ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

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

Final Report

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

September, 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 spring-like attractive forces among the nodes of the graphic and with this simulate movement. Depending on the node information, nodes can have different weights resulting in stronger or lighter forces connecting the nodes. The links can also have different sizes according to their set value.

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 application uses the framework Electron [1], to create a cross-platform application. The main goal of this solution is to make it available to be integrate into applications similar to Phyloviz or to use independently.

The project's source code is available in the GitHub repository:
https://github.com/vascoreves44818/PS_G35_2021

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

[1] Electron.js - https://www.electronjs.org/

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1 Context

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.

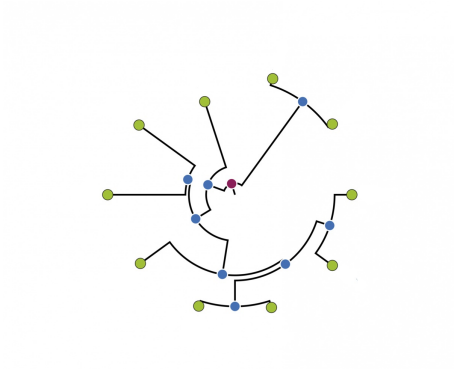
These techniques allow the DNA sequence characterization of bacteria at the strain level providing researchers with important information for the surveillance of infectious diseases, outbreak investigation, pathogenesis studies, natural history of infection and bacterial population genetics.

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.

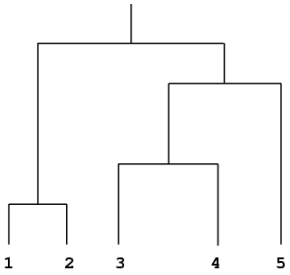
A Phylogenetic tree [4] is composed by nodes and edges, where each node is called a taxonomic unit and edges connect this nodes.

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 1.2.

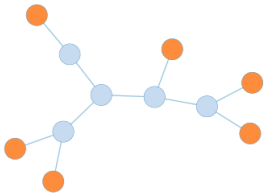


Figure 1.2: Non-static visualizations.

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. In order to demonstrate the use of visualization modules, it was developed a node.js application, cross-platform, using the framework Electron, available to use independently.

Although this platform already has a force directed visualization, figure 1.3, 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.

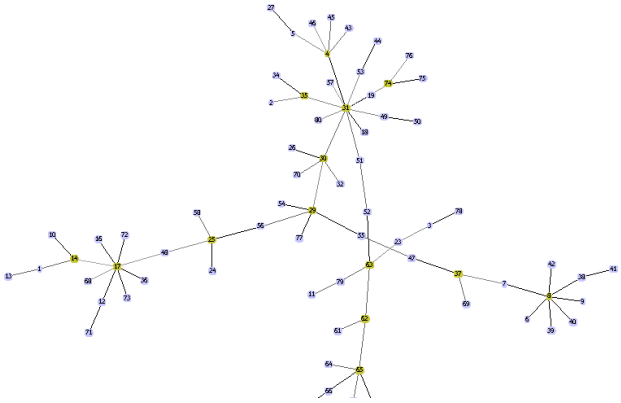
1.2 Outline

This report is divided into 6 chapters.

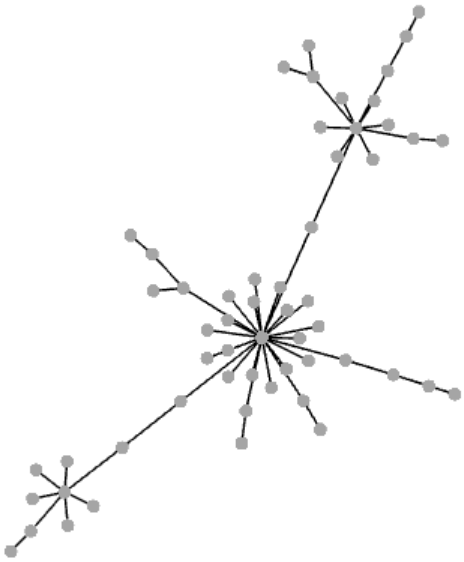
Chapter 2 contains a description of the requirements of the project, both functional and non-functional.

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.



(a) Phyloviz desktop force directed layout.



(b) Phyloviz online force directed layout.

Figure 1.3: Force Direct Layouts from Phyloviz

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.

2

Requirements

Our aim with this project is to provide a module that uses a Force Direct Layout [8] for phylogenetic trees visualization, as well as other features to improve the user experience and make it easier to analyze the data provided.

