INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Force Direct Visualization for Phylogenetic Trees

Vasco Branco Revés

Projeto Final de Curso

Licenciatura em Engenharia Informática e Computadores

Demonstration

Orientadores: Cátia Vaz, ISEL
Ana Correia, IST

Instituto Superior de Engenharia de Lisboa
Bsc in Computer Science and Computer Engineering

Force Direct Visualization for Phylogenetic Trees

Vasco Branco Revés, 44818

Supervisors: Cátia Vaz, ISEL
Ana Correia, IST

Final Report written for Projeto e Seminário
from the Bsc in Computer Science and Computer Engineering
Summer Semester 2020/2021

June, 2021

Abstract

Nowadays more than ever, epidemics have become an issue of increasing importance. As so, the study of biological sequences and epidemiological data is essential and is watching an exponential growth in the world. Epidemiological surveillance is now a global procedure rather than a country-based one.

Combining information of country specific datasets can now reveal epidemic spreading patterns that were not possible to detect before, but phylogenetic tree visualization algorithms are often hard to use and integrate in analysis frameworks and tools.

Thus, the objective of this project is to develop a solution to be integrated in the PHYLOVIZ online platform that uses the force direct layout for phylogenetic trees visualization.

Phylogenetic trees are build using DNA sequences, to represent evolutionary relationships among organisms, and complementary data to filter and help better understand the information. Complementary data consists on information about strains, gender, location, age, etc.

For a better understanding of phylogenetic trees , force Direct Layout is a non-static visualization that uses the data provided to simulate forces between the nodes of the graph and with this simulate movement.

Although the force direct layout is the main goal of this project, other tools will also be included, like friendly user interface, option to add or remove node labels, expand and collapse nodes of the graph, filters for the complementary data, pie chart graphics, statistics, among others.

The project will consist on a modular solution, in Javascript, HTML and CSS, that should scale in the browser for a considerable amount of data.

The main goal of this application is to make it available to be integrate into applications similar to Phyloviz or to use independently.

Keywords: DNA Sequencing, Phylogenetic Trees, Visualization algorithms, Force direct Layout, Javascript.

Table of Contents

List of Figures

List of Tables

1 Introduction

Biological sequences have a very important roll on molecular biology and the study of this sequences is fundamental to represent evolutionary relationships among organisms and to understand their interaction with others.

DNA sequencing [1] is the process used to determine the nucleic acid sequence in a DNA molecule. Sequence-based typing methods include any method or technology used to determine the sequence of the four DNA bases: adenine, guanine, cytosine and thymine.

Choosing the typing method [2] to use depends on the context that is being used and the problem to solve. Most recent technologies allow high speed DNA and RNA sequencing, called High Throughput Sequencing.

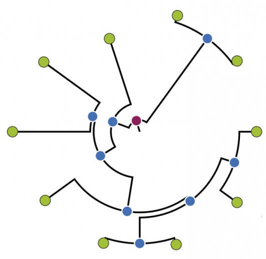
These techniques allow the DNA sequence characterization of bacteria at the strain level providing researchers with important information for the surveillance of infectious diseases, outbreak investigation, pathogenesis studies, natural history of infection and bacterial population genetics .

With the information obtained from these techniques and using phylogenetic inference algorithms [3] it's possible to estimate a Phylogenetic Tree, which is a representation of the relationships of gene or protein sequences to their ancestral sequences and complementary information.

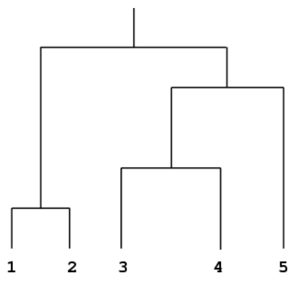
A Phylogenetic tree [4] is composed by nodes and edges, where each node is called a taxonomic unit and edges connect this nodes.

The study of Phylogenetic trees is used to differentiate microorganisms at the subspecies or strain level, allowing to understand the evolutionary history of gene families, tracing the origin and transmission of infectious diseases.

