file (see appendix B).

Three queries from test one and three from test two are also tested again in test three. The results in test three are not always the same as in test one or two. For example for query SH1190303.08FU_MH926037_reps in subtree *Polyporales*, the difference in distance is 0.01140133 in test two and 0.01605295 in test three. An explanation for this is that in test two there were three queries placed in this subtree, while in test three this query was the only query placed in this subtree. Since in the first case three queries are removed from the original subtree and in the second case only one, RAxML made two different trees, resulting in different branch lengths in the tree.



(a) Difference in number of nodes in the original tree. The red branch is the query, the blue branch is its neighbour. The green dot is the node between the two branches.

(b) Difference in number of nodes in the `pplacer` tree. The red branch is the query, the blue branch is its neighbour. The green dots are the nodes between the two branches.

Figure 13: Example where the difference in number of nodes between two leaves is not zero, but three in this case. Result of subtree *Lecideales* in the first test.

4.3 Statistical validation

As stated in section 1.4, we expect that if we remove a leaf and place it back using the proposed method, the difference in nodes between two neighbour leaves is zero. And if the method does not work, the difference in number of nodes is not zero. To validate this, we use a two-tailed, one-sample t-test. The data that is used, contains the 73 unique queries which are placed using `pplacer` and their corresponding difference in number of nodes. The computed mean of difference in number of nodes is 1.712 and the median is 0 for the data. The standard deviation is 9.709 and the maximum value is 82. The significance level is 0.05. This study aims to examine the null hypothesis and its corresponding alternative hypothesis concerning the average difference in the number of nodes between the query sequence and its neighbour. The null hypothesis, H_0, posits that the mean difference in node count is not zero ($\mu \neq 0$), while the alternative hypothesis, H_1, suggests that the mean difference is indeed zero ($\mu = 0$).

To assess these hypotheses, a t-test is employed using the following formula:

$t = \frac{\bar{y}-\mu_0}{se}$
Here, $\bar{y}$ represents the sample mean, μ_0 denotes the hypothesized population mean, and se is the standard error defined as $se = \frac{s}{\sqrt{n}}$, with s representing the standard deviation and n representing the sample size. The degrees of freedom (df) for the t-test are calculated as $n - 1$.
By setting the hypothesized population mean μ_0 to 0, the calculated t-value is 1.507. The corresponding degrees of freedom (df) are determined to be 72. The resulting p-value associated with this t-value is found to be 0.136237.

Upon evaluation, it is observed that the p-value exceeds the significance threshold of $p < 0.05$. Consequently, the obtained p-value of 0.136237 does not achieve statistical significance at the $p < 0.05$ level. This outcome suggests that there is insufficient evidence to reject the null hypothesis (H_0) in favor of the alternative hypothesis (H_1), implying that the mean difference in the number of nodes is not zero.

In a subsequent analysis, a two-tailed, one-sample t-test is conducted on the dataset. This analysis involves the exclusion of the data points from the *Aspergillaceae* subtree. This decision is made due to the intrinsic challenges posed by small branch lengths for the `pplacer` algorithm, resulting in a notably elevated difference in node counts within this particular subtree. Specifically, a total of seven queries from the *Aspergillaceae* subtree are removed from the data set.
With this refined dataset, the analysis now consists of 66 queries. The computed mean for this subset of data is 0.364, while the standard deviation is calculated to be 0.971. The maximum observed value within this dataset is 5. Upon applying the t-test formula to this data, a resulting t-value of 3.134 is obtained, corresponding to a degrees of freedom (df) value of 65. The associated p-value derived from this analysis is calculated to be 0.002591.

It is important to note that this outcome holds statistical significance at the threshold of $p < 0.05$. This indicates that the result of this particular analysis provides strong evidence to reject the null hypothesis, thereby suggesting that the average difference in number of nodes is zero.

4.4 Runtime

Table 12 presents the runtime measurements for different components of the method, as well as the total runtime of the method itself. Test one and test two consist of 25 queries each, while test three includes 31 queries. In test one, RAxML is executed for 22 subtrees, with the first 11 subtrees executed using 2 cores and the remaining 11 subtrees using 8 cores. Test two involves 13 subtrees, and test three involves 24 subtrees. The `pplacer` and guppy tools are both applied to 24 queries in test one, 25 queries in test two, and 30 queries in test three.

	Test 1	Test 2	Test 3
Make blast database	1.342s	1.128s	1.118s
BLAST	7.813s	8.010s	10.085s
Determine subtree	23.662s	21.773s	22.513s
RAxML (using 2 cores)	6h10m59.9496s	x	x
RAxML (using 8 cores)	6h23m26.6316s	6h35m46.1845s	8h16m44.8465s
Make reference package	16.678s	11.839s	18.134s
`pplacer`	1m13.710s	1m7.689s	1m12.297s
guppy	1.596s	1.566s	1.437s
Total	12h36m31.3822s	6h36m38.1895s	8h18m50.4305s
Total without database, RAxML and reference package	**1m46.781s**	**1m39.038s**	**1m46.332s**

Table 12: The runtime of the different steps of the method for the different tests. BLAST is run with 25 queries for tests one and two and 31 for test three. RAxML is run for 22 subtrees for test one, 13 subtrees for test two, and 24 subtrees for test three. For the first test, RAxML is run with two cores for the first 11 subtrees and with eight cores for the other 11 subtrees.

In Table 12, the orange rows indicate the tasks that need to be executed only once whenever the reference tree is updated. These tasks are independent of the method used. On the other hand, the light blue row represents the total runtime of the method, excluding the time required for creating the database, running RAxML, and generating the reference package.

4.5 Visualization

A visualization of the data can help to understand the data, and also give more insight into the data. For this research, two different types of visualization are used. The first is a visualization of the results of the calculations on the data, explained in section 4.5.1. The second visualization is the placements of the queries, explained in section 4.5.2. For both visualizations, the reference tree is used. The reference tree has 23.000+ leaves, so it is hard to visualize the whole tree on a screen. All the labels will overlap and the different clades are not visible. That is why we decided to collapse a big part of the tree. As explained in section 2.1.1, the reference tree is built using different chunks. Each chunk is collapsed into one single node, which gives a reference tree with only 229 leaves. This is easier to visualize on a screen. For calculation the names of the chunks were numbers. For visualization, the numbers were changed to names. The lookup table for this can be found in Appendix A table 13.

4.5.1 Visualization of result

Several values are returned by `pplacer` or calculated in section 4. Visualization of these values gives a better understanding of the results. Four different values are added as annotations to the tree. The visualization is made using ITOL [4]. The annotation files can be found in the zip file, see appendix B. The first value denotes if BLAST found the correct subtree. The green checkmark

[4] https://itol.embl.de/

denotes that all ten hits were in the same, correct subtree. The orange star means that not all hits were in the same subtree, but the majority of hits were in the correct subtree. The red square means that the majority of the hits were not in the correct subtree. The second value is the posterior probability given by `pplacer`. The value is shown as a colour, using a heatmap, with red as the lowest value and green as the highest. The third value is the difference in nodes between two leaves in the original tree and the `pplacer` tree. This is shown as a number. And the last value is the difference in distance between two leaves in the original tree and the `pplacer` tree. This value is also shown as a colour, using a heatmap, with green for the lowest distance and red for the highest distance. Different queries in the same chunk are shown behind each other.
Figure 14 is the result of test one. The corresponding values can be found in table 3, 6 and 9. The colours of the leaves correspond with the colours in table 9. Figure 15 is the result of test two. The corresponding values can be found in table 4, 7 and 10. The colours of the leaves correspond with the colours in table 10. Figure 16 is the result of test 1. The corresponding values can be found in table 5, 8 and 11. The colours of the leaves correspond with the colours in table 11. The figures are also included in the zip file.

