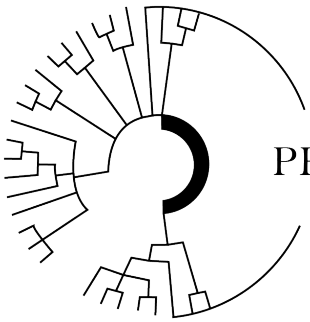
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



PHYLOdiff

Pedro Gardete, 47218
Francisco Ludovico, 47262

Supervisor: Cátia Vaz

Relatório do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho 2022

Instituto Superior de Engenharia de Lisboa

PhyloDiff

47218 Pedro Martim Mendes Alves Casimiro Gardete

47262 Francisco Duarte da Palma Mestre Amaral Ludovico

Supervisor: Cátia Raquel Jesus Vaz

Relatório do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho 2022

Abstract

The biggest challenges in understanding the **evolution** of bacterial and viral epidemics, such as SARS-CoV-2 or some E. coli strains, are to determine their origin, evolution and resistance patterns to the various treatments under study. With the huge volume of data available, including genetic sequencing data, and the rapid volume of information on the treatments tested growing every day, there is a constant need for data analysis, which can be used to detect outbreaks in hospital settings or in the food industry, e.g., by monitoring the spread of antimicrobial resistance, an ever growing concern.

In this context, one important step of data analysis is to infer phylogenetic relationships, involving the construction and comparison of phylogenetic trees or networks for some set of sequence data.

There are some measures that are available to compute in web applications, such as in Phylo.io and TreeComp but they just include the visual comparison of specific kind of phylogenetic trees, namely only including data on the leaves and does not scale for the amount of data under analysis in most real world phylogenetic studies.

PhyloDiff, as a library, is an extension to the Phylo.io tool that allows the analysis of the differences between two phylogenetic trees, while being able to compute trees with large amounts of data and having the ability to be extended in the future with new comparison measures and tree visualization formats. Our project also has a fully-featured web application, just so we can visually demonstrate the results given by our library, which also is an extension to Phylo.io's.

Keywords: evolution; phylogenetic relationships; phylogenetic relationships; extendable.

Acknowledgements

Firstly, I want to thank the people that made everything possible. My parents, Hugo and Zira, who always supported me from the first day of pre-school to be whoever I wanted to be. They always gave me everything to be the best version of myself and to always dream higher. I want to thank them for giving me my best friend too, my dog Baco, that never allowed me to be sad or hopeless, because if I ever felt like that, he would come to me to find a way to cheer me up. Because of all of that, I am forever thankful to the three of them.

I want to thank my girlfriend, Rafaela, for always being there for me when I needed the most, and for the countless hours of study sessions that we did together (and still continue to do) since 7th grade, instead of going out to parties (we went to parties sometimes, but it's always school/college first). She was, and still is, one of the most important pillars in my life, and it's always a pleasure to have a soul mate like her by my side.

I can not forget to mention my friends Miguel and Rafael, my two friends that gave me the privilege of spending almost every day, since the pandemic started back in 2020 with them (although, most of time, virtually). They are someone who I can trust my life with and that I am very happy to know that will cry and help me when I am in my lowest and celebrate my toughest victories.

I wanna thank my partner Pedro, for being with me since day zero of this three year journey. It was a pleasure to see you grow as a person and as an engineer and to write this last chapter of our academic our lives with you as my partner.

To the teachers who I encountered along this path that started 15 years ago, I wanna thank you all, but with special gratitude to teacher Cátia Vaz, who gave us the much needed support and feedback to conclude this important milestone, and my high school Maths tutor, Ana Campos, for she was the one that helped me build my logical thinking that is so important in our study area.

Lastly but not least, I want to thank my grandfather António, who I know will always be looking out for me from the sky and is surely one of the proudest people right now.

September 2022, Francisco Ludovico

To begin I want to say thanks to everyone who has stood with me for the last couple of years, especially my family and friends. I won't be able to write about every single one but I want them to know how much I value their support. While I've never had the most conventional family situation I feel like I've never missed out on anything important in my life thanks to my mom. Without her tireless work to raise me all these years I would not be the person I am today, I can never express how grateful I am for every opportunity she has given me and her unconditional love. When I was younger I spent much time with my grandparents. They were and still are a big presence in my life to this day.

I want to thank my father and my siblings for being there when I need them, my younger brother who I talk with almost everyday. My grandparents on my fathers side were always nice to me and also took care of me when I was younger. My grandmother who still calls me periodically to check up on me and my grandfather Domingos who died while I was still young but I remember him to be a strict but caring man and I know that he always had me in his mind.

I want to thank Francisco who has been a fantastic partner throughout these last three years. It was always comforting to know I was not going through these trials alone. We have supported each other, through ups and downs, and we managed to reach the end of this chapter. We've grown as adults and as professionals and I only hope the best for your future. Thank you so much.

A special thanks to our great teacher and supervisor for this project Cátia Vaz, who has been nothing but supportive of our academic endeavors. I also want to thank every teacher I've encountered on this journey and especially my primary school teacher Graça who I still look back fondly to this day.

As a last note I wanna thank my grandfather Jesuino who I lost last month, as of writing this. He was one of my best friends in the whole world and I know that he is now in a better place looking out for me.

September 2022, Pedro Gardete

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Trees, or more generally, networks, are among the most common combinatorial structures in computer science. In the era of big data, the problem of comparing trees and networks is known to be a task often characterized by particularly high computational complexity. Moreover, when dealing with large quantities of data, that may be remotely stored, and thus can have a significant impact on the overall time and additional costs if it is not efficiently treated. Tree and tree-based network comparison occurs in several areas such as computational biology, structured text databases, image analysis, automatic theorem proving, and compiler optimization.

In computational biology, tree comparison has several applications, such as in, phylogenetics, when comparing phylogenetic trees. In fact, many biological analyses involve the construction of a phylogenetic tree for some set of sequence data, and a variety of methods for inferring phylogenies are available. But the phylogenetic tree produced by an inference method is just a hypothesis, chosen from among possible alternatives. Thus, methods for comparing phylogenies that can reveal where two trees agree or differ are desirable, to assess the quality of phylogenetic trees and analyze different phylogenetic methods.

