INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Visualização Radial e Dendrograma de árvores filogenéticas

Adriano Baptista

Francisco Filipe

Inês Sousa

Projeto e Seminário

Licenciatura em Engenharia Informática e de Computadores

Relatório final

Orientadores: Cátia Vaz
Ana Correia

Julho de 2021

Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores

Visualização Radial e Dendrograma de árvores filogenéticas

Adriano Vieira Baptista, n.º 46072
Francisco Patrício Filipe, n.º 46018
Inês Mestre Galvão de Sousa, nº 45997

Orientadores: Cátia Vaz
Ana Correia

Relatório final realizado no âmbito de Projeto e Seminário, do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020/2021

Julho de 2021

Resumo

As tecnologias atuais de sequenciação do **DNA** (ácido desoxirribonucleico) estão a criar uma mudança nas áreas da microbiologia e epidemiologia molecular, devido à capacidade atual de rápida sequenciação de genomas microbianos. A **análise filogenética** de genomas é fundamental para os estudos epidemiológicos e para o estudo genético de populações microbianas, uma vez que pode ser usada para detetar epidemias em ambientes hospitalares ou nas indústrias (*e.g.* indústria alimentar), auxiliar o desenvolvimento de vacinas ou monitorizar antimicrobianos. Os dados resultantes desta análise, podem ser visualizados através de árvores filogenéticas. Existem vários algoritmos de visualização destas árvores, como por exemplo os algoritmos dendrograma e radial.

O **PHYLOViZ** é uma ferramenta que permite a análise, manipulação e visualização de diferentes conjuntos de dados baseados em diferentes métodos de tipagem, com o objetivo de aprofundar os estudos epidemiológicos. Permite filtrar a informação, através dos dados isolados, isto é, dados relativos à amostra extraída. Atualmente este *software* não está integrado apenas numa única plataforma, uma vez que existem duas aplicações separadas, uma aplicação desktop e uma aplicação web.

O principal objetivo deste projeto é desenvolver uma solução que permita a visualização de árvores filogenéticas, no formato de **Dendrograma** e **Radial**, a partir do resultado da aplicação de algoritmos de inferência filogenética, com a possibilidade de realizar diversas pesquisas sobre as árvores, nomeadamente filtrar por uma determinada característica, colapsar e expandir nós, colocar etiquetas nos nós e arcos, alterar a cor e tamanho dos nós, visualizar estatísticas baseadas nos dados isolados, produzir um relatório com o estudo realizado, entre outras. Ao contrário das restantes funcionalidades, a possibilidade de colapsar e expandir nós e a produção de um relatório não existem atualmente em nenhuma das aplicações do PHYLOViZ.

A biblioteca desenvolvida no contexto do projeto irá ser composta por módulos de visualização, bem como um módulo responsável por guardar o estudo e realizar download do mesmo, que serão integrados no contexto da aplicação PHYLOViZ, podendo também serem utilizados noutra aplicação semelhante. Desta forma, estes módulos são genéricos e não estão dependentes de nenhuma aplicação em concreto, podendo facilmente serem integrados numa aplicação, para visualizar árvores no formato dendrograma e radial.

Para demonstrar uma forma de integrar a biblioteca e respetivos módulos de visualização, foi desenvolvida uma aplicação Javascript, multi-plataforma, que utiliza a *framework* **Electron**.

O código fonte do projeto encontra-se presente num repositório GitHub, com o seguinte *url*: `https://github.com/AdrVB/Radial-Dendrogram-Visualization`.

Palavras-chave: análise filogenética; algoritmos de visualização de árvores filogenéticas

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto

As sequências biológicas, nomeadamente o **DNA**, **RNA** (ácido ribonucleico) e as **proteínas**, têm um papel fundamental na biologia molecular, uma vez que definem as atividades celulares que ocorrem em cada organismo. É, assim, fundamental compreender como estas sequências interagem entre si e com o meio envolvente.

O DNA é uma macromolécula fundamental, que contém o código genético da célula. É constituído por duas cadeias de **nucleótidos**, entrelaçadas entre si, formando uma dupla hélice. Esta por sua vez é formada por diferentes bases azotadas, unidas por pontes de hidrogénio. As quatro bases encontradas no DNA são a **adenina** (A), **citosina** (C), **guanina** (G) e **timina** (T). Estas encontram-se ligadas a um açúcar que, por sua vez, se liga a um grupo fosfato, formando assim um nucleótido completo. Os segmentos de DNA que contêm a informação genética são denominados de **genes**. Estes são a unidade molecular fundamental da hereditariedade de um organismo e apresentam variantes com sequências específicas de DNA, denominados **alelos**. A restante sequência de DNA tem importância estrutural ou está envolvida na regulação do uso da informação genética.

A sequenciação do DNA é o processo que permite determinar a ordem precisa dos nucleótidos presentes numa molécula de DNA. Quando se obtém uma amostra microbial ou viral, seja de um indivíduo infetado ou do ambiente, uma cultura pura, isto é, de uma única espécie, é denominada de **estirpe isolada**. É desta estirpe que é extraído o DNA para caracterização. A identificação da estirpe de um organismo é, em termos microbiológicos, designada por **tipagem**. As tecnologias NGS - **Next Generation Sequencing** [1], isto é, as tecnologias atuais de sequenciação do DNA estão a criar uma mudança radical na vida dos cientistas, nomeadamente nas áreas da microbiologia e epidemiologia molecular.

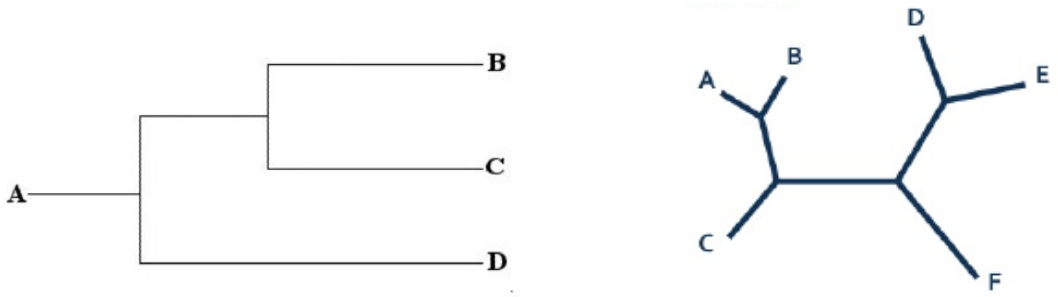
A filogenia [2], ou seja, o estudo da relação evolutiva existente entre diferentes grupos de organismos, como espécies, populações, etc, que provêm de um antecessor comum e a **análise filogenética** de genomas são fundamentais para os estudos epidemiológicos e para o estudo genético de populações microbianas, uma vez que podem ser usadas para detetar epidemias em ambientes hospitalares ou até nas indústrias (*e.g.* indústria alimentar), auxiliar o desenvolvimento de vacinas ou monitorizar antimicrobianos.

Através de algoritmos de inferência filogenética, é possível inferir relações entre os vários isolados, produzindo **árvores de filogenia**. Uma árvore filogenética é composta por nós e por arcos, em que um nó representa uma unidade taxonómica (sequência ou *táxon*[1]) e cada arco liga quaisquer dois nós adjacentes. As árvores filogenéticas dividem-se em dois grupos, **árvores com raiz** e **sem raiz**. As árvores filogenéticas com raiz dividem-se em **folhas, nós, arcos e raiz**. As folhas representam as sequências (*taxa*[2]). Os nós representam a relação existente entre duas sequências, ou seja, o antecessor comum a ambas. Os arcos representam a ligação entre as sequências e o número de alterações genéticas ocorridas até à próxima separação. A raiz representa o antecessor comum a todas as sequências. As árvores filogenéticas sem raiz diferem das anteriores uma vez que não apresentam raiz e o tamanho dos arcos não representa qualquer tipo de informação.