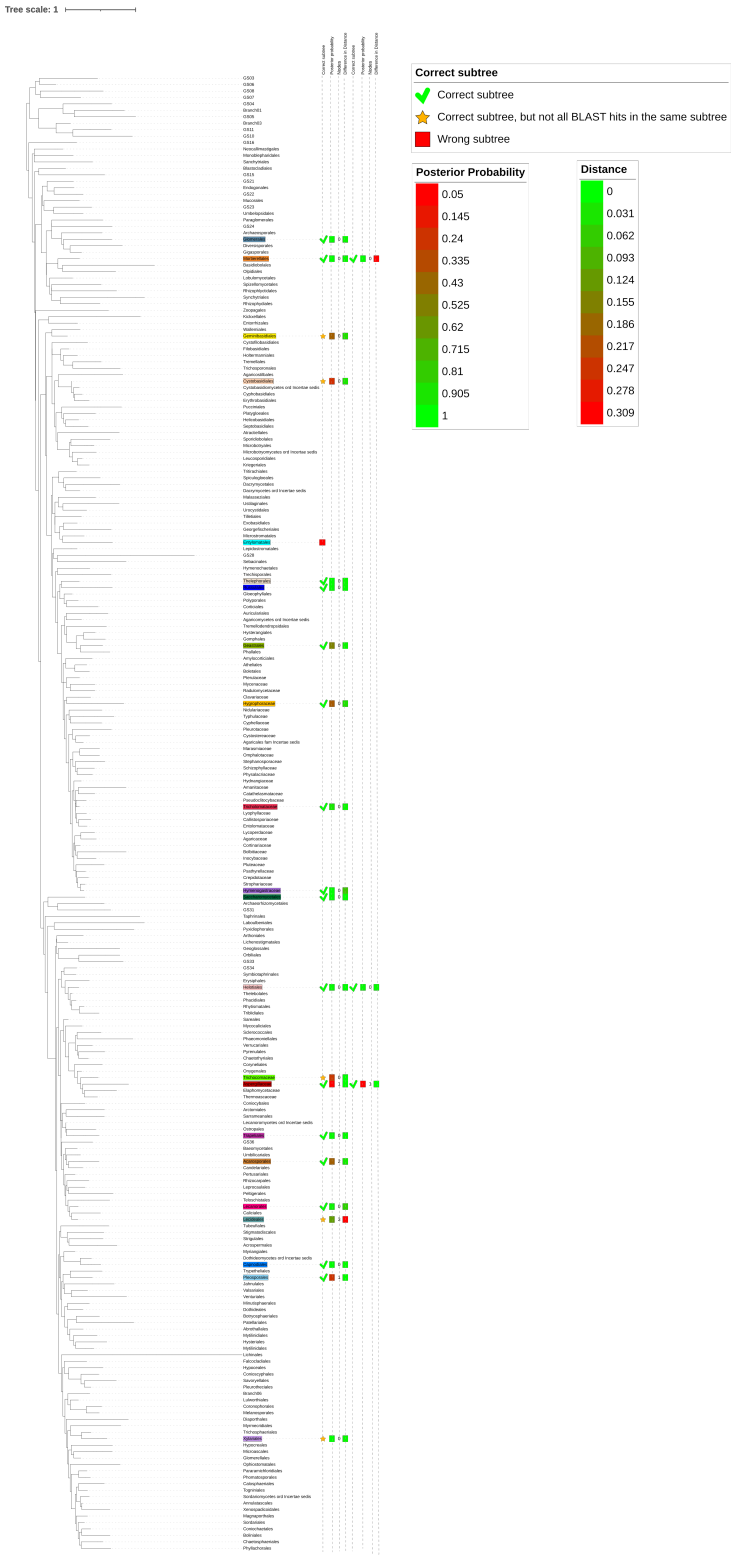


Figure 14: Visualization of the results for the first test. The first column denotes the results for the subtree. The second column is a heatmap for the posterior probability. The third column is the difference in number of nodes. The last column is a heatmap for the difference in distance. Multiple queries that are placed in the same subtree are placed behind each other. The colours of the leaves correspond with the rows in table 9.

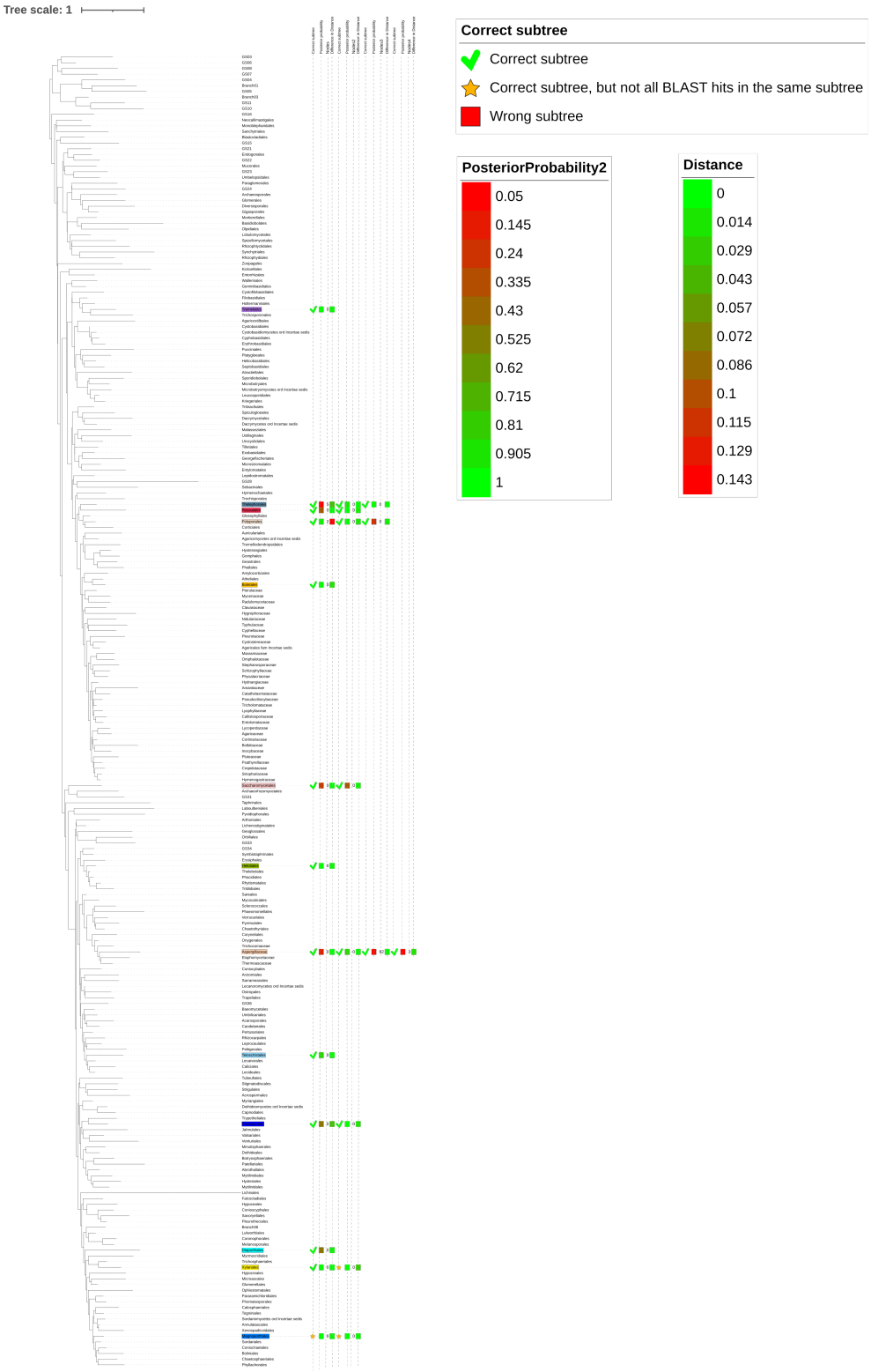


Figure 15: Visualization of the results for the second test. The first column denotes the results for the subtree. The second column is a heatmap for the posterior probability. The third column is the difference in number of nodes. The last column is a heatmap for the difference in distance. Multiple queries that are placed in the same subtree are placed behind each other. The colours of the leaves correspond with the rows in table 10