There are several tree visualization and comparison tools. For instance, there's TreeCmp[1], one of the most famous tools when it comes to phylogenetic trees comparison, while having a rich library with many different comparison measures ,it has poor scalability to large trees. Another example is Compare2Trees [2].It annotates nodes using a novel algorithm for tree structure comparison. However, it does not scale well when comparing large trees. Another tool for side-by-side comparison is TreeJuxtaposer [3]. While it is scalable for large trees and enables users to compare subtrees, it has not been receiving maintanance for quite some time and is cumbersome to start because it relies on old Java security standards. The final drawback with some of the current state-of-the-art tools is their availability. The already mentioned Compare2Trees was built as Java Applet and therefore not so easy to use with the modern browsers. Desktop software, on the other hand, may require legacy systems or only run on specific platforms, just like PhyloComp [4], which is not readily available to download and installation, while suffering from the previously described issues.

While we were analysing some of the tools that already exist to compare and highlight the differences between a pair of trees, we were able to identify three main drawbacks with those same tools: (1) a lack of functionality to compare trees with a format different than a dendrogram [5]; (2) they are not ready to be extended with new comparison measures; and (3) the use of technology that can be considered outdated, thereby compromising their usability.

PhyloDiff's[6] design relies on solving some of these problems. It is a library allows the comparison of two trees with the branch highlights based on certain comparison measures, made using recent technology and tools, such as Phylo.io[7] (of which is an extension) and NodeJS, and the ability to be easily extended with new comparison measures and tree layouts. This project will also have an application so that the results that our library gives can be visualized, making it easier for those conclusions to be analyzed.

To compare such trees, they need to be organized in a computer-readable form. A group of mathematicians in 1986 proposed the Newick tree format [8]. This notation consists in representing graph-theoretical trees with edge lengths using parentheses and commas. Such format can be used to either represent rooted or unrooted trees.

When faced with the need to compare large numbers of nodes within a pair of trees, we need an algorithm that assures both time and resource efficiencies. With this thought in our minds and after comparing the different algorithms that could fit our needs, we came across three variants of the same metric:

- Robinson and Foulds (RF) [9]
- Weighted Robinson and Foulds (WRF) [10]
- Generalized Robinson and Foulds (GRF) [11]

As such, this project will have the following mandatory requirements:

- Development of a library with the three different variations of the Robinson and Foulds metrics to find the branches that differ from a pair of trees, with proper load tests.
- Development of an application to view the differences between the two trees calculated by the algorithms that were already described. These differences will be highlighted by painting the different branches with a color. This module will use pieces of software that were already implemented by Phylo.io [7] (an easy to incorporate open source project) and will support a way to generate a dendogram with the highlighted different branches.
- Serialization of the output of each algorithm. Once calculated, allow the result to be shared and/or exported without any additional computing costs.

The following requirement is optional, since it will only be developed if the mandatory ones are concluded with success:

- Adding a new visualization format other than the dendogram, like the radial tree format.

The next chapter of this document, chapter 2, we will describe what exactly is the process of comparing a pair of phylogenetic trees, while explaining the comparison measures that we will develop, some examples of those same measures and examples of tools to compare and visualize the differences in such trees. Chapter 3 will be about the PhyloDiff library and application. The implementation of our measures and the visualization of the trees will be discussed there. In chapter 4 we will compare the run time execution of our algorithms with the ones implemented by TreeCmp[1] and draw conclusions about them. Finally, in chapter 5 we will conclude this document and talk about some future work that could be done with our project.

Chapter 2

Phylogenetic tree comparison

This chapter is divided in three sections. Firstly, we will give some context on what exactly is a phylogenetic tree and the different formats that they can be represented in. After that, we will present the comparison metrics available in our library. Finally, we will expose some of the comparison and visualization tools that already exist.

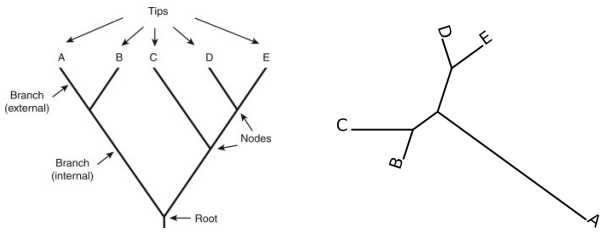
2.1 Phylogenetic trees

Phylogeny studies the evolutive relation between different groups of organisms, such as species, populations, etc, that share one or more common ancestors.

Phylogenetic analysis allows us, by comparing the genes of several species or individuals among the same specie, to infer how they relate to each other. This is done by grouping organisms by their levels of similarity, that is, the closer a group of species is the closer they are to their common ancestor. Representing these relations can be done in many ways, but, arguably, the most common of them are phylogenetic trees. These trees have nodes and branches. A node represents a taxonomic unit (*táxon*) and can either be internal (in the middle of the tree) or terminal (in the tips of the final branches). A branch connects any two nodes that are adjacent.

Phylogenetic trees can be divided in two big groups, with root and rootless. Phylogenetic trees with root have nodes, branches, clades and, of course, a root. The nodes represent the relation between two different sequences (*taxa*) and, when they are internal, do not always have associated data (it depends on the phylogenetic inference algorithm). Clades can be described as partitions of the original tree that have a common ancestor. Finally, branches represent the connection between sequences of information. The root represents the common ancestor between every taxa in a tree - figure 2.1a. The rootless version differ from their rooted counterpart by just not having a root - figure 2.1b.

These trees can have many visualization formats, such as dendrograms and the radial tree formats, which we will dive into in the following subsection.



(a) A rooted tree (b) An unrooted tree

Figure 2.1: A small examples of two phylogenetic trees representations

2.1.1 Tree layouts

There are several different ways of representing trees. Here, we will focus our attention on two of them: the dendogram and the radial tree format. The implementation of the last one, as we already explained, was part of the optional requirements of this project.

The dendogram is a tree format used to visualize and classify taxonomic relationships frequently used to illustrate the arrangement of the clusters produced by hierarchical clustering. The name dendogram derives from the two ancient Greek words *déndron* and *grámma*, meaning tree and drawing, respectively. As figure 2.2 suggests, the dendogram represents a tree in what we can recognize as a common tree-like format.

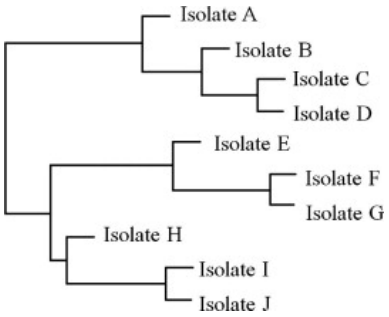


Figure 2.2: A small example of a tree in the dendogram format.

The radial tree format, or radial map, is a method of displaying a tree structure in a way that expands outwards, radially. It is one of many ways to visually display a tree with examples extending back to the early 20th century. In use, it is a type of information graphic, just like the dendrogram.