Existem diversas formas de visualizar árvores filogenéticas, podendo-se dividir o tipo de visualização em dois tipos, visualização estática ou dinâmica, alguns exemplos de visualizações estáticas são as visualizações Dendrograma [3] e Radial [4], e um tipo de visualização dinâmica é o Force Direct Layout [5].

Um **dendrograma** é um tipo de visualização estática, que agrupa os organismos em estudo de acordo com o seu nível de similaridade, ou seja, quanto mais semelhantes são as espécies, mas perto se encontram do antecessor comum. Na figura 1.1a, encontra-se um exemplo deste tipo de visualização, em que a raiz da árvore é o nó A.

Um outro tipo de visualização estática é a visualização **radial**, que tal como o dendrograma agrupa os organismos em estudo de acordo com o seu nível de similaridade, com as diferenças dos arcos serem linhas retas e as coordenadas dos nós serem dadas em coordenadas polares, em vez de coordenadas cartesianas. A figura 1.1b apresenta um exemplo de uma visualização radial.



(a) Dendrograma (b) Radial

Figura 1.1: Exemplos de visualizações de árvores filogenéticas.

[1]Táxon - unidade taxonómica de classificação de seres vivos.
[2]Taxa - plural de Táxon.

1.2 Descrição do Problema

Um dos problemas da filogenia é a visualização das árvores, bem como dos dados isolados, isto é, dados relativos à amostra extraída. A integração dos resultados das árvores de inferência filogenética com dados epidemiológicos e também a análise destes, está, normalmente, limitada pelas técnicas de visualização atuais.

Embora existam bibliotecas genéricas de visualização, como por exemplo as bibliotecas D3.js [6] ou a VivaGarph [7], para além de nem todos os tipos de visualização estarem implementados nestas bibliotecas, estas não estão direcionadas especificamente para a visualização de dados filogenéticos e, por isso, a configuração, adaptação e extensão para dados epidemiológicos, requer normalmente alguma experiência em algoritmia e utilização de estruturas de dados mais avançadas.

Desta forma, o principal objetivo deste projeto é desenvolver uma solução modular, em JavaScript, para visualização de árvores filogenéticas, no formato de **Dendrograma** e **Radial**, a partir do resultado da aplicação de algoritmos de inferência filogenética, com a possibilidade de aplicar diversos filtros sobre as árvores, através de dados epidemiológicos. Esta solução utiliza a biblioteca genérica de visualização D3.js.

As funcionalidades implementadas nesta biblioteca consistem na possibilidade de colapsar e expandir nós das árvores, colocar etiquetas nos nós e arcos, de modo a identificar os mesmos, integrar dados complementares, através dos dados isolados, de forma a filtrar os mesmos por uma determinada característica (*e.g* filtrar por país, sexo, idade, etc), guardar o estudo atual de processamento da árvore, bem como a produção de um relatório para ser realizado download do estudo. O código fonte do projeto encontra-se num repositório GitHub, com o *url* `https://github.com/AdrVB/Radial-Dendrogram-Visualization`.

O **PHYLOViZ** [8] é um *software* open source, para manipulação e visualização de diferentes conjuntos de dados baseados em diferentes métodos de tipagem, com o objetivo de aprofundar os estudos epidemiológicos. Permite filtrar a informação, através dos dados isolados.

Atualmente, existem duas aplicações distintas do PHYLOViZ, o PHYLOViZ desktop e o PHYLOViZ online [9, 10]. Na aplicação desktop, estão implementados os algoritmos Dendrograma, Radial e Force Direct Layout, na linguagem Java. No PHYLOViZ online apenas está implementado o algoritmo Force Direct Layout, em Node.js. Em ambas as aplicações, estão implementadas as funcionalidades de integração de dados complementares com a produção de gráficos de setores ou gráficos de barras nos nós, contendo estatísticas, bem como colocação de etiquetas nos nós e arcos da árvore. É também possível guardar o estado atual da árvore, para estudos futuros. O que atualmente não está implementado em nenhuma das aplicações é a opção de colapsar e expandir nós da árvore. Esta funcionalidade é essencial para visualizar árvores de maiores dimensões, isto é, com um número elevado de nós e arcos, uma vez que a possibilidade de colapsar determinados nós permite uma melhor visualização da árvore. Outra funcionalidade que não existe no PHYLOViZ, é a produção de um relatório contendo a árvore e estatísticas aplicadas à mesma, de modo a ser realizado download do estudo.

Assim, as aplicações atuais do PHYLOViZ não são constituídas por módulos agnósticos de onde irá ser executada a aplicação, visto que dependem de um ambiente de execução específico (desktop ou browser), não existindo interoperabilidade entre as

duas versões. No entanto, está a ser desenvolvida uma nova aplicação PHYLOViZ, que irá ser constituída por diferentes módulos e estará disponível como aplicação web e como aplicação desktop, utilizando, para tal, uma *framework* que permite o desenvolvimento de aplicações multi-plataforma (Electron [11]).

A biblioteca desenvolvida no contexto deste projeto, irá ser composta por módulos de visualização, bem como por um módulo responsável por guardar o estudo e realizar download do mesmo, que serão utilizados no contexto da nova aplicação do PHYLOViZ [12], para visualização de árvores filogenéticas nos formatos de dendrograma e radial, podendo também ser utilizados no contexto de uma aplicação semelhante, como por exemplo as aplicações GrapeTree [13] ou Phylo.io [14]. Desta forma, os módulos de visualização são genéricos, isto é, não dependem de nenhuma aplicação, mas podem facilmente ser integrados numa aplicação já existente.

De forma a demonstrar a integração da biblioteca e dos respetivos módulos de visualização, foi desenvolvida, em Javascript, uma aplicação multi-plataforma, utilizando a *framework* Electron, que permite a visualização de árvores filogenéticas nos formatos dendrograma e radial, com integração de dados epidemiológicos. Os passos necessários para a utilização da *framework* Electron numa aplicação Javascript foram detalhados, e encontram-se na `wiki` do repositório do projeto, no *url* `https://github.com/AdrVB/Radial-Dendrogram-Visualization/wiki/How-to-use-Electron-in-a-Javascript-application`.

1.3 Estruturação do documento

No próximo capítulo, 2, irão ser descritos os requisitos do projeto, no capítulo 3 serão abordadas bibliotecas JavaScript que suportam o desenho de árvores, bem como a que será utilizada no projeto. Irá também ser apresentado um estudo comparativo realizado entre diferentes bibliotecas, de forma a escolher a que será utilizada como base para a biblioteca desenvolvida no contexto do projeto. No capítulo 4, irá ser descrita a arquitetura da biblioteca desenvolvida, nomeadamente do ponto de vista dos módulos que a constituem, assim como o formato de dados utilizados, bem como a arquitetura da aplicação Electron, que integra a biblioteca desenvolvida. A implementação dos dois algoritmos de visualização e restantes funcionalidades serão apresentadas no capítulo 5. No capítulo 6, será apresentada uma avaliação experimental aos algoritmos de visualização desenvolvidos. E por fim, no capítulo 7, serão abordadas as conclusões do projeto.

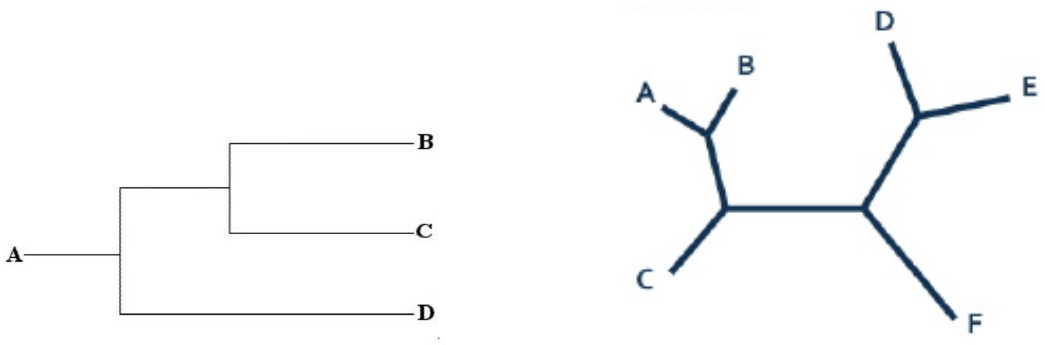
Capítulo 2

Requisitos

Neste capítulo, irão ser abordados os requisitos funcionais definidos para o projeto, bem como o requisito não funcional do mesmo.