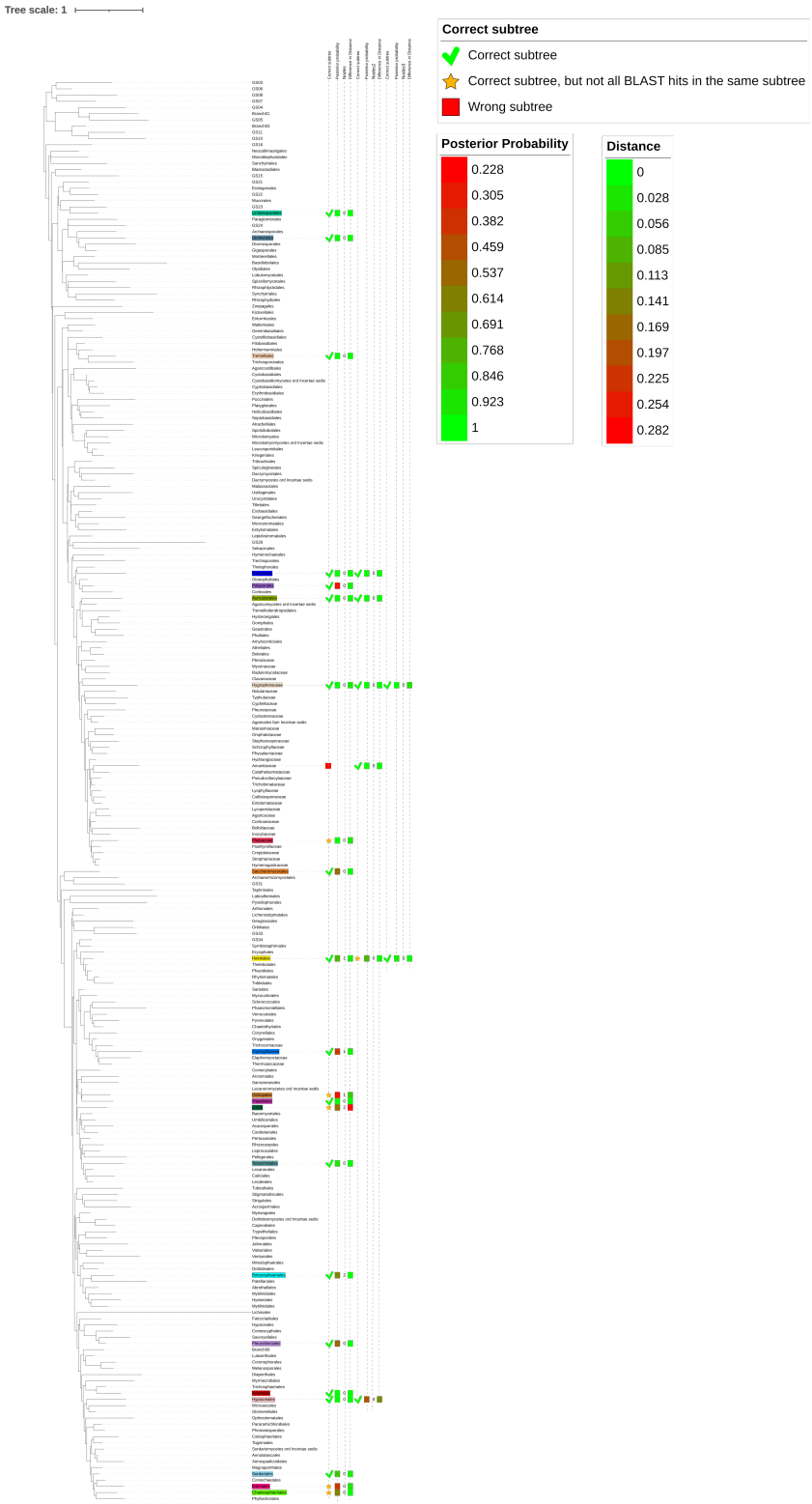


Figure 16: Visualization of the results for the third test. The first column denotes the results for the subtree. The second column is a heatmap for the posterior probability. The third column is the difference in number of nodes. The last column is a heatmap for the difference in distance. Multiple queries that are placed in the same subtree are placed behind each other. The colours of the leaves correspond with the rows in table 11

4.5.2 Visualization for biodiversity

To be able to gain insight into the biodiversity, a visualization of the placement(s) is important. The location of the placements in the tree can give information about how diverse a sample is. Figure 17 is an example, red and blue are two different samples, each with three reads. The dots next to the tree denote the placement of the queries in the tree. Since blue occurs in more different branches, the blue sample has higher biodiversity.

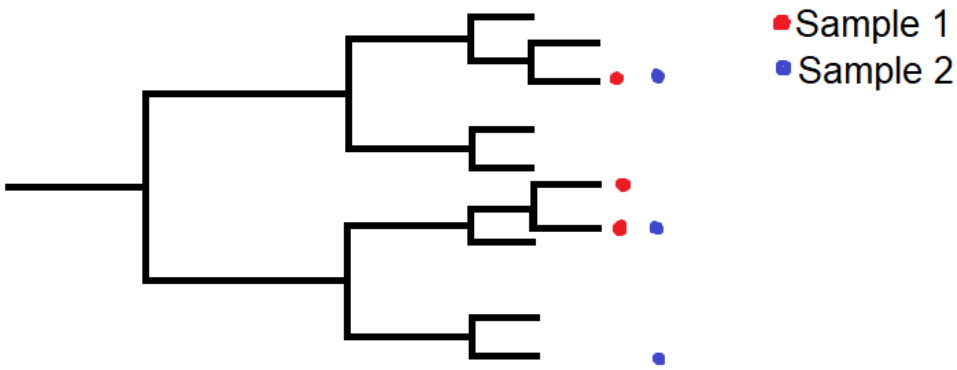


Figure 17: An example of how the difference in biodiversity can be determined using a phylogenetic tree. The sample that is present in more different clades is considered to be more diverse.

A problem you encounter when visualizing a big tree is the fact that you want to see detail (the actual placement) without losing too much context. When you zoom in on the leaves of a small part of the tree, you don't get an overview of the rest of the tree. That is why we made a static visualization of a smaller tree and we propose a method for an interactive visualization. Zooming in, into the static visualization will end up in pixels, while in the dynamic visualization, one can keep zooming in. The smaller tree that is used, is made by collapsing the chunks into one single node.

There are two versions of the static visualization. The first version, figure 18, shows how many queries there are in one chunk. The magenta colour denotes one query in the chunk, the yellow colour denotes two queries in the chunk and the blue colour denotes three or more queries in the chunk. This figure is generated using the data of the third test.

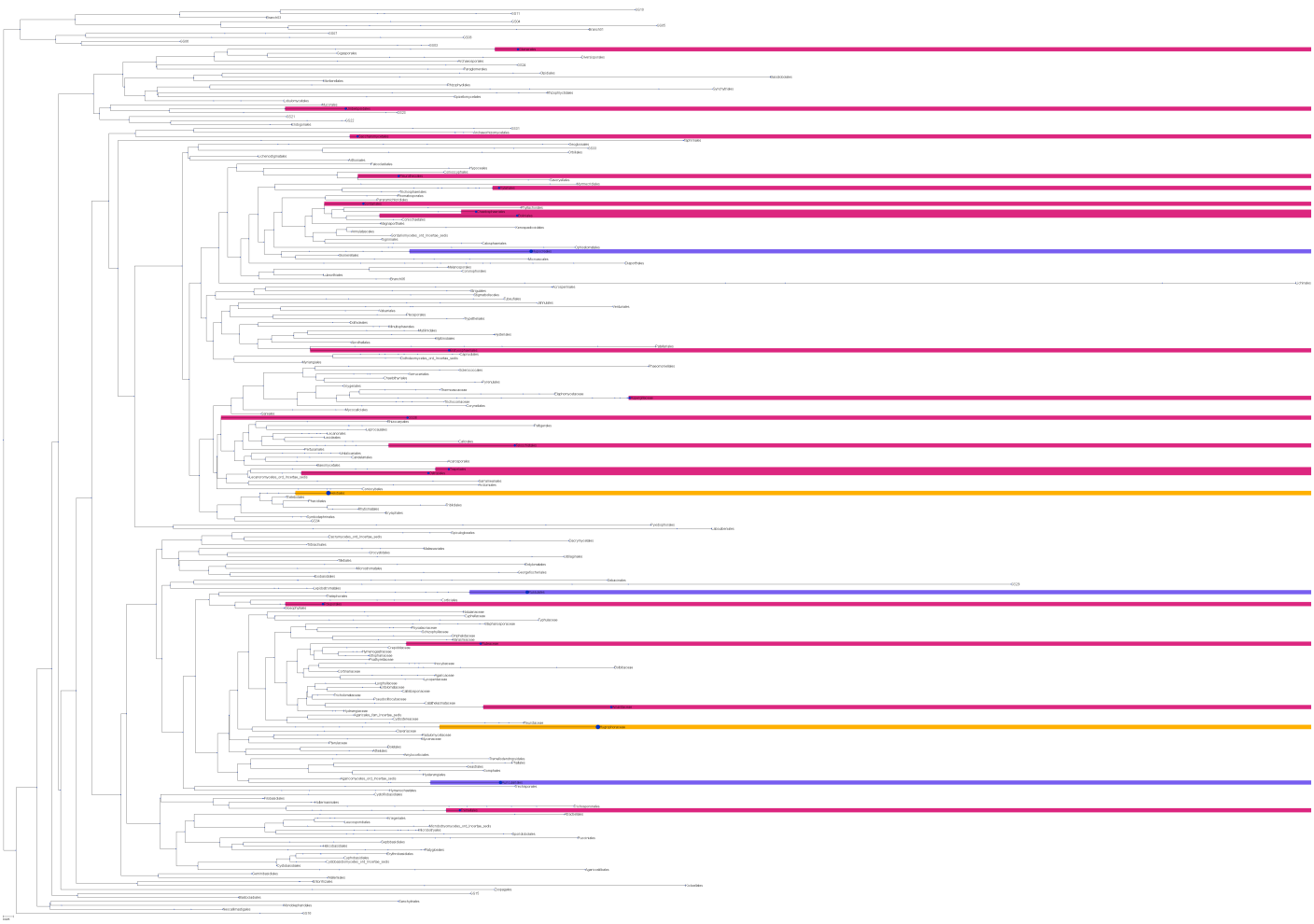


Figure 18: Visualization of the number of queries that are placed in a chunk for the third test. Magenta denotes one query, yellow denotes two queries and blue denotes three or more queries.