Phylogenetic Tree visualization can be divided into static or non-static visualizations. Static visualization includes all the visualizations where the graphics are static, like dendogram [5] or radial charts [6], as we can see in figure 1.1 .



(a) Radial Tree Example.



(b) Dendogram Tree Example.

Figure 1.1: Static Visualizations

In this project we will focus on a non static visualization of phylogenetic trees, Force Directed Layout, as seen in figure 1.2a.

These are interactive visualizations that allow us to better understand the information given. This algorithm is based on a physical model, simulating physical forces among the set of edges and nodes, based on their relative positions and using this forces to simulate movement.

The main goal of this project is to develop a library for phylogenetic tree visualization, to be integrated in the PHYLOViZ online [7] platform. The solution should also be available to use independently.

Although this platform already has a force directed visualization, figure 1.2b, this will be an improved solution, adding important features not yet implemented. Namely, the collapse and expand feature, reports, the possibility to save image state and improve the user interface, this includes responsive graphics, statistics, filters, etc. Also, this solution should be scalable for large amounts of data without damaging the performance.

1.1 Outline

This report is divided into 6 chapters.

Chapter 2 contains a description of the requirements of the project, both functional and non-functional.

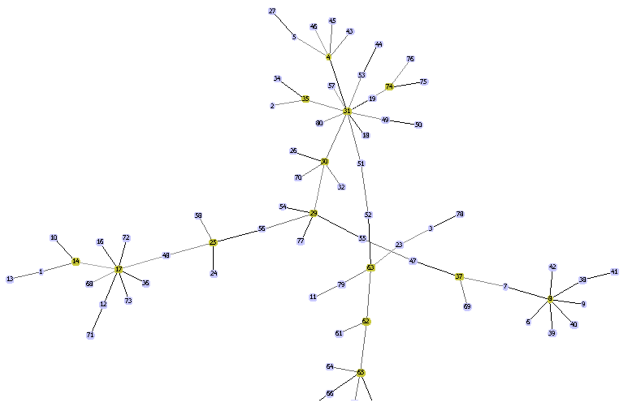
Chapter 3 is an introduction to the technologies and libraries that support the project.

Chapter 4 describes the project's architecture, our approach to implement it, how each component works, as well as the technologies to be used.

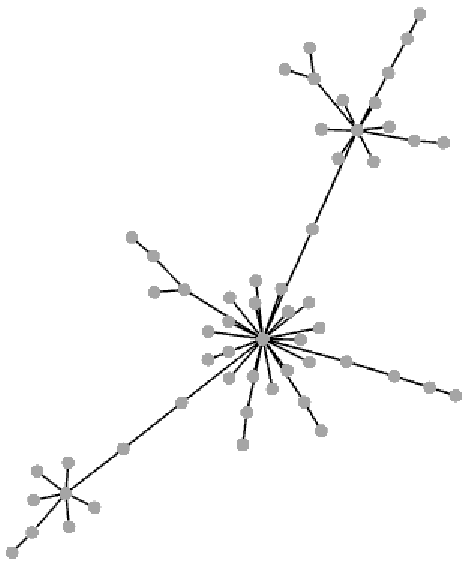
Chapter 5 expresses the implementation details and the logic behind the usage of the algorithms in the project.

Chapter 6 concludes this report, describing the progress made so far and the next steps.

This report tackles the aspects regarding the Force Direct Visualization for Phylogenetic Trees Application back-end and front-end, and everything involved in the realization of this project.



(a) Phyloviz desktop force directed layout.



(b) Phyloviz online force directed layout.

Figure 1.2: Non-Static Visualizations from Phyloviz

2 Requirements

Our aim with this project is to provide a solution to be included in Phyloviz-Online platform that uses a Force Direct Layout [8] for phylogenetic trees visualization.

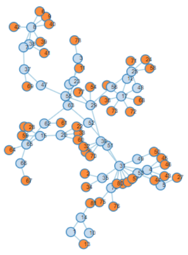
Phyloviz [9] is an open source platform that provides analysis of sequence-based typing methods that generate allelic profiles. This is available as Desktop application developed in Java, available for all platforms, and online application, Phyloviz-Online [7].