The solution will be available to be included in Phyloviz-Online platform or similar applications. In order to test the module it was developed a Node JS application, using the framework Electron, to use independently as a cross-platfrom application.

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].

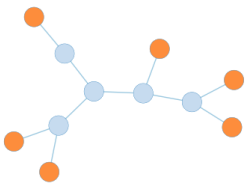
This solution will be independent from the other modules of the Phyloviz platform in order to be available to use in similar platforms or as an independent application and it will be developed in Javascript, html and css.

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 two examples of the force direct graphics.

2. **Collapse & Expand**, to allow user to collapse or expand a specific region of the graph. This requirement has 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.

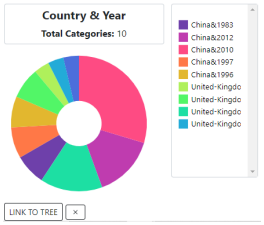


(a) Force direct graphic example.

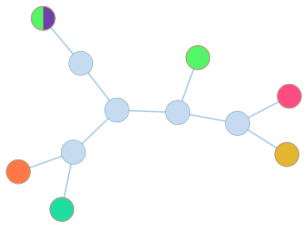
(b) Force direct graphic with collapsed nodes represented in black.

Figure 2.1: Force Direct Visualizations

3. **Pie-Chart Graphics**. The solution includes 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 2.2a, where the data is filtered by isolate. It will also have the possibility to link the pie-chart to the tree and visualize the pie-charts in the tree nodes, as seen in figure 2.2b



(a) Pie chart graphic by country and year from Phyloviz online.



(b) Phylogenetic Tree with node pie-charts.

Figure 2.2: Pie-chart examples.

4. **Labels & Filters**. The application includes the option to add or remove labels to the nodes and links making it easier to read the graph. Also, it should include the possibility to search and filter the data, for example, search tables, search for nodes, filter by profile, isolate, etc .

In figure 2.3, we can see an example of the force direct graphic with node and link labels.

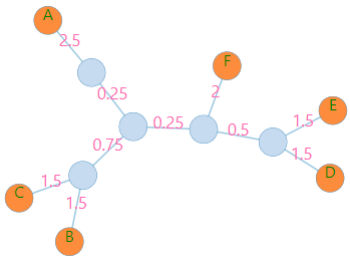


Figure 2.3: Force direct graphic with node and link labels.

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. The application saves all the info in a Json file, this includes, the tree info, node positions, zoom, node colors, or any associated filters. In order to make it possible to resume the visualization at any time.

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 500 nodes designated 'Medium Tree' and a with around 1600 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

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

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

		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).

In the second algorithm we created a graph with node labels. Results in table 3.2

		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

		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).

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.

3.2 D3.js

D3.js is a JavaScript library for visualizing and manipulate documents based on data. D3 lets you create dynamic visualizations using HTML, SVG, and CSS.

D3 allows you manipulate data with Document Object Model (DOM), and then apply data-driven transformations to the document.

In order to build a force directed layout we take use of the D3-Force [4] module.

This module, used to build non-static visualizations like Force directed layouts, implements a velocity Verlet [10], used to calculate trajectories of particles in molecular dynamics simulations and computer graphics.

This module creates a simulation that assumes a constant unit time step and a constant unit mass for all particles. As a result, a force F acting on a particle is equivalent to a constant acceleration over the time period.

It creates a simulation for an array of nodes, and composes the desired forces, according to the type of information being read and the type of layout desired.

The simulation then listens for tick events to render the nodes as they update their relative position in the SVG.

3.3 Electron JS

Electron.js is an open source project, one of the most popular framework for desktop app development.

[4] D3-Force: https://github.com/d3/d3-force

This is a runtime framework that allows users to create desktop-suite applications with HTML5, CSS, and JavaScript.

Its main purpose is to provide fastness, scalability and simplicity to the front-end application code.

The decision to use this framework to develop the project derives from it being used to build cross-platform desktop apps. Using web technologies, it combines the Chromium [5] rendering engine and the Node.js [6] runtime. This means it can be used for both web applications and desktop applications with the same source code.

[5] Chromium - https://www.chromium.org/
[6] Node.js - https://nodejs.org/en/

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.