The second version shows the diversity between different samples. In figure 19 the three different tests are visualized. If a chunk contains a placement of a query from one of the tests, it gets a dot behind the leaf. The magenta dots are for the first test, the yellow dots are for the second test and the blue dots are for the third test. The size of the dots denotes how many queries of that sample there are in the chunk. The smallest dot means one query, the middle dot means two queries and the largest dot means three or more queries. Figure 20 shows a cutout of the larger tree in figure 19.

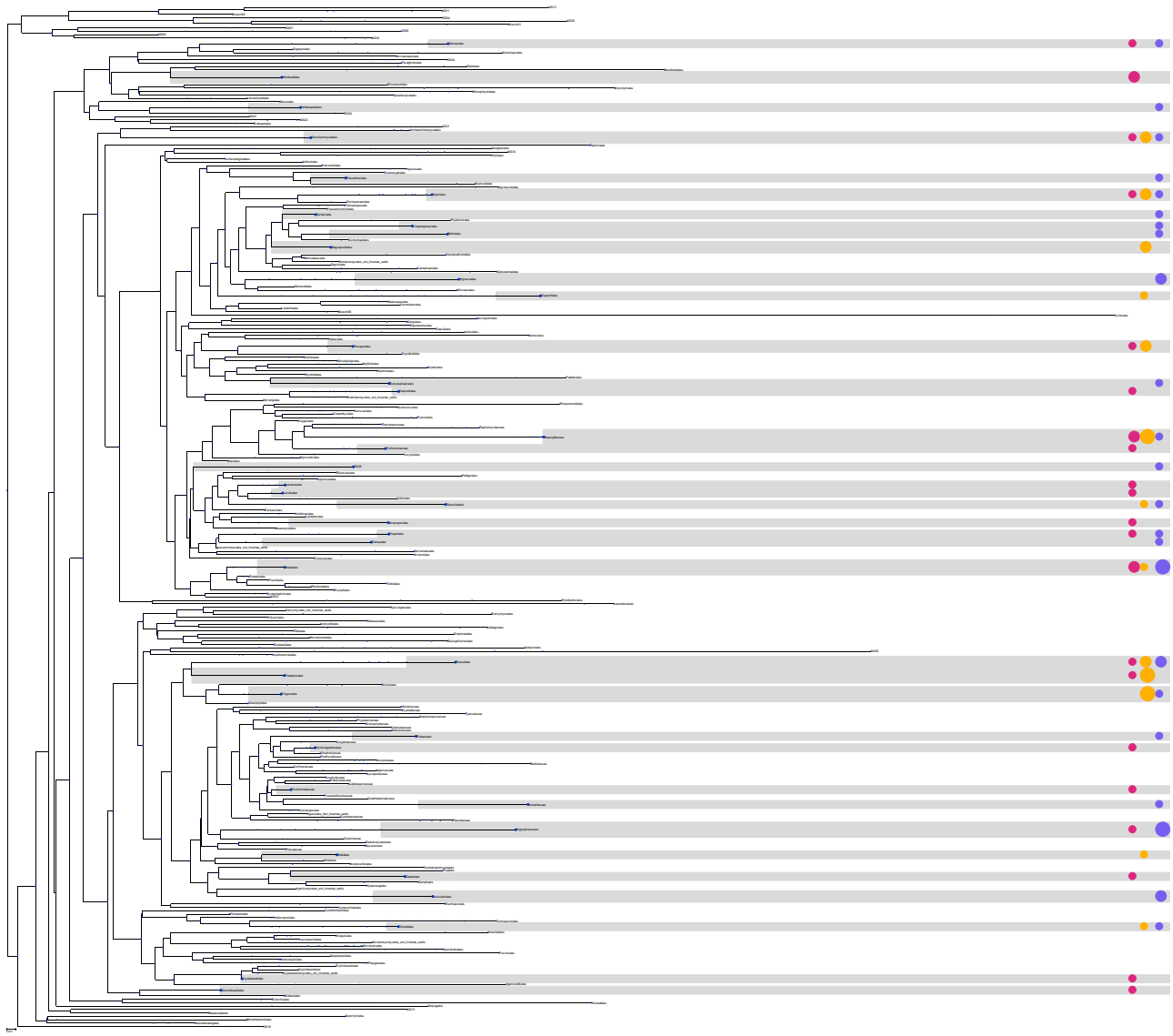


Figure 19: Visualization to compare the three different tests. The magenta dots are the first test, the yellow dots the second, and the blue dots the third. The size of the dots denotes how many queries there are in the chunk. The smallest dot denotes one query, the middle dot denotes two queries and the large dot denotes three or more queries.

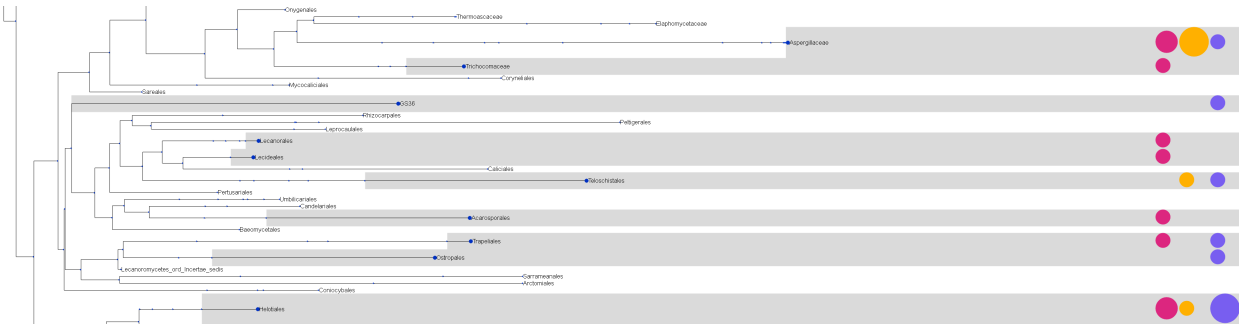


Figure 20: A cutout of the visualization to compare the three different tests. The magenta dots are the first test, the yellow dots the second, and the blue dots the third. The size of the dots denotes how many queries there are in the chunk. The smallest dot denotes one query, the middle dot denotes two queries and the large dot denotes three or more queries.

Another way to visualize the tree could be as a dynamic visualization. There are different ways to make a visualization dynamic. In this part, we propose an idea for a method for dynamic visualization. The branch on which the sequence(s) should be added will be coloured. When making a visualization dynamic, it is important to keep the goal in mind. In this case, the goal is to use it to determine biodiversity. Thus, it is essential to keep an overview of the whole tree, while zooming into a specific part. By using collapsible nodes, parts of the tree that are not of interest can be collapsed and parts that are of interest can be expanded. However, this particular reference tree has a lot of internal nodes. The collapsed tree in figure 19 has 632 internal nodes. If these nodes are all collapsible, one should click lots of nodes to get to the view it wants. To prevent this we came up with two solutions, that also can be combined.

The first solution is to not make all nodes collapsible. In the reference tree, the internal nodes are hypothetical sequences which means that it is estimated based on their children. Since the sequences are hypothetical, they might not be encountered in the real world and that is why internal leaves do not have a name in a phylogenetic tree. The lack of a name can be used to distinguish between the nodes that are and those that are not collapsible. However, the tree would use some preprocessing to be able to use this, since right now only the leaves have names. When making the chunks of the tree collapsible, the resulting tree with collapsed chunks has 229 leaves. To do this, the common ancestor of the chunk should get the name of the chunk, to make the node collapsible. A dot, with a different colour and a different size, at each collapsible node could be used to indicate the number of children of the node. When clicking on a collapsible node, it will expand all its children.