This solution will be independent from the other modules of the Phyloviz platform in order to be available to use in similar platforms or as an independent application and it will be developed in Javascript, html and css.

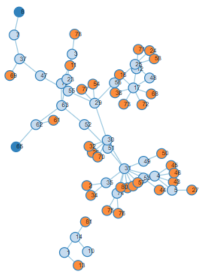
2.1 Functional Requirements

The project will consist on the following requirements:

1. **Force direct layout**, is a algorithm for graphic visualization in the browser. This algorithm is based on a physical model, simulating physical forces among the set of edges and nodes, based on their relative positions and using this forces to simulate movement, figure 2.1a. The nodes are represented by points that repel each other like magnets resulting in as few crossing edges as possible. The Layout of the graph is based solely on the information from the graph data. This algorithm receives as input a phylogenetic tree in either Newick or nexus format and the isolated data. In figure 2.1 we can see an example of the force direct graphic.
2. **Collapse & Expand**, to allow user to collapse or expand a specific region of the graph. This requirement was to be scalable for large amounts of data without damaging performance. As we can see in figure 2.1b, an example of the graphic with 2 nodes collapsed.



(a) Force direct graphic example.



(b) Force direct graphic with collapsed nodes represented in darker blue.

Figure 2.1: Force Direct Visualizations

3. **Pie-Chart Graphics**. The solution should be able to include the possibility to add or remove pie-chart graphics with data from the isolated data file. This graphics are used to visualize filtered information, as seen in figure 2.2, where the data is filtered by profile.

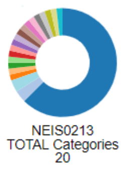


Figure 2.2: Pie chart graphic by profile from Phyloviz online.

4. **Labels & Filters**. The application will include the option to add or remove labels to the nodes making it easier to read the graph. Also, it should include the possibility to search and filter the data, for example, search by country, sex, age, etc.

In figure 2.3, we can see an example of the force direct graphic with no node labels.

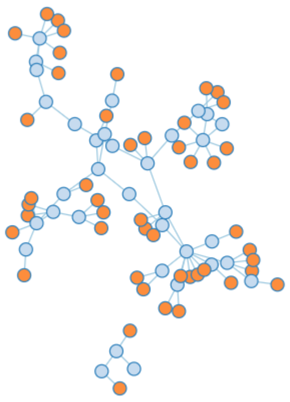


Figure 2.3: Force direct graphic with no node labels.

5. **Statistics**. The application should include simple statistics, to better comprehend the impact or the relevance of the information the user is reading. For example percentage per country or number of isolates associated to a specific strain.

6. **Save Image State**. It should be able to save the state of visualization, this is, the tree, node positions, associated filters, etc.

2.2 Non-Functional Requirements

In the end, a user should be able to:

- Scale the visualization up to 20000 nodes, without damaging performance.
- Use all the characteristics referred, like statistcs, filters, expand and collapse, for all graphics, this includes saved graphics on the database or new ones.

3

Technologies

In this chapter we introduce the technologies used in this project, such as softwares, libraries and resources in order to fulfill the requirements described in the previous chapter:

3.1 Graphic Drawing Libraries

In order to draw force directed graphics we found the three libraries that matched our requirements, D3.js [1], VivagraphJS [2] and SigmaJS [3]. To choose the most appropriate one between those three we decided to test and compare the libraries.

The main requirements for the libraries were:

- Type of render supported, if it supports SVG. This is important because SVG uses vector elements and allows DOM access.
- If it has implemented functions for force directed graphics.
- Allows node customization.
- Allows to save node coordinates / graphic state.
- And finally if it has an active community.

After some testing we decided to discard SigmaJS, it had very little documentation, the last version was released over 3 years ago and as a result wasn't the most appropriated for our needs.

In order to choose between the other 2 libraries , D3.js and VivagraphJS, performance tests were made to test the graphic loading time in the browser.