The specific variant of the radial layout that we intended to use in our project was the daylight method, that is illustrated in figure 2.3.

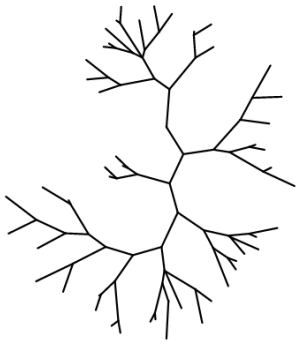


Figure 2.3: A small example of a tree in the radial format.

2.2 Comparison measures

Over the years, several ways of comparing two sets of data structured like a tree have been developed. Algorithms that keep track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets, like the famous Disjoint Set/ Union Find [12], are very useful to study such structures. Tree comparison methods, relies on mathematics to compare a set of two trees. These measures can either expose the similarities and the differences between two trees or be based on the branches' weights. As such,tree comparison measures can be divided in four big groups:

- Rearrangement based
- Based on branch lengths only
- Topology based
- Topology and length based

As requirements of this project, we decided that we would only be focusing our attention on metrics on the last two groups, that will be presented in the following subsections.

2.2.1 Robinson & Foulds

Our first comparison measure falls in the topology based group. The Robinson and Foulds or symmetric difference metric, often abbreviated as RF distance, is a simple way to calculate the distance between a pair of trees in terms of how similar (or dissimilar) they are. It does that by counting the number of not shared clusters in both of the trees that we want to compare.

Defining this concept mathematically, let us consider T being one of the trees that we want to compare, for each r in T, $C(r)$ is the number of leaves of the sub-tree with root r, in

other words, $C(T) = \{C(r) \mid r\ in\ T\}$. With this in mind, the RF distance is calculated using the following equation:

$$DRF(T1, T2) = \frac{|C(T1)/C(T2)| + |C(T1)/C(T2)|}{2}$$

Now, let's use this equation to calculate the distance. Consider the following two trees in Figure 2.4.

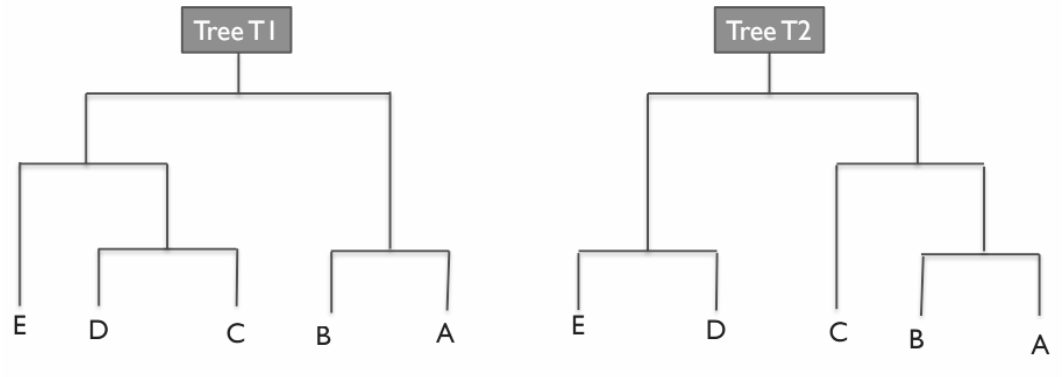


Figure 2.4: A pair of two phylogenetic trees.

Analysing both of the trees, we can write down the clusters that each one has.

- $C(T1) = \{\{E\}, \{D\}, \{C\}, \{B\}, \{A\}, \{D, C\}, \{B, A\}, \{E, D, C\}, \{E, D, C, B, A\}\}$
- $C(T2) = \{\{E\}, \{D\}, \{C\}, \{B\}, \{A\}, \{E, D\}, \{B, A\}, \{C, B, A\}, \{E, D, C, B, A\}\}$

We can notice that each of the trees has two clusters that the other one doesn't. Tree 1 has clusters {D,C} and {E,D,C}, which tree 2 does not, and tree 2 itself also has two unique clusters, {E,D} and {C,B,A}.

Taking the equation, we can conclude that $DRF(T1, T2) = \frac{2+2}{2} = 2$ and that these two trees are no so different after all.

As we can see, being a topological measure, RF distance only allows two trees to be compared if they have the same set of information on their leaves and it doesn't consider any branch lengths. Two variations of this metric where then developed and their cern is based on the two comments that we made.

2.2.2 Weighted Robinson & Foulds

One of the variations of the original Robinson and Foulds distance, Weighted Robinson and Foulds, or Robinson and Foulds Length, has the final goal of showing the sum of the differences in branch lengths in a pair of trees.

Let $A|B$ denote the split of a tree T into two non-empty sets, A and B. Removing and edge e from the tree divides the tree into the two sets. Now, consider *S1* and *S2* be the sets

of all splits in the trees T1 and T2, respectively, the Weighted Robinson and Foulds distance can be calculated with the following equation:

$$DWRF(T1, T2) = \sum_{A|B \exists S1 \cup S2} |w1(A|B) - w2(A|B)|^k$$

Where:

- $wi(A|B)$, with i=1, 2, denotes the length of the corresponding edge in both trees
- $wi(A|B)$ is considered to be equal to 0 if the branch is only present in one of the trees we want to compare
- The original version of this measure considers k to be 1, but some other versions of it consider k to be equal to 2

Now, taking the same pair of trees that we used to exemplify the RF distance but adding some lengths to their branches – Figure 2.5 –, let's calculate the WRF distance of those two structures.

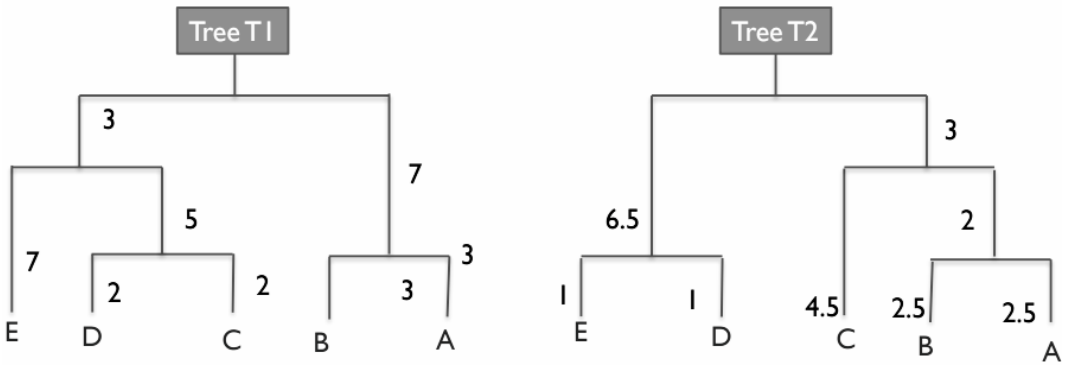


Figure 2.5: A pair of two phylogenetic trees with branch lengths.