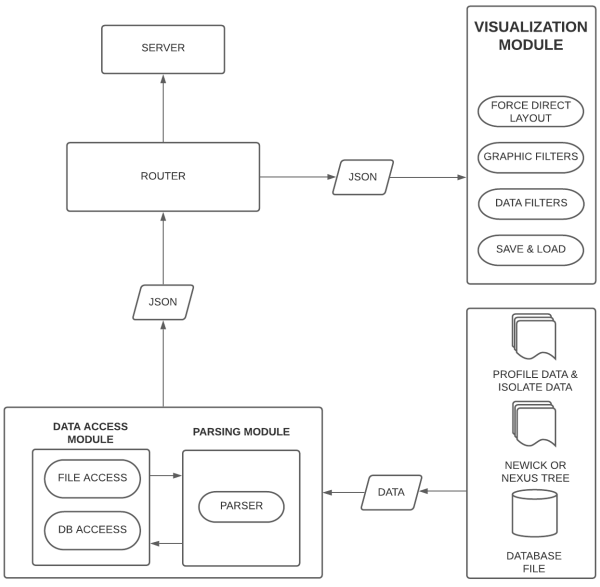


Figure 4.1: Application's architecture diagram

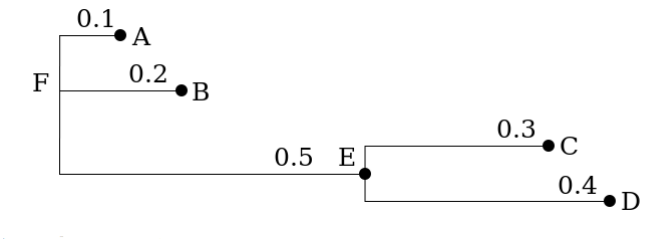
4.2 Modules

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 stored dataset files, in JSON.

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

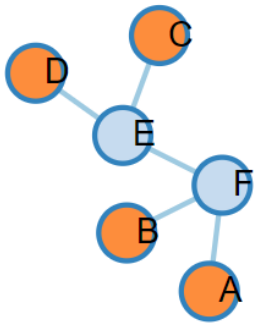
in newick format, the JSON object returned by this module would contain the list of nodes and links, as well as all the other information needed to build the visualization. This includes dataset name, data type, scheme genes, metadata, subset profiles and isolate data, further explained in chapter 5.1.3 .

In document A, we can see the example of the output produced by the parser module after inserting 3 files to application, the newick tree file and the profile and isolate data file.

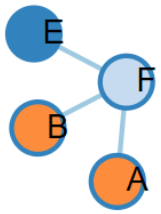
4.2.3 Visualization Module

Visualization module is the main module of this project, as this module is responsible of building the force directed layout and the features attached to it, such as expand and collapse, graphic customization, labels, etc. It is also responsible for the save feature that allow us to save and reload the state of visualization. This feature allows us to save a the information in a JSON file, as shown in document B, where all the information needed to reload the graph is stored. 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.

5

Implementation

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

The project was developed by modules, in which the visualization module, the main goal of this project, is an independent module designed to be used independently and available to integrate similar applications.

In order to test the use of this module, it was developed a node.js application, in which we'll go in to detail about the implementation of each module described in the previous chapter.

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:

- Tree data file;
- Profile and isolate data file;
- Database file;

The tree data file, is a text file with the information in newick or nexus format.

The profile data and auxiliary data, are a text or tab separated file, this information is used to build tables and filter the tree.

Database file is a JSON file, produced by the save feature in this application, which allows you to resume the visualization.

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.

For the stored data-sets, the phylo_db.js file will parse the JSON received and pass it directly to the visualization module.

5.1.2 Parsing Module

This module is responsible for converting the information received and build the JSON object necessary to build to the graphic.

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

The object returned by this module contains the fields:
- `nodes`: List of nodes;
- `links`: List of links with value, source and target node for each link;
- `dataset\_name`: Optional field, if no name is inserted parser assumes name UNKNOWN;
- `data\_type`: Optional field, if data_type is unknown parser assumes value UNKNOWN;
- `schemeGenes`: List of genes from profile data file. Optional field, if schemeGenes is unknown parser assumes value UNKNOWN;
- `metadata`: List of isolated data type values. Example: ["Contry","Year"]. Optional field, if unknown parser assumes value UNKNOWN;
- `subsetProfiles`: Array with list of profile data info;
- `isolatesData`: Array with list of isolate data info;

When the information received is in hierchical JSON, this is, the information was received directly in tree format 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.

For the table elements the parser will split the files by line, resulting in an array where the first element will be the list of headers and the following elements will be list of values for each table row. Then parsed to the json object to return.

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.

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 client-side, to reduce server-side computation 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.

Creating the Simulation

The simulation is what implements the graphic movement in order for it to be non-static visualization. To create the simulation, D3 is given the node and links list through the forceSimulation method. This generates a simulation with no forces. To generate movement specify the types of forces applied and respective values.