2.1 Requisitos Funcionais

Os requisitos funcionais do projeto são os seguintes:
1. Implementação de dois algoritmos de visualização de árvores filogenéticas (**radial** e **dendrograma**). Estes algoritmos recebem como *input* uma árvore filogenética (em formato de newick ou nexus), que é o resultado da aplicação de um algoritmo de inferência filogenética. Para além da árvore filogenética, são também recebidos perfis alélicos (os mesmos perfis utilizados pelo algoritmo de inferência filogenética para produzir a árvore filogenética), assim como dados isolados. A figura 2.1 apresenta um exemplo de cada uma das visualizações a serem implementadas, em que na figura 2.1a podemos observar o Dendrograma e na figura 2.1b podemos observar o Radial.



(a) Dendrograma (b) Radial

Figura 2.1: Exemplo das visualizações Dendrograma e Radial.

2. Possibilidade de colocação de etiquetas nos nós e nos arcos das árvores. Nos nós as etiquetas identificam os perfis alélicos, e nos arcos representam o tamanho dos mesmos. As figuras 2.2 e 2.3 apresentam exemplos desta funcionalidade, em que na figura 2.2 podemos observar a visualização dendrograma, com etiquetas tanto nos nós como nos

arcos, e na figura 2.3 podemos observar a visualização radial, também com etiquetas nos nós e arcos.

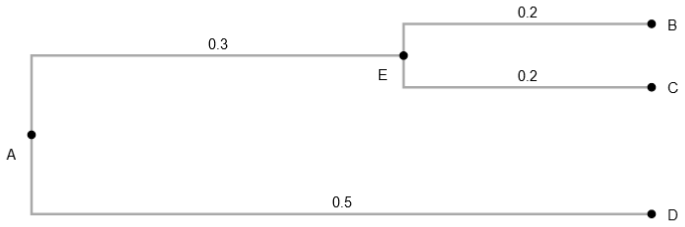


Figura 2.2: Exemplo da funcionalidade de colocação de etiquetas nos nós e arcos no Dendrograma.

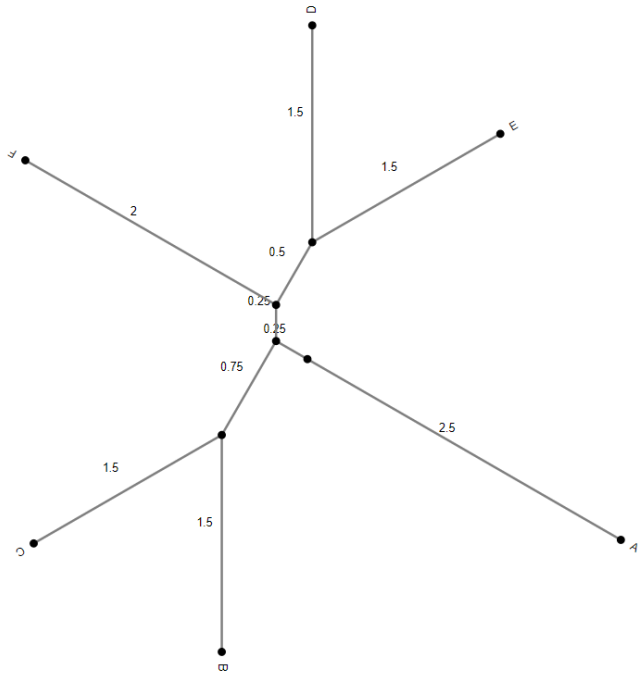
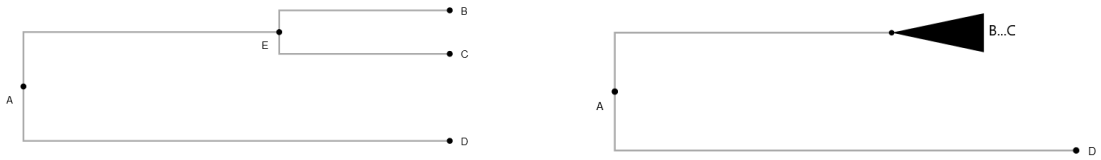


Figura 2.3: Exemplo da funcionalidade de colocação de etiquetas nos nós e arcos no Radial.

3. Possibilidade de colapsar e expandir sub-árvores, quando um nó é pressionado. A sub-árvore colapsada é representada por um triângulo, com tamanho proporcional ao número de nós que integram essa sub-árvore. As figuras 2.4 e 2.5 apresentam exemplos deste requisito, em que na figura 2.4 podemos observar o dendrograma, antes e depois

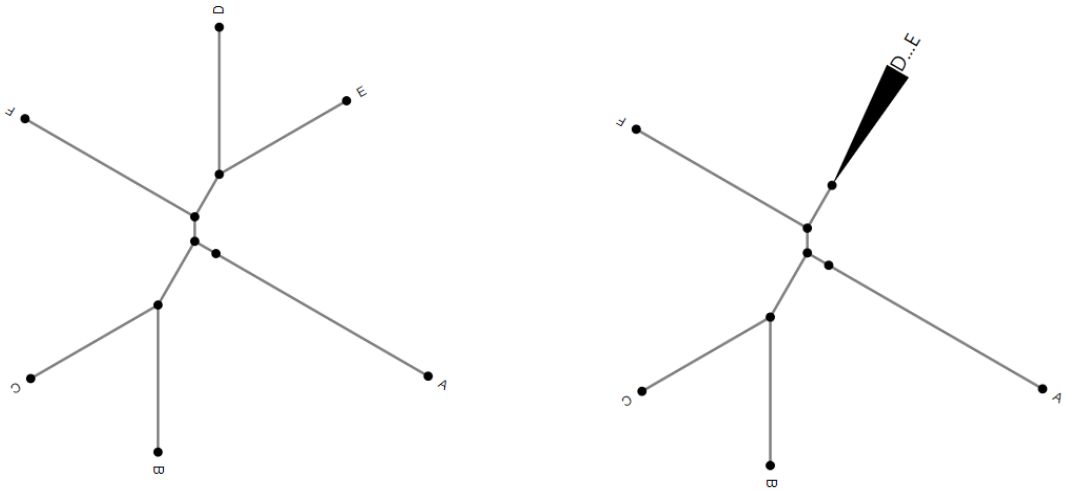
de se colapsar uma das sub-árvores e na figura 2.5, podemos observar o radial, antes e depois de se colapsar uma sub-árvore.



(a) Dendrograma antes de se colapsar uma das sub-árvores

(b) Dendrograma depois de colapsar uma das sub-árvores

Figura 2.4: Exemplo da funcionalidade de colapsar e expandir nós da árvore no Dendrograma.



(a) Radial antes de se colapsar uma das sub-árvores

(b) Radial depois de colapsar uma das sub-árvores

Figura 2.5: Exemplo da funcionalidade de colapsar e expandir nós da árvore no Radial.

4. Integração de dados complementares (através dos dados isolados), com a produção de gráficos de barras nos nós, contendo estatísticas resultantes da aplicação de diferentes filtros. Será possível alterar as cores dos gráficos de barras.

As figuras 2.6 e 2.7 apresentam exemplos deste requisito, onde podemos observar na figura 2.6, um dendrograma, onde foram colocados gráficos de barras nos nós folha, como resultado da aplicação dos filtros ano e país, através da informação presente nos dados isolados. A figura 2.7 representa o radial, onde foram aplicados os mesmos filtros que na figura 2.6.