However, some chunks still have quite a lot of children, which makes it difficult to get a good view. That is why we came up with a second solution.

The second solution prunes branches that are not of interest and it uses also collapsible nodes. The first part is the same as solution one. It differs from when you click the collapsible node. Instead of expanding the whole tree, it will only expand a few leaves. For the biodiversity determination, only the exact placement and a few neighbours around this placement are important, the rest of the subtree is not of interest. Showing only a branch from the root to some of the leaves or nodes, may not provide enough information. By adding data to the branches with the normalized distance to

this node or leaf and the number of nodes between the node/leaf and the root extra information will be provided. This allows the visualization of a small part of the tree to show the placement, and keep the overview of the tree to see where in the tree you are.

A prototype of the first solution is implemented using `D3.js` and can be found here. The code for this visualization is based on this code. Figure 21 is an overview of the dynamic visualization. Figure 22 is a close-up of the visualization, where three subtrees are expanded.

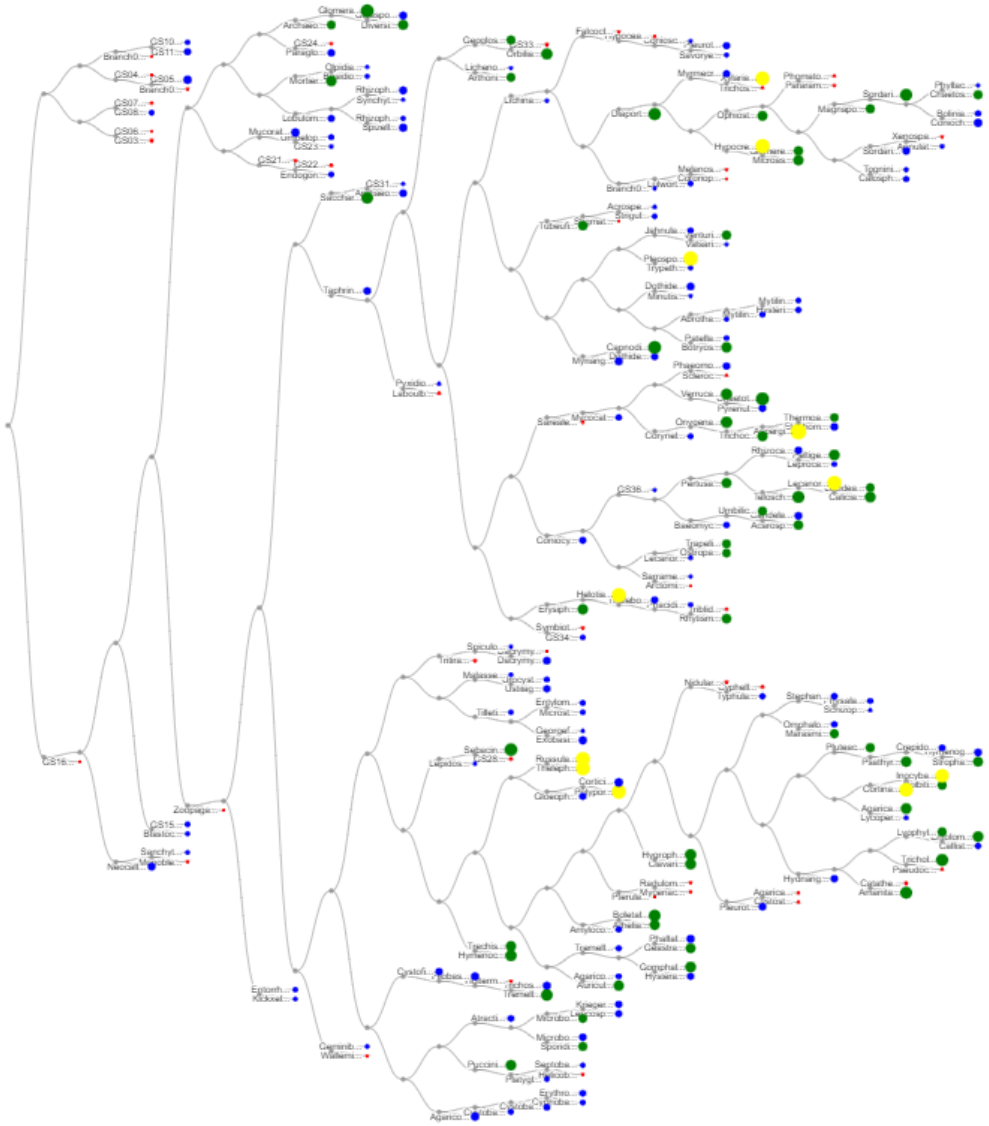


Figure 21: An overview of the prototype. The colour and the size of the dots correspond with the number of children. The coloured dots are expandable. The names of the subtrees are shortened and are shown completely when the mouse is hovered over the name.

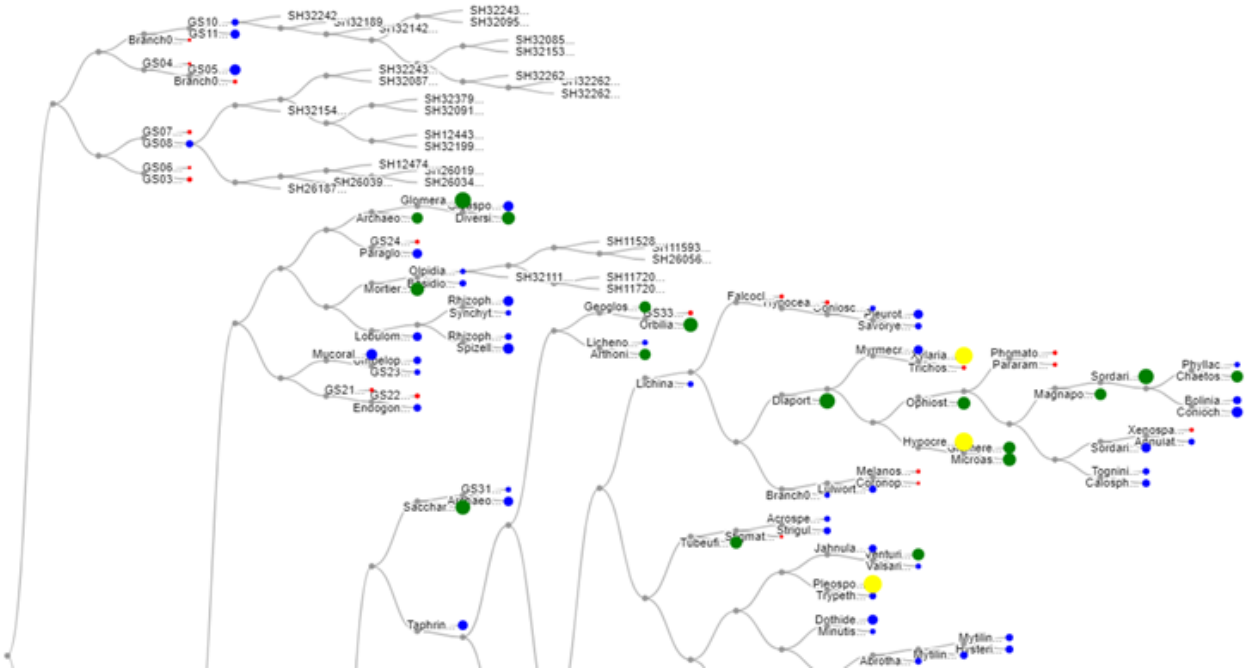


Figure 22: Close-up of the prototype. The subtrees GS10, GS08, and Olpidiales are expanded. The grey dots are the internal nodes that are expanded all at once when expanding a subtree.

The size and the colours of the nodes correspond with the number of children. The bigger the dot, the more children it has. Currently, the visualization lacks placement of the different queries, so updates of this code are required. Another point of work is the size of the tree. If the branches are too long, they may extend beyond the screen, making the visualization incomplete. On the other hand, if the branches are too short, the nodes may overlap, hindering clarity. Experimenting with different branch lengths or implementing an algorithm to dynamically adjust the branch lengths based on the tree structure and available screen space will improve the visualization. It is important to keep in mind that the branch lengths in this interactive tree are all the same, so they do not denote the evolution of the sequences.