The forces involved in the simulation can be:

- Centering: The centering force is responsible of keeping the graphic around the desired center. It is only responsible of modifying the positions of all nodes. It translates nodes so that the mean position of all nodes is at the given position ⟨x,y⟩, according to the node weight. It does not modify velocities, doing so would cause the nodes to overshoot and oscillate around the desired center.

- Collision: Collision force is the force that prevents nodes from overlapping, the value given corresponds to the radius value that repeal other nodes.

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

- Links: The link force pushes linked nodes together or apart according to the value given. The strength of the force is proportional to the difference between the linked nodes' distance and the target distance, similar to a spring force.

- Many-Body: The many-body force allows you to set the strength force. It can be used to attract nodes to the center if the value is positive, or repulse nodes further apart if the strength is negative.

In order to watch the graphic move in the browser, the algorithm listens for 'tick' events on the simulation. On tick, the simulation updates the html elements, nodes and links, position (x,y).

Graphic Filters

Graphic filters allow to change the layout of the graphic. These filters are done using event listeners for different events like clicking, dragging, etc. For both collapse and expand nodes or hide labels, the approach taken was to change the visibility of the elements you want to collapse.

When a node is pressed, all the following nodes and labels are set to 'hidden' 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.

Graphic filters also allow you to change the forces involved in the graphic, such as the charge force and collide force, altering the layout of the graph to the user's preference.

Data filters

Data filters allow you to change the layout of the graphic and filter the information if the profile and isolated data files were inserted. It allows the user to change the node colors, for this the layout as a color key, that can correspond to any of the table headers. When the color function, responsible to assign color to each node, is called it checks if color key is selected, if is assigns colors according to that value if node leaves the default colors.

Save Layout

The save feature is also implemented in this module, this feature allows you to save the graph information, in a JSON file directly to your computer, this includes collapsed or expanded areas, zoom, labels or any filters attached to the graphics.

In document B we can see an example of the saved file, in which it stores the graphic layout. In this example we can see the list of `nodes` with the x and y positions, the graphic forces values, `collideForce` and `strength`, the booleans `nodeLabels` and `linkLabels` as well as the labels size, the `colorKey` for profile or isolate data, and the `transform` attribute that saves the graphic zoom position, if any.

For loading saved files, the script checks if the the information being read is from a saved file by checking the boolean `isSaved` if so it restores all the values for filters, labels, zoom, checks for collapsed nodes, etc.

6

Experimental Evaluation

In order to test the performance of the visualization module, an experimental evaluation was done on the application developed. The tests performed aim to demonstrate the total time for loading the data and building the visualizations and for the memory allocated.

Both tests were done using different datasets, some available as test files in the Docs folder on the project's repository.

This datasets all contain a tree file, a profile data file and an isolate data file.

All the tests were performed on the browser, using the local server on the machine with the following specifications:

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

6.1 Total time for loading and building the layout

This time measurements correspond to the time needed by the application to load the files and build all the visualizations, this is, the force direct layout and both tables, as well as all the features attatched to it.

For all datasets, 3 time measurements were done, and which the results are presented bellow:

Nº of Nodes	161	500	1044	2500	5000
1st	5851	6043	7034	8091	11574
2nd	6074	6062	6959	7901	12303
3rd	5980	6094	6803	7985	10057
Average Time (ms)	5968,333	6066,333	6932	7992,333333	11311,33

Table 6.1: Average loading time for full datasets in the application.

In table 6.1, we can see the average time in milliseconds for 5 different datasets with, respectively, 161, 500, 1044, 2500 and 5000 nodes.

The tests were performed in the browser using the google performance window, where we can analyse the detailed time information.

Analyzing this graphics and taking into account this is a dynamic layout that runs majorly in the client side, we can verify in all datasets that the total time is split evenly between scripting and rendering. Showing the application doesn't max out the machine's cpu.

In figure 6.1, we can see a linear regression graphic, that represents the total time for loading and building the graphics and tables, using the values in table 6.1.

The time sample is consistent with the dataset size, not increasing exponentially the bigger the dataset. Comparing for example the dataset with 500 nodes and the dataset with 5000 nodes, one takes around 6066 ms and the other around 11 311 ms to load and build, only double the time for a dataset ten times bigger.