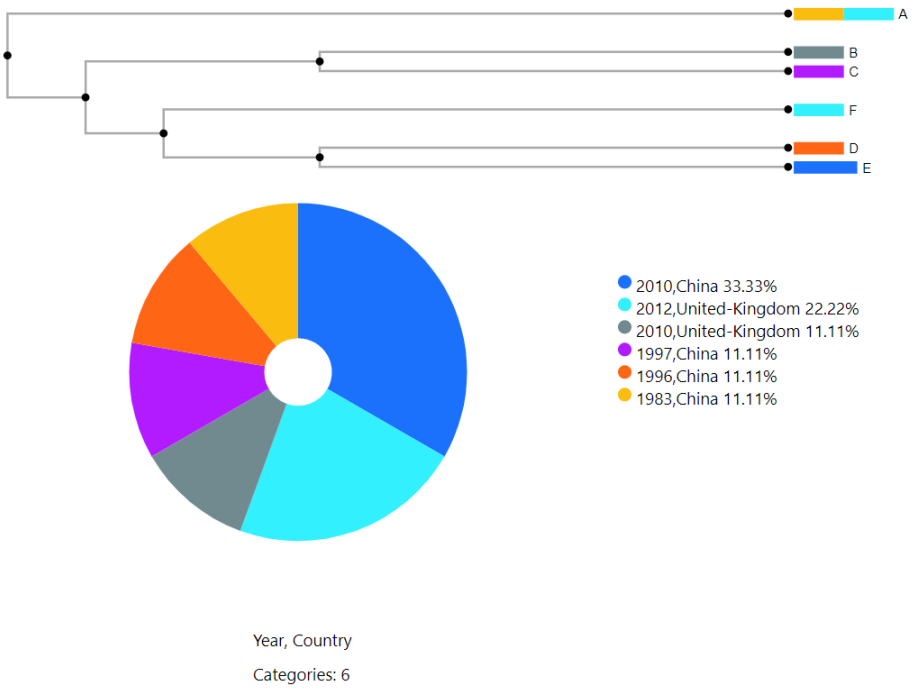


Figura 2.6: Exemplo da funcionalidade de colocação de filtros no Dendrograma.

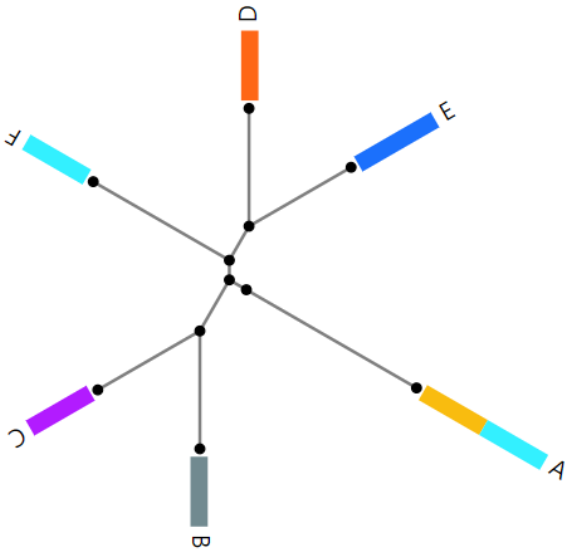


Figura 2.7: Exemplo da funcionalidade de colocação de filtros no Radial.

5. Produção de gráficos de setores com estatísticas, através da informação dos dados isolados. Estes gráficos são constituídos por diferentes combinações (ou categorias), resultantes dos diferentes filtros aplicados. É possível alterar as cores das combinações.

A figura 2.8 apresenta um exemplo de um gráfico de setores com estatísticas, em que é possível observar as diferentes combinações resultantes dos filtros aplicados. Neste exemplo foi filtrado pelo ano e pelo país, e, desta forma, resultaram estas seis combinações.

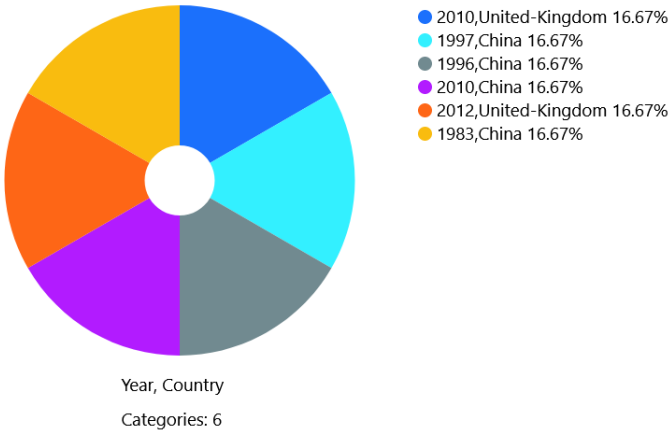


Figura 2.8: Exemplo da funcionalidade de produção de gráfico de setores com estatísticas.

6. Guardar o estado do processamento da árvore para futuros estudos (visualização da árvore filogenética e filtros aplicados à mesma).

7. Produção de um relatório, contendo a árvore filogenética e estatísticas, de forma a ser possível realizar *download* de um PDF, contendo o estudo realizado.

2.2 Requisito não funcional

O requisito não funcional relaciona-se com o facto de ser possível escalar a aplicação para um número elevado de nós e arcos (entre 15 a 20 mil nós e arcos).

Capítulo 3

Tecnologias

Neste capítulo irão ser abordadas bibliotecas JavaScript mais conhecidas para visualização de árvores, que podem ser adaptadas/extendidas para dados epidemiológicos. Irá também ser apresentado um estudo comparativo realizado entre diferentes bibliotecas, de forma a escolher a biblioteca a ser utilizada como base no projeto.

3.1 Bibliotecas Javascript para desenho de árvores

Na implementação dos algoritmos de visualização, é necessário utilizar uma biblioteca para desenhar as árvores filogenéticas. Existem diversas bibliotecas gráficas, em JavaScript, para visualização de gráficos e, em particular, árvores. Alguns exemplos de bibliotecas mais utilizadas são a D3.js [6, 15], VivaGraph [7], Cytoscape [16] e sigmajs [17].

Os principais requisitos pretendidos na biblioteca a ser utilizada como base no projeto são a existência de funções já implementadas, para desenhar o tipo de visualizações pretendidas, ou, caso as mesmas não estejam implementadas, a facilidade de poder implementar novas funções para o efeito. O tipo de tecnologia utilizada para desenhar as árvores é também um fator importante, uma vez que certas bibliotecas utilizam Canvas [18], enquanto que outras utilizam SVG [19]. A utilização da tecnologia SVG pode ser vantajosa, nomeadamente, para a implementação do requisito de produção de um relatório contendo a árvore, uma vez que produz elementos vetoriais, ao contrário da Canvas.

É também importante que a biblioteca a ser utilizada permita costumizar as funções de *render* tanto dos nós e dos arcos, bem como definir as coordenadas dos diferentes componentes que constituem a árvore, de modo a ser possível guardar o estado atual da árvore, para a realização de estudos futuros. Pretende-se também construir gráficos de setores e gráficos de barras, sendo por isso um fator importante a biblioteca a utilizar permitir construir este tipo de gráficos.

Por fim, um dos requisitos fundamentais numa biblioteca é o tempo de carregamento dos gráficos, uma vez que se pretende-se minimizar o tempo de carregamento das árvores.

3.2 Estudo Comparativo

De forma a escolher a biblioteca a ser utilizada como base no projeto, foi realizado um estudo comparativo entre as bibliotecas D3.js, VivaGraph, sigmajs e Cytoscape, em que foi testado o tempo de carregamento, no browser, de três árvores, no formato de dendrograma, em diferentes casos de estudo. Os casos de estudo relacionam-se com os requisitos abordados na secção anterior, que correspondem a requisitos funcionais do projeto.