5 Conclusions and Further Research

The primary objective of this research was to develop a method capable of conducting phylogenetic placement using ITS data. To achieve this, existing phylogenetic placement algorithms were thoroughly examined. Through this analysis, an algorithm was identified and integrated into the newly created method, alongside the utilization of BLAST. The method underwent testing and validation using three distinct test sets. All code written for this research can be found on GitHub. This section presents the findings that address the research questions, offers a discussion of the results obtained, and outlines potential topics for future research and improvements.

It is important to note that the method has only been tested using a subset of the reference tree data. No testing has been conducted on new, unseen data. Therefore, the conclusions and discussions drawn from the results are solely based on the tests where the original placement locations were known.

5.1 Conclusion

Following the results obtained from this research, the research questions stated at the beginning will be answered below. First, the subquestion about the different algorithms will be answered. After that, some conclusions about the results will be given. Then, the second subquestion about the visualization will be answered and finally, the main research question will be answered.

RQ1.1: Which suitable algorithms already exist for phylogenetic placement?

There are different types of algorithms for phylogenetic placement. The two biggest types are algorithms based on maximum likelihood and algorithms based on distances. Since the data that is used, is short, has gaps, and has a high variability, algorithms that use maximum likelihood are preferred. The big disadvantage of such algorithms is that they are computationally expensive, especially with large trees. For the method to be compatible with MonetDB, it is important that it is written in Python. That is why `pplacer` is chosen as placement algorithm. Only, the problem with `pplacer` is, that it fails on trees with more than 5000 leaves. Since the tree that is used has more than 23.000 leaves, BLAST is used to determine the subtree and reduce the size of the tree.

Using BLAST for the subtree determination works reasonably well. 75 unique queries are tested and for 73 of them, the majority of the BLAST hits came from the correct subtree, see also table 2. Of these 73 queries, 59 queries had all ten BLAST hits in the same subtree.
Using `pplacer` for the placement of the query works reasonably well. For 16 of the 73 unique queries, the difference in number of nodes is bigger than zero. This means that for those 16 queries, `pplacer` did not place them back in the same location. The difference in distance for those queries is thus due to the fact that branches have been added or removed. The method is statistically validated using a one-sample t-test with a significance level of 0.05. The null hypothesis being tested was that the average difference in the number of nodes is not zero. Applying this test to a dataset comprising 73 queries, a p-value of 0.136 was obtained. As this value exceeds the significance level of 0.05, the results are not statistically significant. However, when the queries from the *Aspergillaceae* subtree are excluded, 66 queries remain for analysis. Applying the one-sample t-test to this reduced dataset

yielded a p-value of 0.0026. This p-value falls below the threshold of 0.05, indicating statistical significance. Consequently, the null hypothesis can be rejected. These findings suggest that the observed results are unlikely to have occurred by chance alone, under the assumption of the null hypothesis.

When looking at the runtime of the method, RAxML takes up most of the runtime time. However, this only needs to be executed once every time the reference tree is updated. Considering the entire process, the method is executed in approximately 1 minute and 45 seconds.

To be able to understand and gain insight into the data, visualization is needed leading to the following question:

RQ1.2: What is an efficient visualization for the outcome?

Since the reference tree is large, an efficient visualization is needed. For this research, two visualizations are made. A static and a dynamic one. The static visualization can visualize the placement of different reads in a sample or compare different samples. However, the details of the placement are lost in the static visualization. A method for dynamic visualization is proposed to keep an overview, but also be able to zoom in on the details.

Not only does the visualization of the placements on the tree give insight into biodiversity. The different colours given to each subtree also give a quick insight into biodiversity. By representing the results in tables, where each subtree is associated with its respective colour, the visual representation becomes even more informative. The variations and distinctions in colours serve as indicators of differences in biodiversity across different samples.
Based on the retrieved results and the subquestions answered above, the main research question can be answered.

RQ1: How can the phylogenetic tree be a measure to determine biodiversity?

Phylogenetic placement plays a crucial role in using the phylogenetic tree as a measure of biodiversity. Comparing a sample with the reference tree gives the placement of the different queries within the sample onto the tree. This provides valuable insight into the variety of species in the sample, thereby offering a means to assess the biodiversity of a sample. The biodiversity of a sample could be quantitatively expressed using different measures. To express the biodiversity within a sample, Faith's phylogenetic biodiversity measure could be used. To express the biodiversity between samples UniFrac can be used.
While the method proposed in this research demonstrates reasonable performance, it should be noted that further refinements are still required to achieve perfection. It is however a valuable tool that can be used to enrich MDDB because the method is compatible with MonetDB. Even the integration of interactive visualization made in D3.js enhances the functionality of MDDB.

In summary, the method presented in this study contributes to the enrichment of MDDB by facilitating phylogenetic placement and leveraging compatibility with MonetDB. With the potential for integration of interactive visualizations, it holds promise as a valuable tool for studying and understanding biodiversity.

5.2 Discussion

This section presents several points of discussion that need to be considered when interpreting the results of this research.
In the current approach for determining the correct subtree, only the top ten hits from BLAST are considered. The subtree with the highest number of hits is chosen as the primary subtree. However, when two subtrees have an equal number of hits, the first subtree is chosen. As observed in the results in section 4.1.2, this method does not yield perfect results. To improve the accuracy of subtree selection, alternative BLAST metrics can be incorporated. Metrics such as percentage identity, ϵ-value, or percentage coverage can offer additional information to improve the accuracy of selecting the correct subtree. Selecting the correct subtree influences the outcome of the whole method.

Another point of discussion is the algorithm used for the phylogenetic placement. As discussed in section 3, there are different algorithms that can perform phylogenetic placement. However, there are also methods that use k-mers to identify species in a sample. This is a sensible approach, however when using ITS data there are a few elements that need to be taken into account. The DNA regions used in this research are short and highly variable, which can lead to k-mer patterns that overlap between different taxa. On the other hand, specific k-mer patterns might be present in the short sequences, leading to the identification of the species and also resolving lower taxonomic levels. Another element is the quality of the reference database, if the reference database is made using longer sequences, the shorter sequences might not match the reference sequence. However, if the reference database is made using the short variable sequences the identification might become more accurate.

Furthermore, when working with a pruned tree, the use of `pplacer` results in a parsing error in the tree file. To avoid this issue, RAxML is used to generate a new tree of the data with the pruned leaf removed. However, due to the removal of this leaf from the data, the newly generated tree may exhibit variations in its structure compared to the original subtree. Consequently, verifying the placement of `pplacer` becomes more challenging.

When verifying the work of `pplacer`, counting the nodes between the root and the leaf in both the original tree and the tree where `pplacer` places the query would be desirable. A count of zero would indicate that the query is properly placed within the correct subtree. However, in this case, such a method cannot be used since the tree structures can differ. Instead, the number of nodes between the query and a neighbouring node is used as a verification mechanism.

The runtime of the method is heavily dependent on the runtime of RAxML. As leaves are pruned from the subtrees, new trees must be generated, which involves computationally intensive tasks such as creating different trees and calculating the likelihood score for each tree. However, it is important to note that this step does not need to be performed every time the method is executed. It is only required to be executed when the reference tree is updated. In this case, only an information file of the tree needs to be generated since the subtrees are already present and unpruned. This optimization significantly reduces the runtime of RAxML. Additionally, updating the reference tree necessitates updating the BLAST database, as well as generating new information files that are needed to update the reference packages.

It is worth considering that the method has been tested with a maximum of 30 queries.

However, in practical scenarios, an unseen sample may contain more than 100 queries, which can potentially increase the runtime due to the larger dataset.

It is crucial to understand the functionality and principles of `pplacer`. For accurate biodiversity determination, precise placement of queries is important. The placement location is determined through a maximum likelihood calculation, with optimization of branch lengths. When `pplacer` identifies multiple potential locations for a query, a likelihood weight ratio (LWR) is associated with each placement. The location with the highest LWR is chosen as the final placement. However, in certain cases, there may be numerous possible placements with identical LWR values. In such scenarios, `pplacer` selects the first location from the list, which may not necessarily be the correct placement. Considering an alternative location from the list could yield better results.

Furthermore, it is worth noting that `pplacer` has the potential to even improve the placement of queries. During the construction of the original reference tree, decisions are made at higher levels, influencing the order of the tree's leaves. When a sequence is removed from the tree, subsequent decisions may lead to the construction of different trees by RAxML. For the 16 queries where the difference in the number of nodes is non-zero, the new placement might be superior to the original placement. However, further research is required to fully explore this possibility.