Splits	w1	w2	difference
$\{E \mid D, C, B, A\}$	7	1	6
$\{D \mid E, C, B, A\}$	2	1	1
$\{C \mid E, D, B, A\}$	2	4.5	2.5
$\{B \mid E, D, C, A\}$	3	2.5	0.5
$\{A \mid B, C, D, E\}$	3	2.5	0.5
$\{B, A \mid D, C, E\}$	7	2	5
$\{C, D \mid E, B, A\}$	5	0	5
$\{E, D \mid C, B, A\}$	0	6.5	6.5

Table 2.1: Matrix of the branch differences between the two trees

Now that we have all of the branch differences, we can conclude that the WRF distance of these two trees is equal to 27.

2.2.3 Generalized Robinson & Foulds

As for our last comparison measure, which is a much more recent one when compared to the other two that we have already mentioned, the Generalizaed Robinson and Foulds distance, or GRF distance, is more benevolent in terms of what set of *taxa* each of of the trees that we intend to compare have.

The Generalized Robinson and Foulds distance allows the comparison of any structures that can be described by multisets of multisets of labels. Thus, in our phylogenetic tree setting, this means that our data structures are not restricted to be defined on the same set of taxa.

In addition, for every pair of phylogenetic trees, it considers not only their shared clades, but also the ones that are not common between the two trees, comparing their dissimilarities, thus producing a distance with high resolution.

2.3 Existing tools

The advancement of programming technologies has been able to help biology over the last few years. Based on some comparison measures, programmers have been able to develop tools, let them be visualization or comparison ones (sometimes both of them at the same time), that are able to do the the hard work for the biologists. By hard work we mean computing and comparing how similar two phylogenetic trees are.

In the next subsection, we are going to describe some of the existing tools that are very commonly used in the phylogenetic inference context.

2.3.1 Comparison and Visualization Tools

We already stated and gave some examples of tools that are able to compare and show the differences between a pair of phylogenetic trees. We deeply explored and analyzed some of these tools before starting our project. Regarding comparison tools, we discussed the solutions that TreeCmp has for Robinson & Foulds and for Weighted Robinson & Foulds. We were able to identify some things that could be improved in those solutions, such has the usage of the BitSet data structure [9] to try to improve the time complexity of those algorithms.

When it comes to visualization, one that will even be incorporated in our project, although still in the beta version, is Phylo.io. This tool doesn't only serve the purpose of showing the different branches in a pair of trees, since it too comes with some solutions, although we won't be using them in PhyloDiff. The developers of this tool, who we talked to several times, either via e-mail or video calls, were able to solve some of the problems that their old tool had, like the time it took for trees with more than twenty seven thousand clusters to load into their application.

Chapter 3

PhyloDiff

PhyloDiff is a library that extends Phylo.io's and is suitable for users to efficiently compare a pair of two phylogenetic trees. Another feature of this library is the ability to easily add new comparison measures to it, by providing an object to it, with mandatory and non-mandatory properties, described ahead in this document.

By adapting Phylo.io's application, our project also comes with an application, that allows user to visualize the trees they want to compare, in the form of a dendogram (supporting the addition on new visualization methods, like the radial format described previously in this document), and to highlight the branches of both trees so that those differences can be easily identified. Other features of this application are the support for the download of the compare session into a file that can further be loaded by other users into their machines, so that they can continue to work on that comparison, screenshots of the two containers with both trees inside them, the possibility to control both containers' zoom at the same time, and the support for keyboard shortcuts, so that some features our application can be fully accomplished by typing in a specific key combination. This project's code can be found on Github, where we have a brief explanation of our project's scope and how to run it on the user's computer.

3.1 Project architecture

When using our application, along with our library, the user provides the two trees, in the Newick format, that will be passed down to our library, where the results for the available metrics and the information about which branches should be highlighted will be returned from. Figure 3.1 represents the architecture of our project.

We can identify three important agents in it, that will be better described in the next few sections.

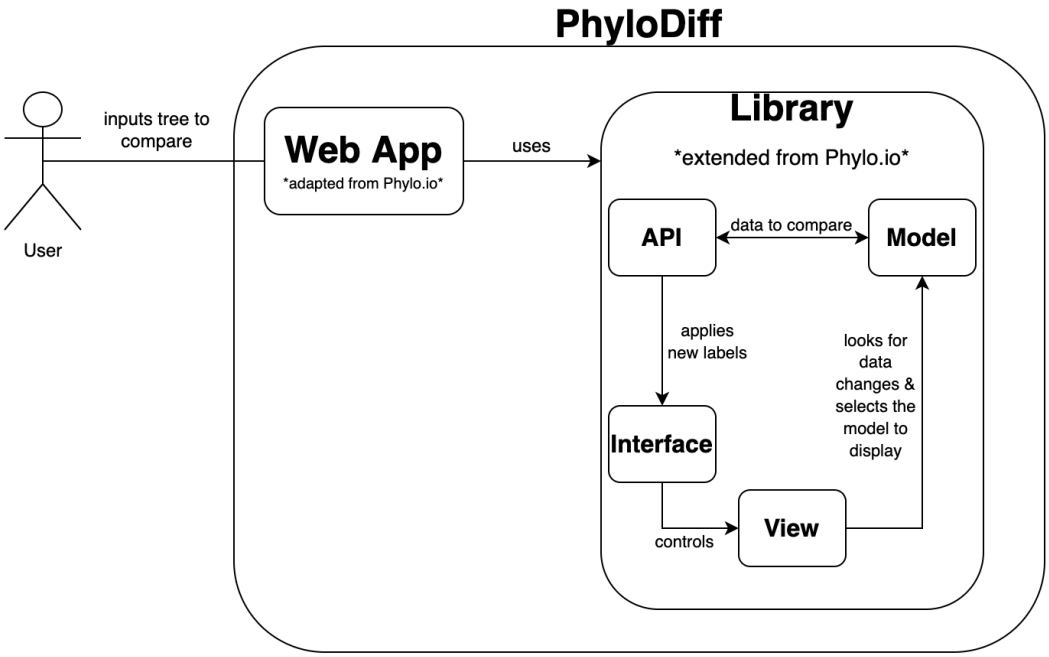


Figure 3.1: Architecture container diagram

3.1.1 API

This module is responsible for checking which measures are available in our library and compute result based on those same metrics and the trees that were inputed for comparison. These results that come from our functions inspired by mathematical formulas, will then be displayed to the user.