Cada árvore é constituída por diferentes pares de nós/arcos, sendo a primeira árvore constituída por 5000 nós e arcos, a segunda por 10000 e a terceira por 20000. Por uma questão de simplicidade, na apresentação dos resultados dos testes, a primeira árvore tem a denominação de árvore pequena, a segunda de árvore média e a terceira de árvore grande. Em cada caso de estudo realizado, foram recolhidos vários valores do tempo de carregamento das árvores, sendo depois realizada a média aritmética desses resultados.

O estudo foi realizado numa máquina com as seguintes características:

- Sistema Operativo: Windows 10
- CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- Memória: 16 GB

No primeiro caso de estudo, foi comparado o tempo de carregamento de uma árvore simples, ou seja, apenas constituída por nós e arcos, utilizando as funções já existentes em cada uma das bibliotecas para as visualizações dendrograma e radial, ou, caso algum dos algoritmos não esteja implementado em alguma das bibliotecas, o mesmo foi implementado de forma a serem apresentados apenas os nós e arcos da árvore.

As visualizações utilizadas neste cenário de teste não correspondem na totalidade às visualizações pretendidas no projeto, tendo sido necessário modificar as mesmas. No capítulo 5, nomeadamente na secção 5.1, serão descritas as alterações realizadas a estas funções, de modo a construir as visualizações pretendidas no projeto.

A tabela 3.1 apresenta a média dos tempos de carregamento das diferentes árvores neste caso de estudo.

Biblioteca	**Árvore pequena**	**Árvore média**	**Árvore grande**
sigmajs	328,34	562,65	989,86
D3.js	221,77	332,80	631,16
VivaGraph	1035,37	3808,23	19561,57
Cytoscape	1455,71	2101,25	3441,16

Tabela 3.1: Média dos tempos de carregamento das árvores simples (em milissegundos).

No segundo caso de estudo, foi comparado o tempo de carregamento de uma árvore com nós e arcos e com etiquetas em cada um destes elementos, em que as etiquetas

nos nós representam o identificador dos mesmos, e as etiquetas dos arcos representam o tamanho destes.

A tabela 3.2 apresenta a média dos tempos de carregamento das árvores neste caso de estudo.

Biblioteca	**Árvore pequena**	**Árvore média**	**Árvore grande**
sigmajs	461,55	772,21	1331,71
D3.js	251,66	540,80	977,45
VivaGraph	2029,74	6815,12	31782,84
Cytoscape	2312,39	3987,39	5825,32

Tabela 3.2: Média dos tempos de carregamento das árvores com etiquetas nos nós e arcos (em milissegundos).

No terceiro caso de estudo foi comparado o tempo de carregamento de uma árvore que utiliza funções de *render* diferentes, de modo a alterar a aparência dos seus componentes, como por exemplo, alterar as cores ou dimensões dos nós e arcos. Este cenário de teste tem como objetivo perceber se o tempo obtido no desenho das árvores que utilizam funções de *render* diferentes é muito superior ao tempo de desenho das árvores simples, uma vez que na implementação do projeto, as árvores são construídas com funções de *render* personalizadas, isto é, não são utilizadas as funções disponibilizadas pela biblioteca gráfica utilizada.

A tabela 3.3 apresenta a média dos tempos de carregamento das árvores neste caso de estudo.

Biblioteca	**Árvore pequena**	**Árvore média**	**Árvore grande**
sigmajs	539,19	992,45	1560,83
D3.js	253,43	590,21	1100,32
VivaGraph	1469,57	6173,76	27969,63
Cytoscape	2467,61	3851,64	6294,56

Tabela 3.3: Média dos tempos de carregamento das árvores com uma função de *render* diferente nos nós e arcos (em milissegundos).

Por fim, no quarto caso de estudo, foi comparado o tempo de carregamento de uma árvore com a funcionalidade de colapsar e expandir partes da árvore ao pressionar sobre os nós, de forma a ocultar a sub-árvore dos mesmos.

De salientar que esta funcionalidade foi implementada, nesta fase de testes, de forma diferente do que se pretende no projeto, isto é, como requisito do projeto pretende-se que quando se pressiona sobre um nó, a sua sub-árvore seja ocultada, e quando se voltar a pressionar sobre o mesmo nó, essa sub-árvore fique visível.

Contudo, para a realização destes testes, à exceção da biblioteca D3.js, foi apenas implementado a remoção da sub-árvore quando se pressiona sobre o nó, não sendo possível voltar a expandir a mesma.

A tabela 3.4 apresenta a média dos tempos de carregamento das árvores neste caso de estudo. Para a biblioteca sigmajs, devido à falta de documentação não foi possível testar esta funcionalidade sem atrasar em demasia a execução do projeto.

Biblioteca	**Árvore pequena**	**Árvore média**	**Árvore grande**
sigmajs	-	-	-
D3.js	303,10	570,45	1148,02
VivaGraph	1071,15	4577,19	24060,22
Cytoscape	2445,89	3972,83	6180,51

Tabela 3.4: Média dos tempos de carregamento das árvores com a opção de colapsar e expandir nós (em milissegundos).

Após a análise dos resultados obtidos, chegou-se à conclusão de que a biblioteca D3.js, é a que melhor se adequa ao projeto, uma vez que é a biblioteca com menor tempo de carregamento das árvores, no browser, em todos os casos de estudo realizados. Para além disto, é a biblioteca que apresenta um maior número de exemplos *online*, inclusive na própria documentação oficial.

A biblioteca sigmajs, apesar de ter sido a biblioteca que apresentou valores mais próximos da D3.js, não foi escolhida para o projeto, devido à falta de documentação.

As bibliotecas VivaGraph e Cytoscape apresentaram resultados signitivamente superiores à biblioteca D3.js, e por esse motivo, não foram escolhidas para o projeto.

3.3 Biblioteca D3.js

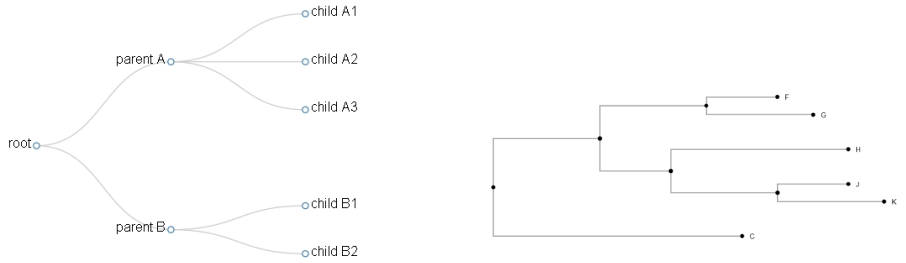
A D3.js [20] é uma biblioteca JavaScript para produção de visualizações dinâmicas no browser. Utiliza as tecnologias SVG, HTML5 e CSS.

Já se encontra implementada, nesta biblioteca, uma função que permite desenhar dendrogramas, no entanto, ao contrário do que se pretende, a mesma não tem em consideração o tamanho de cada arco, ou seja, ao utilizar esta função para desenhar dendrogramas, os nós ficam alinhados por profundidade, tal como podemos observar na figura 3.1a. Desta forma, é necessário a criação de uma função, de forma a implementar o comportamento desejado, isto é, a posição de cada nó depende do tamanho do arco correspondente, tal como podemos observar na figura 3.1b. No capítulo 5, nomeadamente na secção 5.1, é apresentada a implementação desta função.

Ainda relacionado com este tipo de visualização, esta biblioteca, desenha, por omissão, os arcos com formato curvo, como se pode observar na figura 3.1a. Assim, esse comportamento terá de ser modificado, para que os arcos tenham o formato retangular, como se pretende.