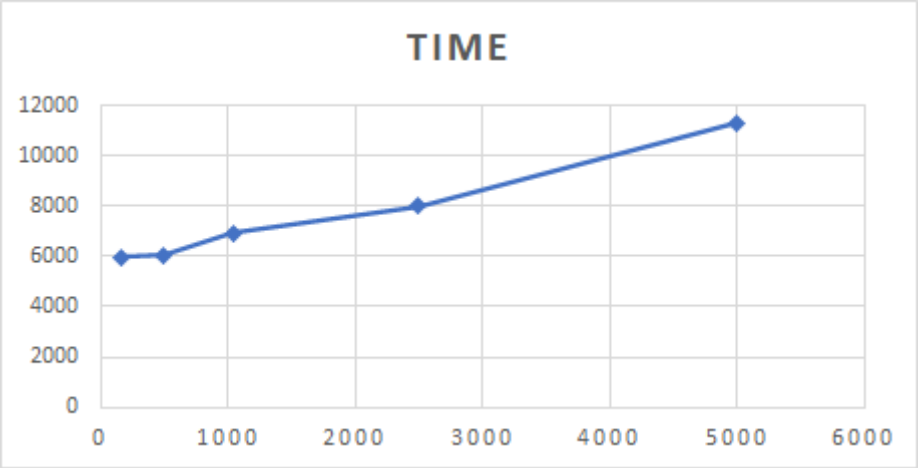


Figure 6.1: Loading time graphic.

6.2 Memory Allocation

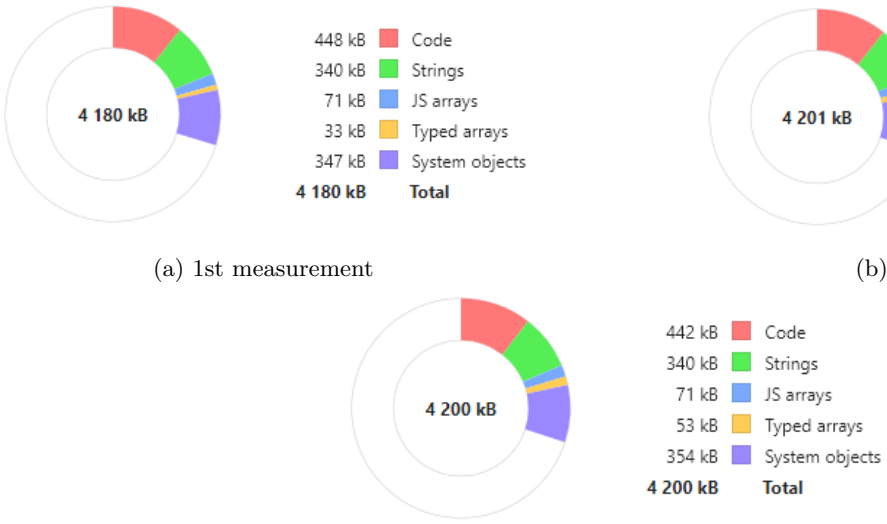
Memory allocation tests correspond to an analysis of the memory allocated by the application during the loading and construction of the datasets.

This are done by taking a snapshot of the memory allocated by the aplication when building the force direct layout and both profile and isolate tables, and it's used to find leaked objects by comparing multiple snapshots to each other.

This measurments were done using the 3 datasets available in the Docs folder. For all datasets, 3 time measurements were done, and which the results are presented bellow.

6.2.1 Small Dataset

In figure 6.2, we can see the detailed results for the memory allocated by the application for the smaller dataset, in which the average memory allocated was `4193.66 kB`.



(a) 1st measurement

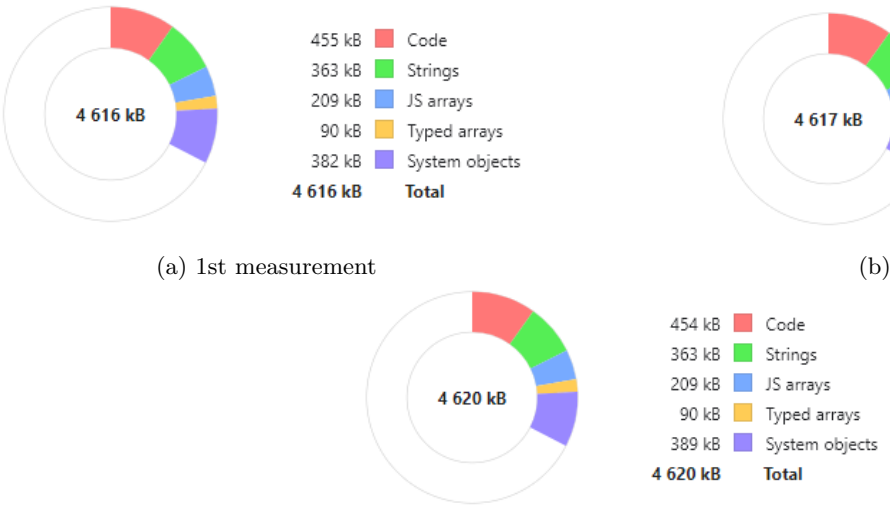
(b) 2nd measurement

(c) 3rd measurement

Figure 6.2: Memory allocated for small dataset.