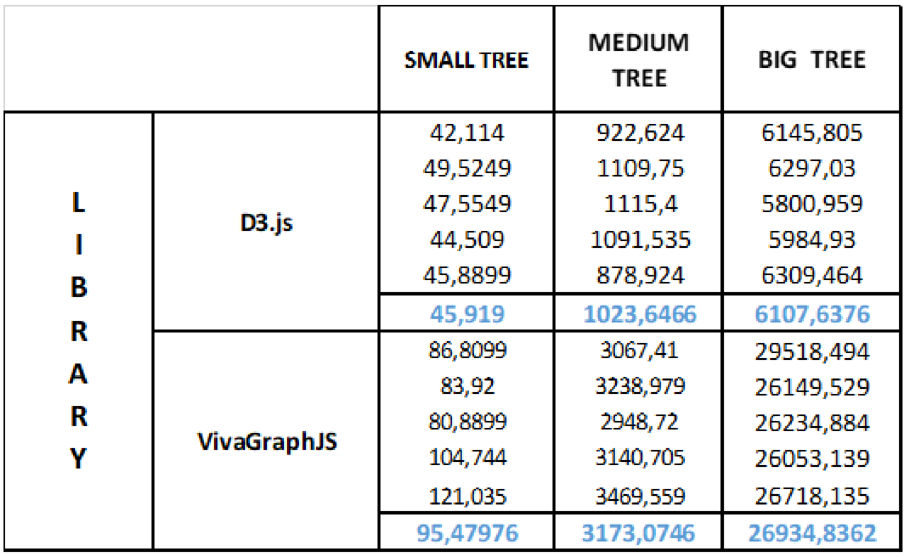
To test this algorithms we used three different trees, a tree with 200 nodes designated 'Small Tree', a tree with 5000 nodes designated 'Medium Tree' and a with around 16000 nodes, designated 'Big Tree'.

For this we did three different force directed algorithms that we tested 5 times for each library and tree.
The tests were made from machine with the following specifications:

- Operative System: Windows 10
- CPU: Intel(R) Core(TM) i3-6100U CPU @ 2.30GHz
- RAM: 8 GB

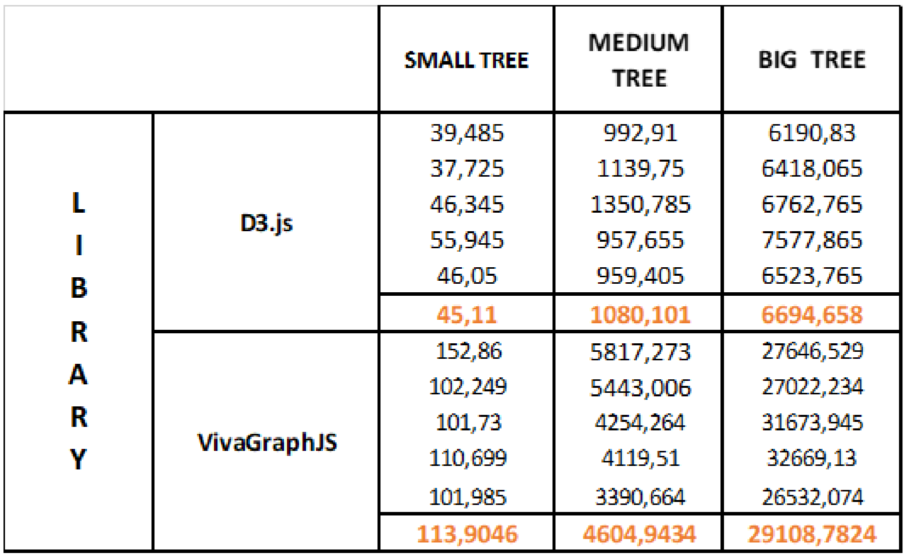
In the first algorithm we created a simple graph with no customization. Results in table 3.1:
In the second algorithm we created a graph with node labels. Results in table 3.2

[1] D3.js - https://d3js.org/
[2] VivagraphJS - https://github.com/anvaka/VivaGraphJS
[3] SigamJS - https://github.com/anvaka/VivaGraphJS



		SMALL TREE	MEDIUM TREE	BIG TREE
LIBRARY	D3.js	42,114	922,624	6145,805
		49,5249	1109,75	6297,03
		47,5549	1115,4	5800,959
		44,509	1091,535	5984,93
		45,8899	878,924	6309,464
		45,919	1023,6466	6107,6376
	VivaGraphJS	86,8099	3067,41	29518,494
		83,92	3238,979	26149,529
		80,8899	2948,72	26234,884
		104,744	3140,705	26053,139
		121,035	3469,559	26718,135
		95,47976	3173,0746	26934,8362

Table 3.1: Average loading time for simple graph with no customization in milliseconds(ms).