Relativamente à visualização radial, não existe uma função já implementada na biblioteca para desenhar este tipo de visualização. Assim sendo, a mesma terá de ser

implementada. No capítulo 5, nomeadamente na secção 5.1, será abordada a implementação desta função. Contudo, existe um outro tipo de visualização radial, denominada de radial circular, que pode ser observada na figura 3.2.



(a) Dendrograma produzido pela D3.js

(b) Dendrograma pretendido no projeto

Figura 3.1: Comparação do dendrograma produzido pela D3.js e o pretendido no projeto.

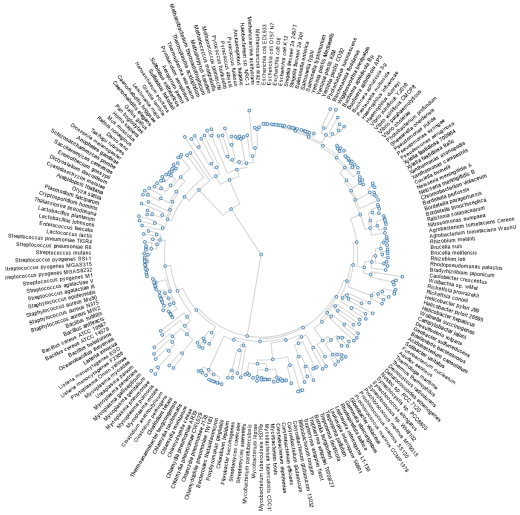


Figura 3.2: Visualização circular radial produzida pela biblioteca D3.js

Capítulo 4

Arquitetura

Neste capítulo irão ser apresentados, primeiramente, os módulos que constituem a biblioteca desenvolvida, sendo depois descrito, com mais detalhe, cada um desses módulos, indicando o formato de dados que utiliza. De seguida, vão ser apresentados os módulos que constituem a aplicação Electron, que integra os módulos de visualização. Os exemplos apresentados ao longo deste capítulo, bem como no capítulo seguinte, são relativos aos mesmos dados, criados de forma abstrata, de modo a manter consistência nos exemplos e a tornar a compreensão dos mesmos mais clara.

4.1 Descrição

A figura 4.1 apresenta o diagrama referente à biblioteca Javascript, e os respetivos módulos de visualização de árvores filogenéticas.

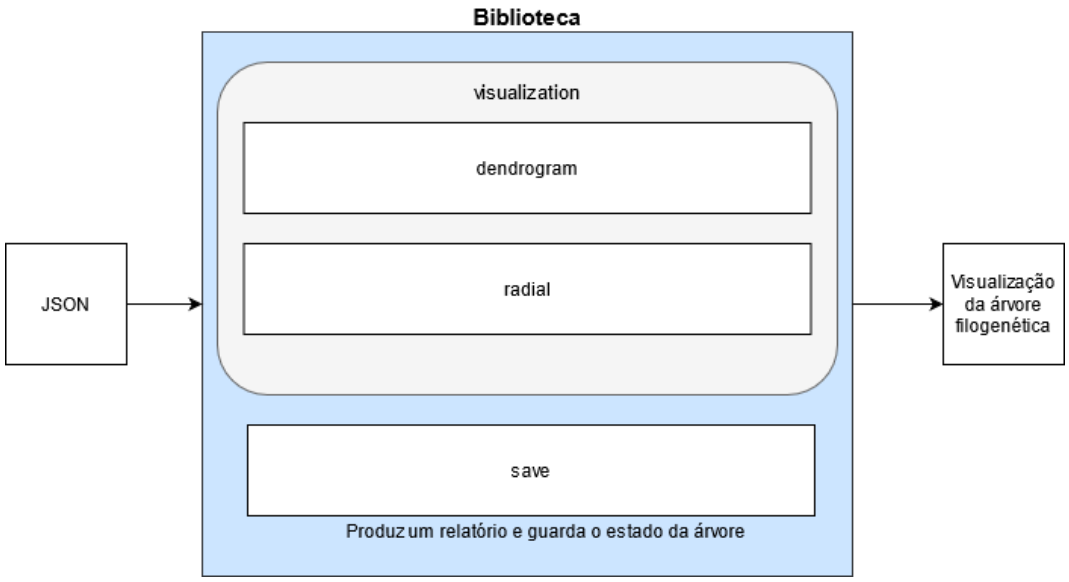


Figura 4.1: Diagrama de arquitetura da biblioteca de visualização.

Esta biblioteca é constituída por dois módulos de visualização, o `dendrogram` e o `radial`. Estes são responsáveis por construir e desenhar as visualizações, bem como a aplicação de filtros sobre as árvores filogenéticas. Existe ainda o módulo `save`, que é responsável por guardar o estudo das árvores, num objeto JSON, bem como realizar download de um PDF com a árvore filogenética.

A figura 4.2 apresenta o diagrama da aplicação Electron, que integra a biblioteca de visualização.

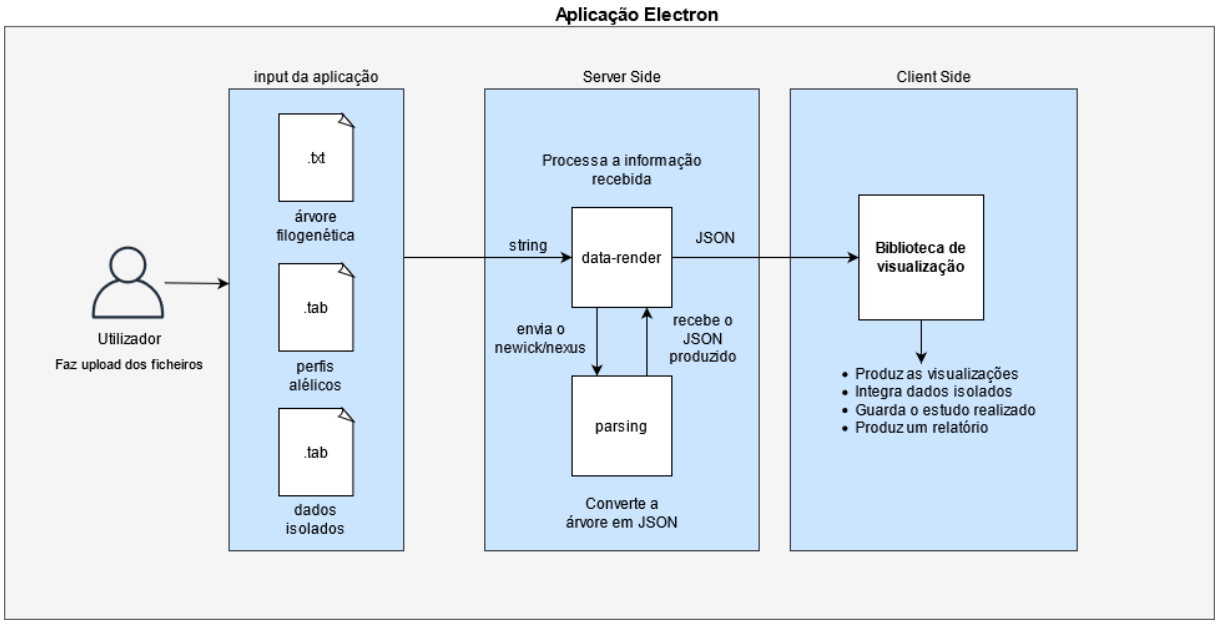


Figura 4.2: Diagrama de arquitetura da aplicação Electron.

A aplicação Electron é constituída pelos seguintes módulos:

1. `data-render`
2. `parsing`

A aplicação Electron pode receber três ficheiros como *input*. Um ficheiro contendo a árvore filogenética, no formato de newick ou nexus, com a extensão .txt. Este ficheiro é necessário para a construção das árvores filogenéticas. Opcionalmente podem também ser recebidos mais dois ficheiros, um ficheiro com os perfis alélicos e outro com os dados isolados. Estes ficheiros têm de estar no formato tabular, com a extensão .tab. Este formato é semelhante ao formato csv, com a diferença dos dados estarem separados entre si pelo separador tab, em vez de vírgula. Os perfis alélicos e os dados isolados serão utilizados posteriormente para a construção de tabelas, de forma a poderem ser aplicados filtros às árvores. O conteúdo destes ficheiros é lido no lado do cliente, sendo depois enviado para o módulo `data-render`, de forma a processar os mesmos.