6.2.2 Medium Dataset

In figure 6.3, we can see the detailed results for the memory allocated by the application for the smaller dataset, in which the average memory allocated was `4617.66 kB`.



(a) 1st measurement

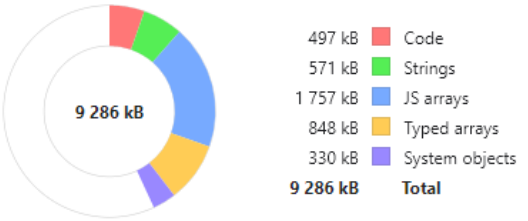
(b) 2nd measurement

(c) 3rd measurement

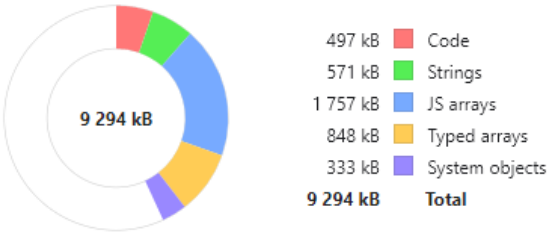
Figure 6.3: Memory allocated for medium dataset.

6.2.3 Big Dataset

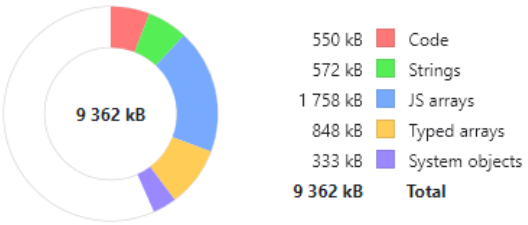
In figure 6.4, we can see the detailed results for the memory allocated by the application for the smaller dataset, in which the average memory allocated was `9314 kB`.



(a) 1st measurement



(b) 2nd measurement



(c) 3rd measurement

Figure 6.4: Memory allocated for big dataset.

Analyzing the results presented in figures 6.2, 6.3 and 6.4, and comparing the results for each measurement we can see the values are very similiar showing almost none leaked memory.

As we can see the amount of memory allocated to JS arrays increases the bigger the dataset, being this the Json data given to the module to build the views.

Overall the total value for the memory allocated is very little in all 3 cases.

7

Final Remarks

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.

As so, the main purpose of this project was to develop a library that uses the force direct layout for phylogenetic trees visualization. This library, the visualization module, responsible for building the force directed layout, was developed independently from the other modules of this application, as this is available to be integrated in similar applications such as the PHYLOVIZ online platform.

This application provides a friendly user interface that allows to navigate through the layout and easily read the information.

The visualization module allows you to:
- Add or remove labels to the nodes and links;
- Modify node and link size or graphic forces;
- Collapse and expand specific regions of the graph;
- Search tree for specific node;
- Produce Pie-chart graphics and link to tree;
- Statistic Analysis;
- Save and reload visualization;
- Visualize and filter profile and isolate data;

In this list, we can see an improvement to the previous Force direct layout solution developed for Phyloviz, adding features like collapse and expand or save state, that weren't available before.

In the initial planning, we did not take into account the amount of research time needed for this project, this includes biology terms and drawing libraries. This resulted in the delay of the overall project's schedule.

In the future, we hope this solution can be incorporated in various platforms and continuously be improved, such as allowing to visualize more than one layout at the same time, allowing to compare two different datasets, create more filters to read information, improve the features already implemented and overall improve the user's experience.

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.
10. Loup Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.