The value that corresponds to the color of a branch highlight is being computed at the same as the metric is being calculated.

3.1.2 Model

The model module has the responsibility of parsing the trees in Newick format that the user wants to compare. It then builds a representation of that structure with its tree-search methods that will then be used to look some information up on those same trees.

3.1.3 View

Our module that manages the visualization functionality has the responsibility of constructing the tree view and the layout associated to that tree.

The View module also has the capability of using the list of nodes, generated by some of the API components, to paint the branches on both trees based on the measure that the web application user wanted to compute.

3.1.4 Interface

Finally, the interface module is the one that has full control over the View module. Therefore, it is responsible for the deciding which tree model and highlight are being displayed inside each of the tree's containers.

This module needs to be manually updated to display new highlight labels that are added by the API module.

3.2 Phylo.io modifications

Before we started to even implement any metrics or other new features into the Phylo.io environment, firstly we had to make some changes and improvements to this tool. Table 3.1 displays these modifications.

	Removed	**Adapted**	**Added**
leaves	X	-	-
comparison	X	-	-
deep_leaf_list	-	X	-
compute_metrics	-	X	-
leaf_info	-	-	X
Metrics Object	-	-	X
id_set	-	-	X
get_next_node	-	-	X
get_clusters_rf	-	-	X
get_clusters_wrf	-	-	X
get_clusters_grf	-	-	X
compute_RF_distance	-	-	X
compute_WRF_distance	-	-	X

Table 3.1: Changes made to the Phylo.io tool

We removed the leaf property that the models had, since we wouldn't be using it, and the comparison function, whose objective was to calculate the intersection of the cluster sets of both trees to then pass that information to the function that is in charge of highlighting the branches. This function caused the load times of bigger trees to be way too long, and since we have a different way of calculating which branches are going to be painted when the user wants to see how different a pair o trees is topology-wise, we deleted it.

As for the components that we adapted, the `deep\_leaf\_list` and `compute\_metrics` functions are on that list. For the last one, we had to add our own function that are going to compute the metrics we implemented. Regarding the `deep\_leaf\_list` function, before our modifications, it added to every node a list containing the leaves' names up to that point, starting from the tip leaves of the tree. Since we only needed that information from the root, right now it creates a list with those same names from the beginning.

In terms of new tool components, we added the `id\_set` Hashmap, that contains the unique identifiers for each node of both trees, an object with the metrics to be imported, so that we can dynamically import new metrics different from the R&F ones if we want to, the functions that return the cluster sets for each of our default metrics, as well as the functions that actually give us the final results for them.

3.3 PhyloDiff metrics implementation

In this section, we will describe the solutions that we came up with to our three algorithms, by briefly explaining their pseudo code and analyzing their time complexities.

3.3.1 Robinson & Foulds

For the Robinson & Foulds, we adapted a version of the metric that uses the BitSet data structure [9]. We initialize a HashMap [13] that has an id that corresponds to a node on a tree and a BitSet that represents a cluster of nodes. For each node, we then instantiate a BitSet that represents the cluster of the current node. After that, we check if the node has any children, and if it does, we go do the same before going to to rightmost node to the one we started in. In the end, the cluster to the node being analized is going to be the union between its left and right children, if it has any. We can conclude that our algorithm has a prefix route, since it starts on the leftmost node, and only goes up to its parent node after analysing the node to its right.

With the clusters of both trees obtained, we then subtract to the total number of clusters in both trees the number of ones that are not shared between them, and divide that result by two, as the following code snippet suggests:

```
if(shared_clusters < total_clusters) {
    rfDistance = (total_clusters - shared_clusters) / 2
}
```

The following structures, figure 3.2 and table 3.2, represent the pseudo code for the *get_clusters_rf* and its time complexity, respectively:

Algorithm 1: RF Algorithm

```
1  function get_clusters_rf(idSet)
2      // *size will depend on HashMap implementation
3      map ← HashMap < Int, BitSet > (size)
4      node ← this.data
5      do
6          node = next_node
7          bs = BitSet()
8          if node.name ≠ null then
9              nodeId = idSet.get(node.name)
10             bs.set(nodeId)
11         end
12         if node.left ≠ null then
13             bsLeft = map.get(node.left.id)
14             bs ← bs || bsLeft
15         end
16         if node.right ≠ null then
17             bsRight = map.get(node.righ.id)
18             bs ← bs || bsRight
19         end
20         map.set(node.id, bs)
21     while node≠ root;
```

Figure 3.2: Pseudo code of the get_clusters_rf function

get_clusters_rf Algorithm (with n being the number of clusters in a tree)	**Cost**
1. Creating the cluster map	1
2. Getting the node to start the loop	1
3. Do While loop until node != root	n + 1
3.1 Getting the next node	1
3.2 Creating the BitSet for the cluster	1
3.3 Checking if the node has a name, setting its id and setting the id in the BitSet	1
3.4 Checking if node has children	1
3.4.1 Checking if the children are on the left, getting their BitSets and setting them in the cluster map	1
3.4.2 Same set of instructions but for the right children	1
3.5 Setting the current node id and its BitSet in the cluster map	1
Total	O(n)

Table 3.2: Time complexity of the get_clusters_rf function

We can conclude that this algorithm has a linear time complexity, something that was quite predictable since the only operation that can be considered complex is our loop, which will run as many times as there are `n` clusters in our tree.

3.3.2 Weighted Robinson & Foulds

For this version of the Robinson & Foulds algorithm to get the clusters of a tree, a similar approach to its "base" version was taken. The data structure that contains the clusters is obtained with the exact same logic, but, after that is done, we then associate each cluster to its weight in a map that was previously created. The map's keys are the result from converting the corresponding BitSet to strings and the values are subtraction of the weights of the same cluster in both trees.