Lastly, while the static visualization is currently suitable for its intended purpose, the interactive visualization requires further enhancements. The design of the interactive visualization aligns with Ben Shneiderman's information visualization mantra: *'overview first, zoom and filter, then details on demand'* [Shn96]. This mantra emphasizes the importance of providing an initial overview of the entire tree, allowing users to subsequently zoom in and focus on specific areas of interest to access detailed information.

Although a framework is already in place to support this approach, improvements are necessary to enhance its usability and functionality. Upgrading the framework will facilitate a more seamless and intuitive user experience, enabling users to effectively navigate and explore the details of the placement in the visualization. By upgrading the framework, the interactive visualization can better fulfill its purpose as a valuable tool for phylogenetic placement exploration and analysis.

5.3 Future work

There are several topics that can be added or changed in the proposed method, which may be interesting for future work on this topic. Firstly, right now BLAST is used as the first step to identify the subtree. There exists a different algorithm, called `Protax-fungi` that can be used to identify fungal ITS sequences [ASN+18]. This algorithm can be used to replace BLAST and maybe improve the method.
Another topic that can be reconsidered, is the length of the sequences. In MDDB the sequences are curated at 250 nucleotides. With longer sequences, the BLAST search might be more accurate, which can result in better placement. However, when using longer sequences for the queries, the reference tree should also contain longer aligned sequences. Otherwise, the identification might get worse.
Furthermore, the method is not yet tested with new data. The method is only validated using known data. When adding new samples, the biodiversity of these samples can be computed using different measures. The phylogenetic biodiversity (PD) can be expressed quantitatively using Faith's

PD to express PD within a sample or UniFrac to express PD between different samples. Such a number can be more informative when a visualization is added. The visualizations shown in this research are already informative but could use some upgrades. When not only the tips are colour coded, but also the interior branches leading to the tips have colours, this could subtend which paths are common and how the paths are shared among samples. This will allow for comparing different trees with each other and clarify the quantitative expressions.
Lastly, for this method, `pplacer` is used as placement algorithm. `pplacer` is chosen, because it can be incorporated into the Python interface of MDDB and become a standard tool of MDDB. New samples from the ARISE project for longitudinal and local research, can then be analyzed using this tool. As described in section 3, there are several algorithms that can perform phylogenetic placement. It might be interesting to see how the other algorithms perform in this approach.

References

[ANL+10] Kessy Abarenkov, R Henrik Nilsson, Karl-Henrik Larsson, Ian J Alexander, Ursula Eberhardt, Susanne Erland, Klaus Høiland, Rasmus Kjøller, Ellen Larsson, Taina Pennanen, et al. The unite database for molecular identification of fungi–recent updates and future perspectives. *The New Phytologist*, 186(2):281–285, 2010.

[ASN+18] Kessy Abarenkov, Panu Somervuo, R Henrik Nilsson, Paul M Kirk, Tea Huotari, Nerea Abrego, and Otso Ovaskainen. Protax-fungi: A web-based tool for probabilistic taxonomic placement of fungal internal transcribed spacer sequences. *New Phytologist*, 220(2):517–525, 2018.

[Bau08] D Baum. Reading a phylogenetic tree: The meaning of monophyletic. *Nature Education*, 2008.

[Bio] FHCRC Computational Biology. taxtastic.

[BKC+19] Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. Epa-ng: massively parallel evolutionary placement of genetic sequences. *Systematic biology*, 68(2):365–369, 2019.

[BL01] Fiona SL Brinkman and Detlef D Leipe. Phylogenetic analysis. *Bioinformatics: a practical guide to the analysis of genes and proteins*, 2:323–324, 2001.

[BM21] Matthias Blanke and Burkhard Morgenstern. App-spam: phylogenetic placement of short reads without sequence alignment. *Bioinformatics Advances*, 1(1):vbab027, 2021.

[BSM20] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. Apples: scalable distance-based phylogenetic placement with or without alignments. *Systematic Biology*, 69(3):566–578, 2020.

[CCA+09] Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L Madden. Blast+: architecture and applications. *BMC bioinformatics*, 10:1–9, 2009.

[CR22] Casper Carton and Luuk Romeijn. Building a phylogeny for the fungal kingdom with its data, 2022.

[CSDB22a] Lucas Czech, Alexandros Stamatakis, Micah Dunthorn, and Pierre Barbera. Metagenomic analysis using phylogenetic placement—a review of the first decade. *Frontiers in Bioinformatics*, 2, 2022.

[CSDB22b] Lucas Czech, Alexandros Stamatakis, Micah Dunthorn, and Pierre Barbera. Metagenomic analysis using phylogenetic placement—a review of the first decade. *Frontiers in Bioinformatics*, 2:44, 2022.

[Fai92] Daniel P Faith. Conservation evaluation and phylogenetic diversity. *Biological conservation*, 61(1):1–10, 1992.

[GM22] Moumita Ghosh and Kartick Chandra Mondal. Computational biodiversity. In *Proceedings of International Conference on Advanced Computing Applications: ICACA 2021*, pages 739–750. Springer, 2022.

[GZR22] Melissa Gray, Zhengqiao Zhao, and Gail L Rosen. How scalable are clade-specific marker k-mer based hash methods for metagenomic taxonomic classification? *Frontiers in Signal Processing*, 2:842513, 2022.

[JLM+16] Jeffrey B Joy, Richard H Liang, Rosemary M McCloskey, T Nguyen, and Art FY Poon. Ancestral reconstruction. *PLoS computational biology*, 12(7):e1004763, 2016.

[KE08] W John Kress and David L Erickson. Dna barcodes: genes, genomics, and bioinformatics. *Proceedings of the National Academy of Sciences*, 105(8):2761–2762, 2008.

[KPW21] Elizabeth Koning, Malachi Phillips, and Tandy Warnow. ppiacerdc: a new scalable phylogenetic placement method. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 1–9, 2021.

[LCB+20] Mingxin Liu, Laurence J Clarke, Susan C Baker, Gregory J Jordan, and Christopher P Burridge. A practical guide to dna metabarcoding for entomological ecologists. *Ecological entomology*, 45(3):373–385, 2020.

[LLK+11] Catherine Lozupone, Manuel E Lladser, Dan Knights, Jesse Stombaugh, and Rob Knight. Unifrac: an effective distance metric for microbial community comparison. *The ISME journal*, 5(2):169–172, 2011.

[LSP19] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, 2019.

[MHK+20] Irene Martorelli, Leon S Helwerda, Jesse Kerkvliet, Sofia IF Gomes, Jorinde Nuytinck, Chivany RA van der Werff, Guus J Ramackers, Alexander P Gultyaev, Vincent SFT Merckx, and Fons J Verbeek. Fungal metabarcoding data integration framework for the mycodiversity database (mddb). *Journal of integrative bioinformatics*, 17(1), 2020.

[MKA10] Frederick A Matsen, Robin B Kodner, and E Armbrust. pplacer: linear time maximum-likelihood and bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC bioinformatics*, 11(1):1–16, 2010.

[MKA14] Frederick A Matsen, Robin B Kodner, and E Armbrust. pplacer documentation, guppy tog, 2014.

[NK12] Stratos Idreos Fabian Groffen Niels Nes and Stefan Manegold Sjoerd Mullender Martin Kersten. Monetdb: Two decades of research in column-oriented database architectures. *Data Engineering*, 40, 2012.

[OPJ+05] Heath E O'Brien, Jeri Lynn Parrent, Jason A Jackson, Jean-Marc Moncalvo, and Rytas Vilgalys. Fungal community analysis by large-scale sequencing of environmental samples. *Applied and environmental microbiology*, 71(9):5544–5550, 2005.

[PG06] Owen L Petchey and Kevin J Gaston. Functional diversity: back to basics and looking forward. *Ecology letters*, 9(6):741–758, 2006.

[PH20] Teresita M Porter and Mehrdad Hajibabaei. Putting coi metabarcoding in context: The utility of exact sequence variants (esvs) in biodiversity analysis. *Frontiers in Ecology and Evolution*, 8:248, 2020.

[SBPG+19] Shahab Sarmashghi, Kristine Bohmann, M Thomas P Gilbert, Vineet Bafna, and Siavash Mirarab. Skmer: assembly-free and alignment-free sample identification using genome skims. *Genome biology*, 20(1):1–20, 2019.

[SG94] EaBMG Stackebrandt and Brett M Goebel. Taxonomic note: a place for dna-dna reassociation and 16s rrna sequence analysis in the present species definition in bacteriology. *International journal of systematic and evolutionary microbiology*, 44(4):846–849, 1994.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pages 336–343. IEEE, 1996.