Appendices

A

Parser output example


```
{
  nodes: [
    {
      isNodeLeaf: true,
      key: "A",
      profile: [
        "A",
        "10",
        "6",
        "6",
        "12",
        "13",
      ],
      isolates: [
        "A",
        "United-Kingdom",
        "2012",
      ],
    },
    {
      isNodeLeaf: true,
      key: "B",
      profile: [
        "B",
        "5",
        "4",
        "4",
        "2",
        "15",
      ],
      isolates: [
        "B",
        "United-Kingdom",
        "2010",
      ],
```

```
},
{
  isNodeLeaf: true,
  key: "C",
  profile: [
    "C",
    "5",
    "3",
    "4",
    "6",
    "2",
  ],
  isolates: [
    "C",
    "China",
    "1997",
  ],
},
{
  isNodeLeaf: true,
  key: "D",
  profile: [
    "D",
    "2",
    "2",
    "4",
    "8",
    "7",
  ],
  isolates: [
    "D",
    "China",
    "1996",
  ],
},
{
  isNodeLeaf: false,
  key: "E",
  profile: [
    "E",
    "2",
    "2",
    "1",
    "1",
    "12",
  ],
  isolates: [
```

```
        "E",
        "China",
        "2010",
      ],
    },
    {
      isNodeLeaf: false,
      key: "F",
      profile: [
        "F",
        "1",
        "3",
        "1",
        "1",
        "1",
      ],
      isolates: [
        "F",
        "United-Kingdom",
        "2012",
      ],
    },
  ],
  links: [
    {
      source: "F",
      target: "A",
      value: 0.1,
    },
    {
      source: "F",
      target: "B",
      value: 0.2,
    },
    {
      source: "E",
      target: "C",
      value: 0.3,
    },
    {
      source: "E",
      target: "D",
      value: 0.4,
    },
    {
      source: "F",
      target: "E",
```

```
      value: 0.5,
    },
  ],
  dataset_name: [
    "Demo",
  ],
  data_type: [
    "ST",
  ],
  schemeGenes: [
    "ST",
    "Gene_1",
    "Gene_2",
    "Gene_3",
    "Gene_4",
    "Gene_5",
  ],
  metadata: [
    "ST",
    "Country",
    "Year",
  ],
  subsetProfiles: [
    {
      profile: [
        "A",
        "10",
        "6",
        "6",
        "12",
        "13",
      ],
    },
    {
      profile: [
        "B",
        "5",
        "4",
        "4",
        "2",
        "15",
      ],
    },
    {
      profile: [
        "C",
        "5",
```

```
        "3",
        "4",
        "6",
        "2",
      ],
    },
    {
      profile: [
        "D",
        "2",
        "2",
        "4",
        "8",
        "7",
      ],
    },
    {
      profile: [
        "E",
        "2",
        "2",
        "1",
        "1",
        "12",
      ],
    },
    {
      profile: [
        "F",
        "1",
        "3",
        "1",
        "1",
        "1",
      ],
    },
  ],
  isolatesData: [
    {
      isolate: [
        "A",
        "China",
        "1983",
      ],
    },
    {
      isolate: [
```

```
      "A",
      "United-Kingdom",
      "2012",
    ],
  },
  {
    isolate: [
      "B",
      "United-Kingdom",
      "2010",
    ],
  },
  {
    isolate: [
      "C",
      "China",
      "1997",
    ],
  },
  {
    isolate: [
      "D",
      "China",
      "1996",
    ],
  },
  {
    isolate: [
      "E",
      "China",
      "2010",
    ],
  },
  {
    isolate: [
      "E",
      "China",
      "2010",
    ],
  },
  {
    isolate: [
      "E",
      "China",
      "2010",
    ],
  },
```

```
    {
      isolate: [
        "F",
        "United-Kingdom",
        "2012",
      ],
    },
  ],
}
```