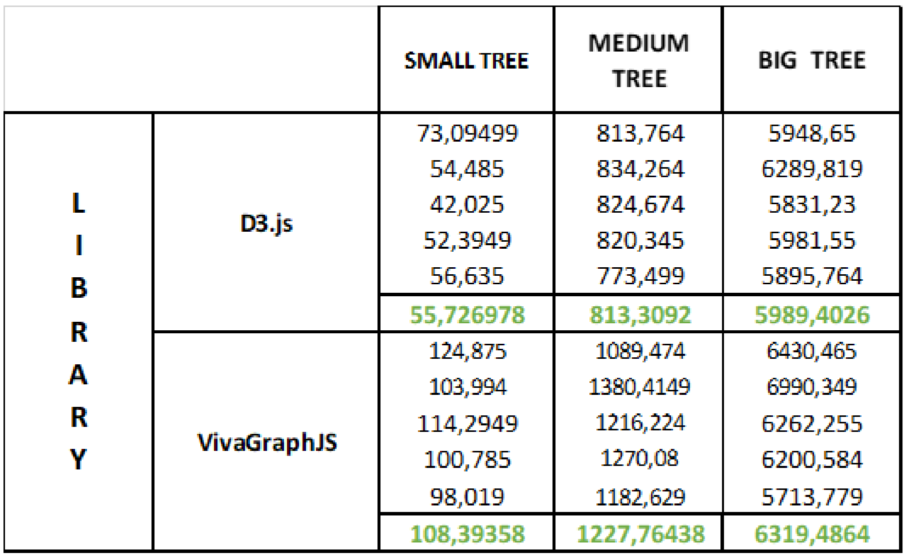


		SMALL TREE	MEDIUM TREE	BIG TREE
LIBRARY	D3.js	39,485	992,91	6190,83
		37,725	1139,75	6418,065
		46,345	1350,785	6762,765
		55,945	957,655	7577,865
		46,05	959,405	6523,765
		45,11	1080,101	6694,658
	VivaGraphJS	152,86	5817,273	27646,529
		102,249	5443,006	27022,234
		101,73	4254,264	31673,945
		110,699	4119,51	32669,13
		101,985	3390,664	26532,074
		113,9046	4604,9434	29108,7824

Table 3.2: Average loading time for simple graph with node labels in milliseconds(ms).

And finally, in the third algorithm we created a graphic using a different rendering function. Results in table 3.3

As we can see in the results, for each test we did five measurements, the values in color represent the average loading time. Comparing this results we can see D3.js was substantially faster in all tests. Also this library has a big online community making it our clear choice to use in this project.



		SMALL TREE	MEDIUM TREE	BIG TREE
LIBRARY	D3.js	73,09499	813,764	5948,65
		54,485	834,264	6289,819
		42,025	824,674	5831,23
		52,3949	820,345	5981,55
		56,635	773,499	5895,764
		55,726978	813,3092	5989,4026
	VivaGraphJS	124,875	1089,474	6430,465
		103,994	1380,4149	6990,349
		114,2949	1216,224	6262,255
		100,785	1270,08	6200,584
		98,019	1182,629	5713,779
		108,39358	1227,76438	6319,4864

Table 3.3: Average loading time for simple graph with labels with different rendering functions in milliseconds(ms).

4 Architecture

In this chapter we will talk about the project's architecture. The project is being developed as a node.js application and its architecture is divided by modules, which we will talk about individually in the following chapters.

4.1 Architecture Diagram

Represented in figure 4.1 is a diagram of the project's architecture described in the next section.