Os módulos `data-render` e `parsing` podem ser executados tanto no lado do servidor como no lado do cliente, dependendo da opção que é utilizada na inicialização da aplicação.

Depois dos ficheiros terem sido processados, a árvore filogenética que foi recebida, em formato de newick ou nexus, é convertida para um objeto JSON[3], através do módulo `parsing`. Este objeto JSON sofre depois algumas transformações no módulo `data-render`, isto é, são adicionados mais alguns campos, de forma a de seguida ser utilizado pelos módulos da biblioteca de visualização, que são responsáveis pela construção das visualizações das árvores, bem como de outras funcionalidades como o colapsar e expandir nós, a colocação de etiquetas na árvore, entre outros.

4.2 Formato dos Dados

Nesta secção serão apresentados os formatos dos dados utilizados pelos módulos de visualização, que constituem a biblioteca desenvolvida, assim como o módulo `save` pertencente à mesma. Serão também apresentados os formatos dos dados utilizados pelos módulos que constituem a aplicação Electron, nomeadamente os módulos `data-render` e `parsing`.

Módulos de visualização

Os módulos de visualização recebem um objeto JSON. Este objeto é do mesmo tipo do objeto retornado pelo módulo `data-render`, que será apresentado nesta secção. A figura 4.3 apresenta um exemplo da visualização dendrograma e a figura 4.4 apresenta um exemplo da visualização radial, produzidas pelos módulos de visualização.

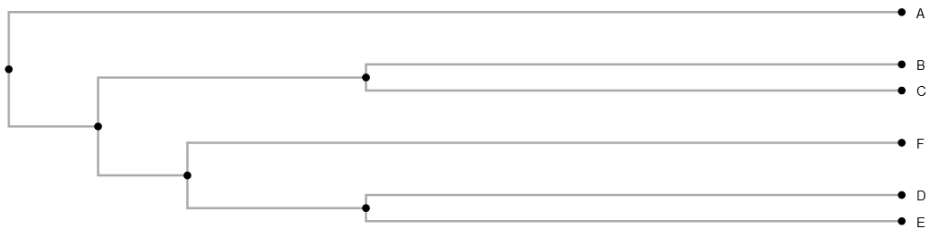


Figura 4.3: Exemplo da visualização dendrograma.

[3]JavaScript Object Notation

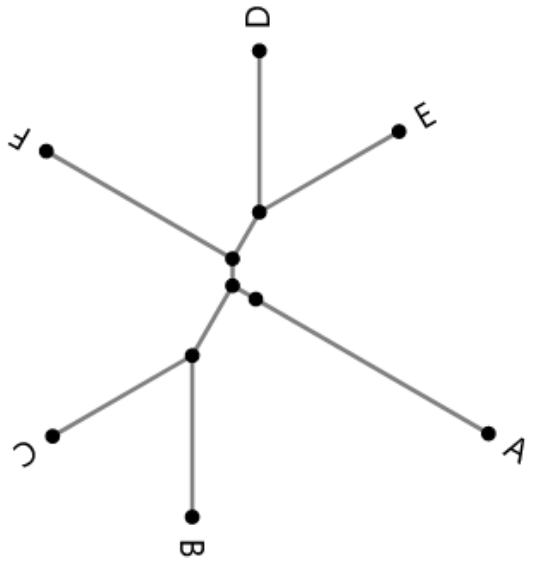


Figura 4.4: Exemplo da visualização radial.

`save`
O formato dos dados produzidos pelo módulo `save`, nomeadamente a produção de um relatório, no formato PDF, com o estudo da árvore filogenética, encontra-se no Anexo A. Os dados resultantes da funcionalidade de guardar o estudo realizado, encontram-se no Anexo B.

Aplicação Electron

`data-render`
Os dados utilizados pelo módulo `data-render` são constituídos, tal como mencionado anteriormente, por um ficheiro com a árvore filogenética e dois ficheiros em formato tabular, com os perfis alélicos e os dados isolados.

Para representar árvores filogenéticas, são utilizados os formatos standard **newick** [21] ou **nexus** [22]. O formato de newick representa as árvores filogenéticas numa única linha, indicando o nome do nó e tamanho do arco, separados por dois pontos. Os diferentes nós são separados entre si através de vírgulas, e as diferentes sub-árvores são separadas por parênteses. O ponto e vírgula, indica a terminação da árvore. A figura 4.5 apresenta um exemplo de uma árvore filogenética em formato de newick. Esta árvore foi obtida, através do algoritmo de inferência filogenética SL (Single-linkage clustering), utilizando a biblioteca de inferência filogenética **phylolib** [23] para o efeito.

Esta árvore é constituída por dez nós, em que quatro deles não têm um identificador, isto é, os nós folha têm de ter, obrigatoriamente, um identificador, no entanto, os restantes nós podem não ter identificadores. Nestes casos, é comum os nós serem representados pelo símbolo '_'.

```
(D:1.8125,(A:2.8125,(B:1.625,C:1.375)_:2.1875)_:2.6875,(E:1.25,F:1.75)_:4.4375)_;
```

Figura 4.5: Árvore filogenética no formato de newick.

O formato de nexus é um pouco diferente do formato de newick, uma vez que é constituído por uma primeira linha com a denominação #NEXUS, seguida de diferentes indicadores opcionais, com os nomes de TAXA, CHARACTERS e TREES. O indicador TAXA contém informação referente às taxas em estudo. O indicador CHARACTERS contém informação relativa às diferentes sequências e o bloco TREES contém as árvores filogenéticas, no formato de newick. A figura 4.6 apresenta um exemplo de uma árvore filogenética no formato de nexus, utilizando o mesmo algoritmo de inferência filogenética do exemplo anterior.

```
#NEXUS

BEGIN TAXA;
    Dimensions NTax=6;
    TaxLabels A B C D E F;
END;

BEGIN CHARACTERS;
    Dimensions NChar=20;
    Format DataType=DNA;
    Matrix
        A   ACATA GAGGG TACCT CTAAG
        B   ACATA GAGGG TACCT CTAAG
        C   ACATA GAGGG TACCT CTAAG
        D   ACATA GAGGG TACCT CTAAG
        E   ACATA GAGGG TACCT CTAAG
        F   ACATA GAGGG TACCT CTAAG
END;

BEGIN TREES;
        Tree result =
        (D:1.8125,(A:2.8125,(B:1.625,C:1.375)_:2.1875)_:2.6875,(E:1.25,F:1.75)_:4.4375)_;
END;
```

Figura 4.6: Árvore filogenética no formato de nexus.

Nos ficheiros com os perfis alélicos e dados isolados, tem sempre de existir uma coluna identificadora do perfil. A figura 4.7 apresenta um exemplo de perfis alélicos, em que a coluna ST é o identificador do perfil. Nas restantes colunas, podemos observar as sequências dos diferentes genes em estudo.

ST	Gene_1	Gene_2	Gene_3	Gene_4	Gene_5
A	10	6	6	12	13
B	5	4	4	2	15
C	5	3	4	6	2
D	2	2	4	8	7
E	2	2	1	1	12
F	1	3	1	1	1

Figura 4.7: Exemplo de perfis alélicos, em formato de tabela.

Os dados isolados têm o mesmo formato que os perfis alélicos, sendo que uma das colunas tem de coincidir, obrigatoriamente, com a coluna identificadora na tabela dos perfis alélicos correspondente. A figura 4.8 apresenta um exemplo de dados isolados, em que podemos observar que a coluna ST identifica os perfis, e a restante informação, país e ano, é referente ao local e data da amostra em estudo, respetivamente.