B

Save file example


```
{
    "nodes": [
        {
            "isNodeLeaf": true,
            "key": "A",
            "profile": [
                "A",
                "10",
                "6",
                "6",
                "12",
                "13"
            ],
            "isolates": [
                "A",
                "United-Kingdom",
                "2012"
            ],
            "backupSize": 64.69187950045936,
            "size": 64.69187950045936,
            "index": 0,
            "x": 3413.556841065666,
            "y": 2010.246043954464,
            "vy": -0.03172767997546775,
            "vx": 0.013839587119532959
        },
        {
            "isNodeLeaf": true,
            "key": "B",
            "profile": [
                "B",
                "5",
                "4",
                "4",
```

```
            "2",
            "15"
        ],
        "isolates": [
            "B",
            "United-Kingdom",
            "2010"
        ],
        "backupSize": 64.69187950045936,
        "size": 64.69187950045936,
        "index": 1,
        "x": 3276.4887285824625,
        "y": 2405.4956375174543,
        "vy": -0.027398415589229064,
        "vx": 0.02632281659781115
    },
    {
        "isNodeLeaf": true,
        "key": "C",
        "profile": [
            "C",
            "5",
            "3",
            "4",
            "6",
            "2"
        ],
        "isolates": [
            "C",
            "China",
            "1997"
        ],
        "backupSize": 64.69187950045936,
        "size": 64.69187950045936,
        "index": 2,
        "x": 2663.716661411937,
        "y": 1766.1221685517094,
        "vy": -0.008045352848743786,
        "vx": 0.006128968528076933
    },
    {
        "isNodeLeaf": true,
        "key": "D",
        "profile": [
            "D",
            "2",
            "2",
```

```
            "4",
            "8",
            "7"
        ],
        "isolates": [
            "D",
            "China",
            "1996"
        ],
        "backupSize": 64.69187950045936,
        "size": 64.69187950045936,
        "index": 3,
        "x": 2547.049604426486,
        "y": 2165.2250816163864,
        "vy": -0.004360419782782629,
        "vx": 0.018733908837736322
    },
    {
        "isNodeLeaf": false,
        "key": "E",
        "profile": [
            "E",
            "2",
            "2",
            "1",
            "1",
            "12"
        ],
        "isolates": [
            "E",
            "China",
            "2010"
        ],
        "backupSize": 64.69187950045936,
        "size": 64.69187950045936,
        "index": 4,
        "x": 2827.9984907011058,
        "y": 2030.6056694168817,
        "vy": -0.013233787954410747,
        "vx": 0.014482319813966625,
        "fx": null,
        "fy": null,
        "isCollapsed": true,
        "previousColor": "rgb(110, 64, 170)"
    },
    {
        "isNodeLeaf": false,
```

```
            "key": "F",
            "profile": [
                "F",
                "1",
                "3",
                "1",
                "1",
                "1"
            ],
            "isolates": [
                "F",
                "United-Kingdom",
                "2012"
            ],
            "backupSize": 61.33031284438359,
            "size": 61.33031284438359,
            "index": 5,
            "x": 3127.2868725046233,
            "y": 2132.197947026322,
            "vy": -0.022686260630566335,
            "vx": 0.01769109138457543
        }
    ],
    "links": [
        {
            "value": 0.1,
            "distance": 307.77547662892283
        },
        {
            "value": 0.2,
            "distance": 307.8785907716964
        },
        {
            "value": 0.3,
            "distance": 307.98170379835983
        },
        {
            "value": 0.4,
            "distance": 308.0848157097128
        },
        {
            "value": 0.5,
            "distance": 308.18792650655433
        }
    ],
    "dataset_name": [
        "Demo"
```

```
],
"data_type": [
    "ST"
],
"schemeGenes": [
    "ST",
    "Gene_1",
    "Gene_2",
    "Gene_3",
    "Gene_4",
    "Gene_5"
],
"metadata": [
    "ST",
    "Country",
    "Year"
],
"subsetProfiles": [
    {
        "profile": [
            "A",
            "10",
            "6",
            "6",
            "12",
            "13"
        ]
    },
    {
        "profile": [
            "B",
            "5",
            "4",
            "4",
            "2",
            "15"
        ]
    },
    {
        "profile": [
            "C",
            "5",
            "3",
            "4",
            "6",
            "2"
        ]
```

```
        },
        {
            "profile": [
                "D",
                "2",
                "2",
                "4",
                "8",
                "7"
            ]
        },
        {
            "profile": [
                "E",
                "2",
                "2",
                "1",
                "1",
                "12"
            ]
        },
        {
            "profile": [
                "F",
                "1",
                "3",
                "1",
                "1",
                "1"
            ]
        }
    ],
    "isolatesData": [
        {
            "isolate": [
                "A",
                "China",
                "1983"
            ]
        },
        {
            "isolate": [
                "A",
                "United-Kingdom",
                "2012"
            ]
        },
```

```
{
    "isolate": [
        "B",
        "United-Kingdom",
        "2010"
    ]
},
{
    "isolate": [
        "C",
        "China",
        "1997"
    ]
},
{
    "isolate": [
        "D",
        "China",
        "1996"
    ]
},
{
    "isolate": [
        "E",
        "China",
        "2010"
    ]
},
{
    "isolate": [
        "E",
        "China",
        "2010"
    ]
},
{
    "isolate": [
        "E",
        "China",
        "2010"
    ]
},
{
    "isolate": [
        "F",
        "United-Kingdom",
        "2012"
```

```
            ]
        }
    ],
    "collideForce": 45,
    "strength": -300,
    "nodeLabels": true,
    "linkLabels": false,
    "colorKey": {
        "profile": "Gene_3"
    },
    "nodeLabelsSize": "87",
    "transform": {
        "k": 1.7758143597409195,
        "x": -1439.507834434069,
        "y": -1338.6319074753542
    },
    "isSaved": true
}
```