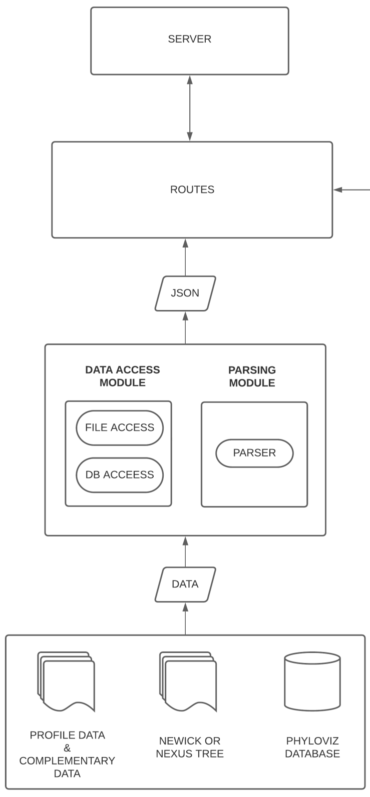


Figure 4.1: Application's architecture diagram

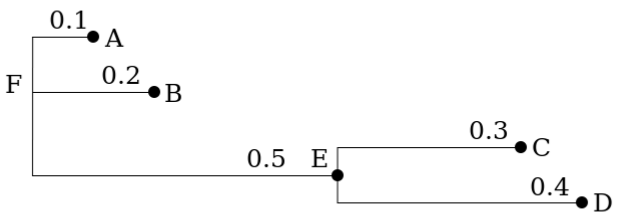
4.2 Modules

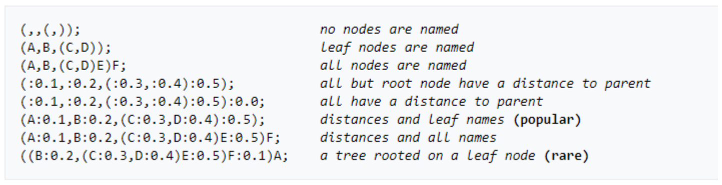
4.2.1 Data Access Module

Data Access module is the module responsible for getting the data to build the tree and the module that communicates directly with server. This data can be received through file input, in newick or nexus format, or from the phyloviz database.

To produce a phylogenetic trees, data can be given in two formats, Newick or Nexus tree format.

Newick is a way of representing trees with edge lengths and node names using parentheses and commas, as seen in figure 4.2.





```
(,,(,));                                 no nodes are named
(A,B,(C,D));                             leaf nodes are named
(A,B,(C,D)E)F;                           all nodes are named
(:0.1,:0.2,(:0.3,:0.4):0.5);             all but root node have a distance to parent
(:0.1,:0.2,(:0.3,:0.4):0.5):0.0;         all have a distance to parent
(A:0.1,B:0.2,(C:0.3,D:0.4):0.5);         distances and leaf names (popular)
(A:0.1,B:0.2,(C:0.3,D:0.4)E:0.5)F;       distances and all names
((B:0.2,(C:0.3,D:0.4)E:0.5)F:0.1)A;      a tree rooted on a leaf node (rare)
```

Figure 4.2: Newick Tree Example.

Nexus tree format is composed by a fixed header NEXUS followed by multiple blocks. Each block starts with BEGIN followed by the block name and ends with END;. The keywords are case-insensitive and comments are enclosed inside square brackets [...]

4.2.2 Parsing Module

This module is responsible for converting the information received and obtain the list of nodes and links, necessary to build to the graphic.

The data is received from the Data Access module to then be passed as a JSON object to the visualization module.