After getting our map with the clusters and their weights, we get the absolute value of the weight subtraction, and add it to the final WRF result, just like the next code snippet demonstrates:

```
distance_map.forEach(function (value, key) {
    wrf_distance += Math.abs(value)
})
```

Figure 3.3 and table 3.3 represent, respectively, the pseudo code and the time complexity for the `get\_clusters\_wrf` algorithm:

Algorithm 2: Weighted RF Algorithm

```
1  function get_clusters_wrf(idSet, distanceMap)
2      // *distanceMap->HashMap<String,Int> initilized
3      // *before this function
4      // *keys are the conversion from bitset to string
5      // *values are the weight of the clusters
6      clusterMap ← model.clusters
7      node ← this.data
8      do
9          // *the cluster map is obtained before
10         // *using the same logic as get_clusters_rf
11         if !distanceMap.has(bitset.toString) then
12             distanceMap.set(bitset.toString, node.distance)
13         end
14         else
15             currDist =distanceMap.get(bitset.toString)
               distanceMap.set(bitset.toString, currDist-node.distance)
16         end
17     while node≠ root;
18
```

Figure 3.3: Pseudo code of the get_clusters_wrf function

get_clusters_wrf Algorithm (with n being the number of clusters in a tree)	**Cost**
1. Creating the cluster map	1
2. Getting the node to start the loop	1
3. Getting the cluster map like get_clusters_rf	n+1
4 Checking if the distance map doesn't have the current bitset and setting it if not	n
5 Else set the weight diff in the map	n
Total	O(n)

Table 3.3: Time complexity of the get_clusters_wrf algorithm

Just like our `get_clusters_rf` algorithm, `get_clusters_wrf` also has linear time complexity. This conclusion was the one we were waiting for, since this algorithm bases itself on the last one and only adds two other operations with complexities bigger than *O(1)*, that will run `n` times.

3.3.3 Generalized Robinson & Foulds

As we stated in chapter 2, to calculate the Generalized version of the Robinson & Foulds metric, one of the trees has to contain every cluster set from the other tree. As such, our solution for this metric is very similar to the Robinson & Foulds one. In fact, we not only use the same exact function, `get_clusters_rf`, to obtain the clusters from both trees, but we also use a pretty similar way of calculating the result, with the main difference being the

condition an the requirements to calculate it. In terms of requirements, this metric will only be available to the user if the leaf sets of both trees don't intersect each other and one of those sets is included in the other. As for the condition to get a correct result from this metric, the number of shared clusters must be below or equal to the biggest number of shared cluster possible, or in other words, the size of the smallest tree from the pair. The following code snippet shows this last statement:

```
 if(shared_clusters <= max_shared_clusters ) {
    grf_distance = (total_clusters - shared_clusters*2) / 2
}
```

3.4 Dynamic metric imports

Right from the beginning of this project's development, when we where doing our search for the set of tools that are already available for everyone to use, we came up with a common weakness among them. Even while adding our own versions of the R&F family of comparison measures to Phylo.io, we had to dig up their code to find a way on how to successfully do it.

To tackle this problem, we thought it would be interesting to have a way for future users/modifiers of our tool to have a way of adding new comparison measures to our library, hence why we are dynamically importing metrics. These imports come in a form of an object array, in which of those objects has to have a minimum set of properties. To add new comparison measures do PhyloDiff, one must specify the name of the metric to appear before the user (`name` property), the actual function that calculates the metric's final result (`compute` property) and the conditions (`conditions` property) on when it can be computed, like if the cluster sets of both trees intersect each other, like R&F, or even if one of the sets is completely included in the other. Other information can also be added, like a new highlight label (`highlight_settings` property) that is directly tied to such metric and even the reference to a link with the original source for the comparison measure (`ref` propery), if the people that are adding it to our library intend to show that in our application. For these imports to be made with Webpack, we needed to specify a specific directory path, where the module with the comparison measures will be imported from. Right now, that module is inside of our source directory, most specifically in the `metric_modules` folder. In the `lookup-test.js` add an object to our `metrics` array, by providing an object with, at least, the mandatory properties that we just descibed. After that is done, the `push()` function must be used, by passing that same object as its argument.

We believe this to be one the main selling points of our library, and that it will, most certainly, aid the future enrichment of it, via the addition of new methods of comparing and to visualize the differences between a pair of phylogenetic trees.

3.5 PhyloDiff Web Application

Regarding our web application, to test the features , like comparison measures and new behaviors that our code could've had after the changes we made do Phylo.io, right after we implemented them, we used a simpler version, UI and feature-wise, of our final application, containing only two containers that would host the two trees that we wanted to compare and buttons that, when pressed, would display our metric results. The code for this web application is inside our Examples package, inside another package called Easy, in the `index.html` file. Figure 3.4 shows that same web application.

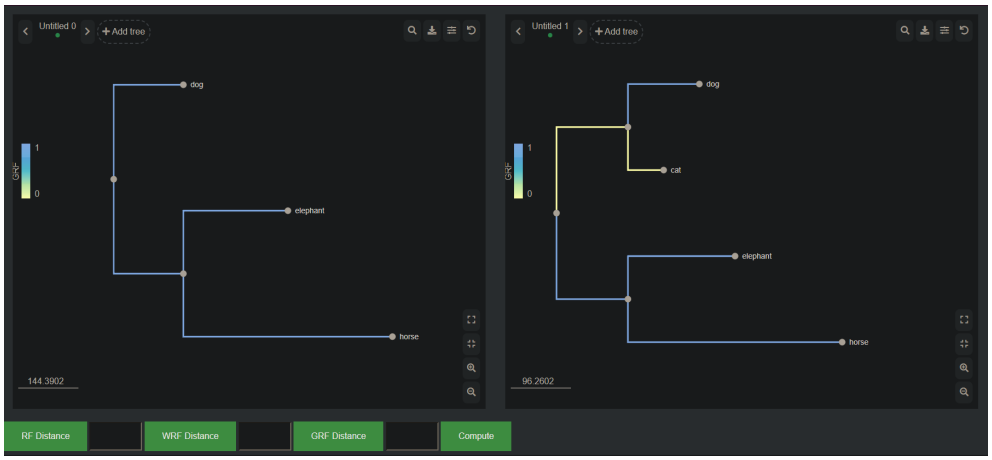


Figure 3.4: Simple web application where we tested new features

For our main web application, we adapted the phyloio.html file that Phylo.io had inside the Website package, which also was inside the Examples package. This application already had some features that we intended to implement, like the ability to save and load a file with the comparison session, to take screenshots of both containers with the trees inside them, to share a link that the user can share for others to see that session on their machine. Phylo.io had other features that can be useful to users of our web application, like the ability to control both containers' zoom at the same time and keyboard shortcuts. The figure below, 3.5, shows our we applications main page, with the features buttons on a sidebar on the left and the settings drop down menus where the user chooses how he paints the trees, in other words, which form of highlight they want to see displayed.