[SSH+12] Conrad L Schoch, Keith A Seifert, Sabine Huhndorf, Vincent Robert, John L Spouge, C André Levesque, Wen Chen, Fungal Barcoding Consortium, Fungal Barcoding Consortium Author List, Elena Bolchacova, et al. Nuclear ribosomal internal transcribed spacer (its) region as a universal dna barcode marker for fungi. *Proceedings of the national academy of Sciences*, 109(16):6241–6246, 2012.

[Sta14] Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 01 2014.

[Swe11] Nathan G Swenson. The role of evolutionary processes in producing biodiversity patterns, and the interrelationships between taxonomic, functional and phylogenetic biodiversity. *American Journal of Botany*, 98(3):472–480, 2011.

[VCMFM11] Mark Vellend, William K Cornwell, Karen Magnuson-Ford, and Arne Ø Mooers. Measuring phylogenetic biodiversity. *Biological diversity: frontiers in measurement and assessment*, pages 194–207, 2011.

[VESB23] Julia Van Etten, Timothy G Stephens, and Debashish Bhattacharya. A k-mer-based approach for phylogenetic classification of taxa in environmental genomic data. *Systematic Biology*, page syad037, 2023.

[WKW90] Carl R Woese, Otto Kandler, and Mark L Wheelis. Towards a natural system of organisms: proposal for the domains archaea, bacteria, and eucarya. *Proceedings of the National Academy of Sciences*, 87(12):4576–4579, 1990.

[ZJ12] Nadine Ziemert and Paul R Jensen. Phylogenetic approaches to natural product structure prediction. In *Methods in enzymology*, volume 517, pages 161–182. Elsevier, 2012.

A Lookup table

Tree number	Tree name	Tree number	Tree name	Tree number	Tree name
001	Glomerales	002	Diversisporales	003	Gigasporales
004	Archaeosporales	005	Paraglomerales	006	GS24
007	Thelephorales	008	Gomphales	009	Hygrophoraceae
010	Cortinariaceae	011	Inocybaceae	012	Amanitaceae
013	Lycoperdaceae	014	Agaricaceae	015	Typhulaceae
016	Clavariaceae	017	Hydnangiaceae	018	Tricholomataceae
019	Marasmiaceae	020	Mycenaceae	021	Psathyrellaceae
022	Strophariaceae	023	Callistosporiaceae	025	Omphalotaceae
026	Cyphellaceae	027	Entolomataceae	028	Pluteaceae
029	Lyophyllaceae	030	Pleurotaceae	031	Pterulaceae
032	Bolbitiaceae	033	Catathelasmataceae	034	Stephanosporaceae
035	Cystostereaceae	036	Hymenogastraceae	037	Schizophyllaceae
038	Agaricales_fam_ Incertae_sedis	039	Crepidotaceae	041	Physalacriaceae
042	Nidulariaceae	045	Radulomycetaceae	046	Pseudoclitocybaceae
048	Hymenochaetales	049	Polyporales	050	Boletales
051	Russulales	052	Corticiales	054	Auriculariales
055	Geastrales	056	Trechisporales	057	Phallales
058	Gloeophyllales	059	Sebacinales	060	Hysterangiales
061	Atheliales	062	Amylocorticiales	063	Agaricomycetes_ord_ Incertae_sedis
064	Tremellodendropsidales	065	GS28	067	Lepidostromatales
070	Tremellales	071	Cystofilobasidiales	072	Trichosporonales
073	Holtermanniales	074	Filobasidiales	075	Cystobasidiomycetes_ ord_Incertae_sedis
076	Erythrobasidiales	077	Cyphobasidiales	078	Cystobasidiales
080	Sporidiobolales	081	Microbotryomycetes_ ord_Incertae_sedis	082	Microbotryales
083	Leucosporidiales	084	Kriegeriales	086	Agaricostilbales
087	Septobasidiales	088	Pucciniales	089	Platygloeales
090	Helicobasidiales	091	Exobasidiales	092	Entylomatales
093	Tilletiales	094	Georgefischeriales	095	Microstromatales
099	Tritirachiales	100	Geminibasidiales	101	Ustilaginales
102	Urocystidales	104	Atractiellales	105	Spiculogloeales
107	Dacrymycetales	108	Dacrymycetes_ord_ Incertae_sedis	110	Malasseziales
111	Wallemiales	116	Dothideomycetes_ord_ Incertae_sedis	117	Capnodiales

Continuation of Table 13					
Tree number	Tree name	Tree number	Tree name	Tree number	Tree name
118	Pleosporales	119	Acrospermales	120	Tubeufiales
121	Botryosphaeriales	122	Dothideales	123	Venturiales
124	Myriangiales	125	Strigulales	127	Abrothallales
128	Mytilinidales	129	Patellariales	130	Mytilinidiales
131	Trypetheliales	132	Hysteriales	133	Jahnulales
135	Stigmatodiscales	137	Valsariales	139	Minutisphaerales
140	Helotiales	141	Erysiphales	142	Rhytismatales
143	Thelebolales	144	Triblidiales	146	Phacidiales
149	Aspergillaceae	150	Trichocomaceae	151	Thermoascaceae
152	Elaphomycetaceae	153	Onygenales	154	Chaetothyriales
155	Verrucariales	156	Phaeomoniellales	157	Mycocaliciales
158	Sclerococcales	159	Coryneliales	160	Pyrenulales
161	Glomerellales	162	Sordariomycetes_ord_Incertae_sedis	163	Microascales
164	Diaporthales	165	Coniochaetales	166	Sordariales
167	Hypocreales	168	Xylariales	169	Magnaporthales
170	Chaetosphaeriales	171	Melanosporales	172	Phyllachorales
173	Pleurotheciales	174	Myrmecridiales	175	Branch06
176	Ophiostomatales	177	Conioscyphales	178	Hypoceales
179	Boliniales	181	Calosphaeriales	182	Annulatascales
183	Togniniales	184	Xenospadicoidales	185	Coronophorales
186	Pararamichloridiales	187	Trichosphaeriales	188	Lulworthiales
189	Phomatosporales	190	Falcocladiales	191	Savoryellales
196	Ostropales	197	Lecanorales	198	Caliciales
199	Rhizocarpales	200	Peltigerales	201	Umbilicariales
202	Acarosporales	203	Pertusariales	204	Arctomiales
206	Trapeliales	207	Teloschistales	208	Lecanoromycetes_ord_Incertae_sedis
209	Leprocaulales	210	Lecideales	211	Baeomycetales
212	Candelariales	213	Sarrameanales	214	GS36
218	Orbiliales	219	GS33	221	Taphrinales
222	Saccharomycetales	223	GS34	224	Symbiotaphrinales
227	Coniocybales	228	Geoglossales	229	Laboulbeniales
230	Pyxidiophorales	231	Archaeorhizomycetales	232	GS31
233	Arthoniales	234	Lichenostigmatales	236	Sareales
237	Lichinales	241	GS05	242	GS08
243	GS07	244	GS03	245	GS11
246	Branch01	247	GS06	249	GS10
250	GS04	252	Branch03	254	Spizellomycetales

Continuation of Table 13					
Tree number	Tree name	Tree number	Tree name	Tree number	Tree name
255	Rhizophydiales	256	Lobulomycetales	259	Rhizophlyctidales
260	Synchytriales	262	Basidiobolales	263	Endogonales
264	GS21	265	GS22	267	Mucorales
268	GS23	269	Umbelopsidales	270	Blastocladiales
271	GS15	272	Mortierellales	273	Neocallimastigales
274	GS16	277	Olpidiales	278	Monoblepharidales
280	Sanchytriales	281	Zoopagales	283	Kickxellales
284	Entorrhizales				

Table 13: Look up table for tree number and name.

B Zip file

All results of this research are combined in a zip file. The structure of the zip file is shown below. Some files contain numbers in their name. The corresponding subtree name can be found in appendix A, table 13. The folders for BLAST and `pplacer` contain the raw output and the calculations done with this output. All figures for biodiversity are added in the folder `Visualizations biodiversity`.

```
Results
├── Test 1
│   ├── Annotation
│   ├── BLAST
│   └── pplacer
│       ├── Calculations
│       └── Output pplacer
├── Test 2
│   ├── Annotation
│   ├── BLAST
│   └── pplacer
│       ├── Calculations
│       └── Output pplacer
├── Test 3
│   ├── Annotation
│   ├── BLAST
│   └── pplacer
│       ├── Calculations
│       └── Output pplacer
└── Visualizations biodiversity
```