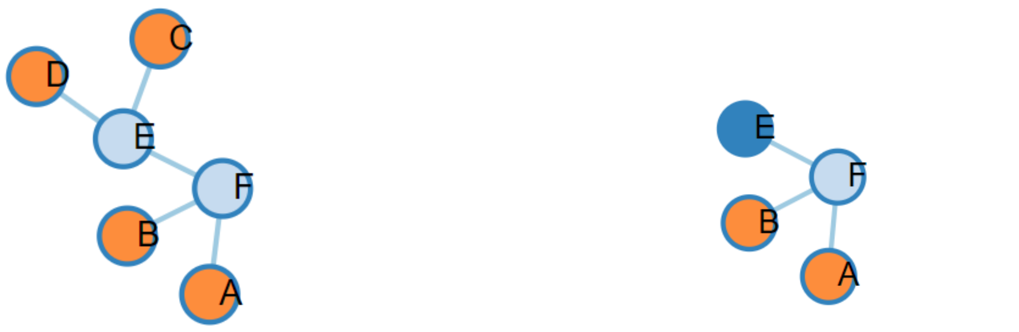
Using the previous example:
(A:0.1,B:0.2,(C:0.3,D:0.4)E:0.5)F
in newick format, the JSON object returned by this module would be:
”nodes”: [
”name”: ”A”, ”length”: 0.1 ,
”name”: ”B”, ”length”: 0.2 ,
”name”: ”C”, ”length”: 0.3 ,
”name”: ”D”, ”length”: 0.4 ,
”name”: ”E”, ”branchset”: [...], ”length”: 0.5 ,
”name”: ”F”, ”branchset”: [...]]
],

```
”links”: [
”source”: ”F”, ”target”: ”A” ,
”source”: ”F”, ”target”: ”B” ,
”source”: ”F”, ”target”: ”E” ,
”source”: ”E”, ”target”: ”C” ,
”source”: ”E”, ”target”: ”D”
]
```

4.2.3 Visualization Module

Visualization module is the main module of the application, as this module is responsible of building the force directed layout and some of the features attached to it, such as expand and collapse, node customization and labels. This module uses the JSON data received from the parsing module and the D3 library to build the graphic.

Continuing the previous example, the graphic produced by this module would be as presented in figure 4.3.



(a) force direct layout.

(b) force direct layout with collapsed node.

Figure 4.3: Newick format tree examples.

4.2.4 Save Module

This module will be the module responsible for saving the visualization state. It should be able to save the graphic layout, this includes node’s coordinates, collapsed or expanded areas, zoom, labels or any filters attached to the graphics.

5

Implementation

In this chapter we will talk about the project's implementation.

The project is being developed as node.js application, we'll go in to detail about the implementation of each module described in the prevoius chapter.

The implementation details described in this chapter are not final as the application is still being developed and all the modules are continuously being improved.

5.1 Modules

5.1.1 Data Access Module

Data Access module is the module responsible for getting the data to build the tree and the module that communicates directly with server. This module handles all file input reading and it can receive three different types of information through file input:

The tree information directly, in newick or nexus format, and the profile data and auxiliary data to build tables to allow to filter the information.

To convert the data received from tree format we use the NewickJS [1] library, available as an npm tool.

This library converts the data to hierarchical JSON. Having the information in this format makes it easier for the parsing module to get the links and nodes list. The hierarchical JSON is composed by nodes and each node contains a branchset, a list of the child nodes. This module is also responsible for reading the data-sets stored in the database.

5.1.2 Parsing Module

This module is responsible for converting the information received and return the list of nodes and links, necessary to build to the graphic.

If the information received is in hierchical JSON the module uses a recursive algorithm that filters through the JSON and obtains the list of all the nodes and list of links, containing the source and target of each link. This lists are then passed as a JSON object to the visualization module.

5.1.3 Visualization Module

Visualization module is the main module of the application, as this module is responsible for building all the views of the application, the force directed layout and some of the features attached to it, such as expand and collapse, node customization and labels.

[1] NewickJS - https://www.npmjs.com/package/newick

To build the views, this module uses Handlebars [2]. Handlebars compiles templates into JavaScript functions, making the template execution faster.

The construction of the force direct layout is build server-side, as the load times are generally faster than client-side scripting, it reduces client-side computation overhead and because this is an interactive layout that provides customized interface for the user to navigate through the graphic.

For this we use the JSON data received from the parsing module and the D3 library to build the graphic. The D3 receives the node list through the forceSimulation method, followed by the links information, specifying the types of force and strength values to be applied. This creates the simulation that generates the layout with movement.