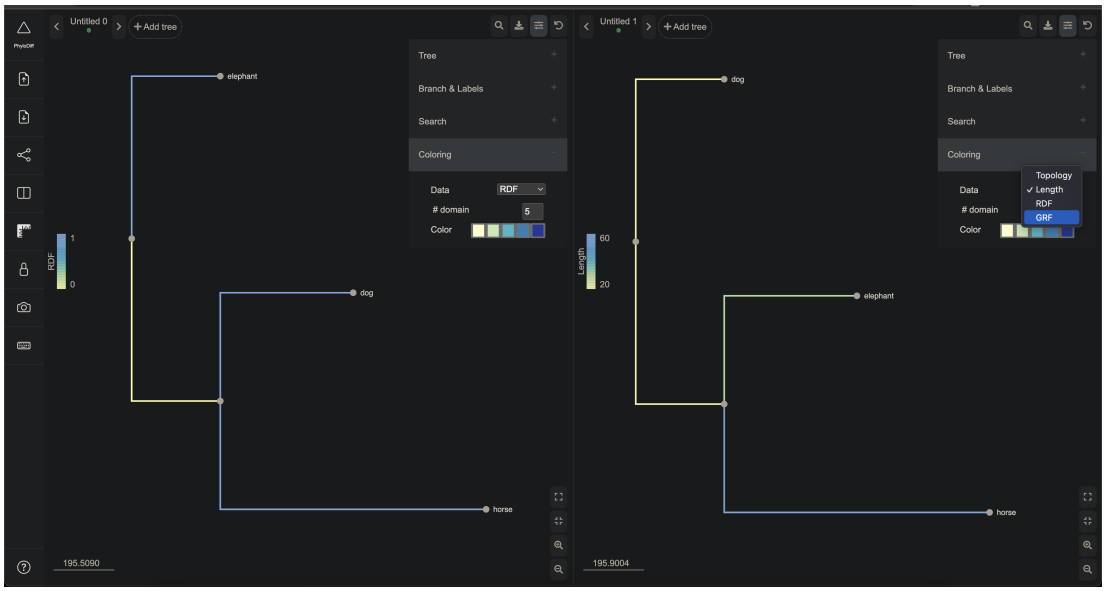


Figure 3.5: Main page of our web application with two trees inside their containers

We then decided to modify this web application to show our metrics results and to have the highlight labels that are connected to them available for the user to choose. Figures 3.6 and 3.7 show the display window, that appears when the user clicks the "Compute metrics" button (the one with the ruler icon) and the dropdown list with the highlight labels we added.

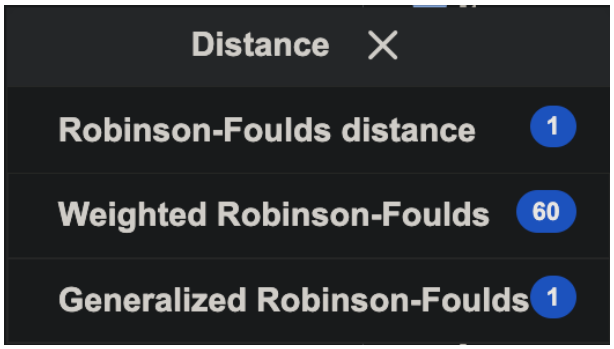


Figure 3.6: Distance display window

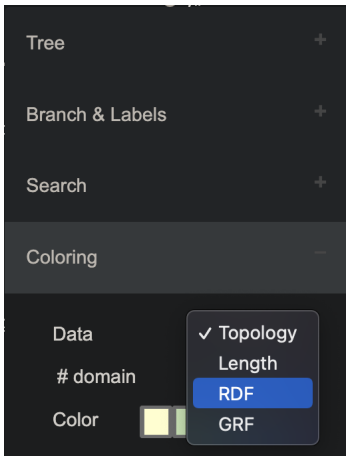


Figure 3.7: Dropdown list with the different highlight forms we added

Chapter 4

Performance tests: PhyloDiff vs TreeCmp

As our project is more based on the research of efficient solutions for a set of comparison measures, it's always important to draw some conclusions about that same efficiency. To do that, we decided to compare our performance tests, that is, the time that it takes for our metrics to present a final result for a pair of trees, with one of the most used tools in this subject, TreeCmp([1]). TreeCmp is a toolset that implements a rather rich set of phylogenetic tree comparison measures, developed using the Java language. Although this project has a complex library of comparison measures, the Generalized Robinson & Foulds metric is missing. Therefore, we will only measure the execution time of the Robinson & Foulds and Weighted Robinson & Foulds metrics.

These sets of tests were made using some built-in functions in both Java and JavaScript. To get the execution times for TreeCmp, we had to download their GitHub repository([14]) and make some adjustments to their tests files. The project already had some unit tests with the metrics we wanted to compare, so the only thing we had to do was to add the `currentTimeMillis()` method of the `System` class. This method returns the current system time in milliseconds, so everything we had to do was call it two times, one before invoking each comparison measure's function and one after them, and subtract both values in the end. For our very own PhyloDiff, a similar approach was used. We called the `console.time()` and `console.timeEnd()` functions just like we did in TreeCmp, except this time we didn't have to subtract any values. The first function starts a timer we can use to track how long an operation takes. When we call `console.timeEnd()` with the same name, the browser will output the time, in milliseconds, that elapsed since the timer was started. Phylo.io offers a set of trees in their web application, as forms of examples of what their tool is able to compute. We used some of those trees to make the following tests.The machine used to run these tests was an Apple MacBook Pro(2019) with an Intel Core i9 9th Gen CPU, 16GB of DDR4 memory @2667 MHz and a 500GB SSD. For the first set of tests, we used a simple phylogenetic tree containing just 4 different leaves.

Tables 4.1 and 4.2 show the results for both Robinson & Foulds and its weighted variant.

PhyloDiff	**TreeCmp**
0.53	2.67
1.23	2.89
0.56	3.02
0.72	2.92
0.86	3.14
0.65	2.51
0.67	2.60
0.66	2.73
0.89	3.02
0.85	2.77
Avg: 0.76	Avg: 2.83

Table 4.1: Execution times (in ms) for the Robinson & Foulds metric for a tree with 4 leaves

PhyloDiff	**TreeCmp**
0.66	3.20
2.46	3.32
1.11	3.12
0.72	2.98
1.04	3.04
0.67	3.17
0.84	3.22
0.93	3.15
1.02	2.99
0.86	3.10
Avg: 1.03	Avg: 3.13

Table 4.2: Execution times (in ms) for the Weighted Robinson & Foulds metric for a tree with 4 leaves

In both cases, we can see that PhyloDiff had a better performance when compared to TreeCmp. We can also denote that, for both tools, the average time to compute the Robinson & Foulds metric is less than the time to compute the Weighted Robinson & Foulds metric. This may be because of the first one not needing to make any computation with the weights of each branch, while the second one has that characteristic into account.