Features like expand and collapse, node labels, or the dragging of the nodes are done using event listeners for different events like clicking, dragging, etc.

For both collapse and expand and node labels features, the approach taken was to change the visibility when pressed making it hidden from the visualization. This way, the data stays intact and we don't have to reload the entire graphic. This makes a big difference in performance when visualizing big trees.

5.1.4 Save Module

This module will be the module responsible for saving the visualization state. It should be able to save the graphic layout, this includes node's coordinates, collapsed or expanded areas, zoom, labels or any filters attached to the graphics. To this moment, this module hasn't been started to be implemented yet.

[2] Handlebars - https://handlebarsjs.com/

6 Project Progress

The original planning made for this project and the project's architecture is continuously being discussed and suffering changes.

Some of the requirements take more time then planned in order to have the best implementation possible, and to better suit our goals.

Even though there were delays and changes made to the initial planning the project's progress is positive so far.

In table 6.1, is represented the initial project's schedule.

Start Date	**Duration (in weeks)**	**Description**
11/03/21	2	- Introduction to Phyloviz and Phyloviz-Online Platform - Preliminary Studies concerning Biology concepts
25/03/21	2	- Writing project proposal - Study of graphic design platforms available (d3, Sigma, Viva-Graph) - Study of pie-chart design techniques/algorithms
06/04/21	1	- Finish and deliver Project Proposal
12/04/21	2	- Testing graph design platforms performance (Speed, compatibility) - Testing platforms performance
26/04/21	2	- Study API's architecture - Start implementing force directed algorithms. - Pie-chart graphics algorithms - How to add filters/labels
10/05/21	2	-Start implementing algorithms in the phyloviz-Online platform -Writing Progress Report
24/05/21	2	-Deliver Progress Report -Continue implementation - Start Tests - How to save state
07/06/21	2	- Finish Tests and refining last details

Table 6.1: Schedule.

To this day, it is implemented the force direct algorithm, with possibility to expand and collapse nodes and the possibility to add or remove labels to the nodes, also at the same time the user interface is continuously being developed.

The next step will be to adjust the parsing module in order to also convert the data received from the Phyloviz database, to add the possibility to receive the isolated data to complete the other features.

All this features are being developed and have in consideration the processing of the data, as the application should be able to scale to large amounts of data, being that the main challenge so far.

References

1. Michael M. Miyamoto and Joel Cracraft. *Phylogenetic Analysis of DNA Sequences.* Oxford University Press, 1991.
2. João A. Carriço, Maxime Crochemore, Alexandre P. Francisco, Solon P. Pissis, Bruno Ribeiro-Gonçalves, and Cátia Vaz. Fast phylogenetic inference from typing data. *Algorithms for Mol Bio.*, 2018.
3. T Heath Ogden and Michael S Rosenberg. Multiple sequence alignment accuracy and phylogenetic inference. *Systematic biology*, 55(2):314–328, 2006.
4. Masatoshi Nei, Fumio Tajima, and Yoshio Tateno. Accuracy of estimated phylogenetic trees from molecular data. *Journal of molecular evolution*, 19(2):153–170, 1983.
5. Wikipedia. Dendrogram — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Dendrogram&oldid=1000269204`, 2021. [Online; accessed 10-June-2021].
6. Geoffrey M Draper, Yarden Livnat, and Richard F Riesenfeld. A survey of radial methods for information visualization. *IEEE transactions on visualization and computer graphics*, 15(5):759–776, 2009.
7. Bruno Ribeiro-Gonçalves, Alexandre P Francisco, Cátia Vaz, Mário Ramirez, and João André Carriço. Phyloviz online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees. *Nucleic acids research*, 44(W1):W246–W251, 2016.
8. Se-Hang Cheong and Yain-Whar Si. Force-directed algorithms for schematic drawings and placement: A survey. *Information Visualization*, 19(1):65–91, 2020.
9. Marta Nascimento, Adriano Sousa, Mário Ramirez, Alexandre P Francisco, João A Carriço, and Cátia Vaz. Phyloviz 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. *Bioinformatics*, 33(1):128–129, 2017.