Tables 4.3 and 4.4 show the tests results for one those trees provided by Phylo.io. The tree used for this test has a bigger number of leaves (over 20) than the tree used for the previous test.

PhyloDiff	TreeCmp
1.78	4.62
1.06	3.34
0.96	2.96
1.18	3.27
0.79	2.89
0.91	3.16
1.46	3.03
1.69	3.41
0.96	3.12
2.02	2.87
Avg: 1.28	Avg: 3.26

Table 4.3: Execution times (in ms) for the Robinson & Foulds metric for Phylo.io's example tree #1

PhyloDiff	TreeCmp
5.85	4.82
1.81	2.78
3.14	2.83
1.98	4.12
2.23	2.36
1.74	2.33
3.15	2.74
2.13	4.21
2.31	3.87
2.97	2.66
Avg: 2.73	Avg: 3.27

Table 4.4: Execution times (in ms) for the Weighted Robinson & Foulds metric for Phylo.io's example tree #1

Just like the previous test, for this bigger tree we can see that our tool accomplished better results when compared to TreeCmp. It's also curious to see that, while PhyloDiff has somewhat different values for Robinson & Foulds and the weighted version of the metric, TreeCmp's average values for the two metrics are almost the same.

Tables 4.5 and 4.6 show the times for an even bigger tree, with over twenty seven thousand clusters,that imposed a great challenge, since its size is similar to the ones used in many computational biology studies. As the results demonstrate, once again PhyloDiff has some slightly better values, with an average difference of 50 and 31 milliseconds for Robinson & Foulds and Weighted Robinson & Foulds, respectively.

PhyloDiff	TreeCmp
85.63	129.14
94.31	132.09
88.24	142.84
79.86	130.51
68.50	149.67
100.28	117.11
82.93	135.61
84.81	127.96
77.51	141.32
86.78	136.15
Avg: 84.88	Avg: 134.24

Table 4.5: Execution times (in ms) for the Robinson & Foulds metric for Phylo.io's example tree with over twenty thousand clusters

PhyloDiff	TreeCmp
124.72	168.20
121.86	172.04
128.06	159.87
131.42	166.96
119.81	178.01
147.12	161.61
135.17	142.83
158.33	152.93
134.26	168.24
129.13	163.90
Avg: 132.98	Avg: 163.46

Table 4.6: Execution times (in ms) for the Weighted Robinson & Foulds metric for Phylo.io's "big tree" example

We can conclude that for every set of tests, PhyloDiff has achieved better results. Before we started to develop our comparison measures, we first investigated TreeCmp to better understand how we should implement our solutions for them. While analyzing their code, we identified some things that could take some improvements. This and the fact that our solutions are based on the versions of the comparison measures that use the BitSet data structure, explain the difference in values between PhyloDiff and TreeCmp.

Chapter 5

Final Remarks

We believe to have achieved our main goals of this project. We were able to implement a library with efficient comparison measures and with the ability to easily add new ones in the future, via the solution we develop that dynamically adds metrics to this module of our project. We also successfully adapted Phylo.io's application, so that it could display the results that our comparison measures and highlighting formats returned, and, like so, anyone that wants to use our library to study and draw conclusions about a set of phylogenetic trees can visualise the outputs from our library.

By talking to the developers of the Phylo.io tool, we were also able to improve technical communication skills and learn a new things about some new frameworks and other tools that their prject takes advantage of, like D3.

For future work, we could had some new tree layouts, different than the dendogram format, like one of the many variations of the radial tree format, improve the application's UI and UX, specifically some of the modals that appear when some of our feature buttons are clicked, to add some new features, like the ability to share a link so that other people can access the current comparison session in an easier way, and, finally, to launch our application in a tool like Heroku or Electron, or even to publish it in an internet domain.

Bibliography

[1] Damian Bogdanowicz Tomasz Goluch and Krzysztof Giaro. Visual treecmp: Comprehensive comparasion of phylogenetic trees on the web. *BES Journals*, 03(1):117–119, Nov 2019.

[2] Pietro Liò Tom M.W. Nye and Walter R. Gilks. A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics*, 22(1):117–119, Jan 2006.

[3] Serdar Tasiran Li Zhang Tamara Munzner, François Guimbretière and Yunhong Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, page 453–462, Jul 2003.

[4] Martin Heß Tobias Schreck Philipp Weil Sebastian Bremm, Tatiana von Landesberger and Kay Hamacherk. Interactive visual comparison of multiple trees. *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 31–40, Jul 2011.

[5] Wikipedia contributors. Dendrogram — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Dendrogram&oldid=1067615558`, 2022. [Online; accessed 23-March-2022].

[6] Pedro Gardete and Francisco Ludovico. Phylodiff. `https://github.com/franciscoludovico/phylodiff`, 2022.

[7] David Dylus Oscar Robinson and Christophe Dessimoz. Phylo.io : Interactive viewing and comparison of large phylogenetic trees on the web. *Molecular Biology and Evolution*, 33(8):2163–2166, Aug 2016.

[8] Wikipedia contributors. Newick format — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Newick_format&oldid=1060134696`, 2022. [Online; accessed 23-March-2022].

[9] Eric J. Gottlieb Nicholas D. Pattengale and Bernard M.E. Moret. Efficiently computing the robinson-foulds metric. *Journal of Computational BiologyVol*, 14(6), Jul 2007.

[10] D. F. Robinson and L. R. Foulds. Comparison of weighted labelled trees. *Lecture Notes in Mathematics*, 748, Dec 2006.

[11] Francesc Rosselló Mercè Llabrés and Gabriel Valiente. The generalized robinson-foulds distance for phylogenetic trees. *Journal of Computational Biology*, 28(12), Dec 2021.

[12] "GeeksForGeeks". Disjoint set (or union-find) — set 1 (detect cycle in an undirected graph), 2022. [Online; accessed 6-August-2022].

[13] "flesler". Hashmap class for javascript. `https://www.npmjs.com/package/hashmap`, 2019. [Online; accessed 27-April-2022].

[14] "TreeCmp". Treecmp: comparison of trees in polynomial time. `https://github.com/TreeCmp/TreeCmp`, 2020. [Online; accessed 2-August-2022].