ST	Country	Year
A	China	1983
D	China	1996
C	China	1997
B	United-Kingdom	2010
E	China	2010
E	China	2010
E	China	2010
A	United-Kingdom	2012
F	United-Kingdom	2012

Figura 4.8: Exemplo de dados isolados, em formato de tabela.

`parsing`
O módulo `parsing` recebe o newick ou nexus, nos formatos descritos anteriormente, e converte-os para um objeto JSON. A figura 4.9 apresenta a representação do resultado da aplicação do módulo `parsing`. O objeto JSON obtido é constituído por dois campos, com os nomes **links** e **nodes**. O primeiro representa as ligações entre os diferentes nós, indicando o `source` e `target` de cada ligação, isto é, o nó onde começa e termina a mesma. O campo `value` representa o tamanho do arco. O campo `nodes` contém o identificador de cada nó, presente no campo `key`.

```
{
    links: [
        { source: 'unnamed_node_0', target: 'B', value: 1.625 },
        { source: 'unnamed_node_0', target: 'C', value: 1.375 },
        { source: 'unnamed_node_1', target: 'A', value: 2.8125 },
        { source: 'unnamed_node_1', target: 'unnamed_node_0', value: 2.1875 },
        { source: 'unnamed_node_2', target: 'E', value: 1.25 },
        { source: 'unnamed_node_2', target: 'F', value: 1.75 },
        { source: 'unnamed_node_3', target: 'D', value: 1.8125 },
        { source: 'unnamed_node_3', target: 'unnamed_node_1', value: 2.6875},
        { source: 'unnamed_node_3', target: 'unnamed_node_2', value: 4.4375},
        { source: null, target: 'unnamed_node_3', value: 0 }
    ],
    nodes: [
        { key: 'B' },
        { key: 'C' },
        { key: 'A' },
        { key: 'unnamed_node_0' },
        { key: 'E' },
        { key: 'F' },
        { key: 'D' },
        { key: 'unnamed_node_1' },
        { key: 'unnamed_node_2' },
        { key: 'unnamed_node_3' }
    ]
}
```

Figura 4.9: JSON produzido pelo módulo parsing, para os dados de exemplo.

O módulo `data-render`, depois de receber o objeto JSON resultante da execução do módulo `parsing`, constrói um novo JSON, com os seguintes campos:

- `links`
- `nodes`
- `schemeGenes`
- `metadata`

O campo `schmeGenes` contém os nomes dos perfis alélicos em estudo e o campo `metadata` contém os nomes dos dados isolados em estudo. A figura 4.10 apresenta um exemplo do objeto JSON produzido pelo módulo `data-render`, e que é posteriormente enviado para os módulos de visualização.

```
{
  "schemeGenes": ["ST", "Gene_1", "Gene_2", "Gene_3", "Gene_4", "Gene_5"],
  "nodes": [
    {"key": "B",
      "profile": ["B", "5", "4", "4", "2", "15"],
      "isolates": [["B", "United-Kingdom", "2010"]]
    },
    {"key": "C",
      "profile": ["C", "5", "3", "4", "6", "2"],
      "isolates": [["C","China","1997"]]
    },
    {"key": "D",
      "profile": ["D", "2", "2", "4", "8", "7"],
      "isolates": [["D", "China", "1996"]]
    },
    {"key": "E",
      "profile": ["E", "2", "2", "1", "1", "12"],
      "isolates": [["E", "China", "2010"]]
    },
    {"key": "F",
      "profile": ["F", "1", "3", "1", "1", "1"],
      "isolates": [["F", "United-Kingdom", "2012"]]
    },
    {"key": "unnamed_node_1"},
    {"key": "unnamed_node_0"},
    {"key": "unnamed_node_2"},
    {"key": "A",
      "profile": ["A", "10", "6", "6", "12", "13"],
      "isolates": [["A", "China", "1983"]]
    },
    {"key": "unnamed_node_3"},
    {"key": "unnamed_node_4"}
  ],
  "links": [
    {"source": "unnamed_node_0", "target": "B","value": 1.5},
    {"source": "unnamed_node_0", "target": "C", "value": 1.5},
    {"source": "unnamed_node_1", "target": "D", "value": 1.5},
    {"source": "unnamed_node_1", "target": "E", "value": 1.5},
    {"source": "unnamed_node_2", "target": "F", "value": 2},
    {"source": "unnamed_node_2", "target": "unnamed_node_1", "value": 0.5},
    {"source": "unnamed_node_3", "target": "unnamed_node_0", "value": 0.75},
    {"source": "unnamed_node_3", "target": "unnamed_node_2", "value": 0.25},
    {"source": "unnamed_node_4", "target": "A", "value": 2.5},
    {"source": "unnamed_node_4", "target": "unnamed_node_3", "value": 0.25},
    {"source": null, "target": "unnamed_node_4", "value": 0}
  ],
  "metadata": ["ST", "Country", "Year"]
}
```


Figura 4.10: JSON produzido pelo módulo data-render, para os dados de exemplo.

4.3 Biblioteca de visualização

`visualization`
Existem dois módulos responsáveis pela produção das visualizações, assim como a implementação das funcionalidades de colapsar e expandir nós da árvore, aplicar filtros às árvores, entre outras. Um dos módulos, é responsável pela visualização dendrograma e o outro é responsável pela visualização radial. Estes utilizam um objeto JSON, com o mesmo formato que o objeto JSON resultante do módulo `data-render`, para a construção das visualizações.

`save`
O módulo `save`, guarda o estudo da árvore num objeto JSON, constituído pelos campos `type`, que indica o tipo de visualização, `data`, que contém, para além da árvore construída, no campo `tree`, o objeto JSON retornado pelo módulo `data-render`, no campo `input`. No campo `graph`, estão presentes as informações relativas aos filtros aplicados à visualização da árvore. Por fim no campo `canvas`, está indicado o valor do *zoom* aplicado à árvore. O anexo B apresenta o objeto JSON resultante do módulo `save`, para os dados de exemplo.

Este módulo produz também um relatório com a árvore filogenética. Este relatório é um ficheiro em formato PDF, com a árvore filogenética, bem como um gráfico de setores com os filtros aplicados à mesma, se tiverem sido integrados os dados isolados.

4.4 Integração da biblioteca na aplicação Electron

Para integrar a biblioteca de visualização desenvolvida, foi necessário criar os módulos que se apresentam de seguida.

4.4.1 data-render

Este módulo é utilizado pela aplicação para processar os dados recebidos. Quando são submetidos os ficheiros, o conteúdo destes é enviado para este módulo, para serem processados, de forma a serem utilizados depois, quer seja para a construção da árvore, como para a construção das tabelas contendo os perfis alélicos e os dados isolados, de modo a serem aplicados diferentes filtros sobre as árvores.

Este módulo comunica diretamente com o módulo `parsing`, de forma a enviar a árvore filogenética para ser convertida num objeto JSON.

4.4.2 parsing

Este módulo é responsável por transformar o newick ou nexus num objeto JSON. Comunica apenas com o módulo anterior, uma vez que recebe a árvore processada no módulo `data-render` e envia o resultado da conversão da mesma para um objeto JSON.

Capítulo 5

Implementação

Neste capítulo, irá ser descrita a implementação de cada módulo constituinte da biblioteca de visualização, bem como os módulos da aplicação Electron.

5.1 Biblioteca de visualização

Para construir as visualizações, foi criada a função `build`. Esta função recebe um objeto JSON, com o formato descrito na figura 4.10 do capítulo anterior.

A biblioteca D3.js, tem implementada uma função, denominada `hierarchy` [24], que obtém um objeto que representa uma árvore filogenética, a partir de um objeto JSON hierárquico. Assim, a função `build`, utiliza esta função `hierarchy`, da biblioteca D3.js, para obter a árvore filogenética a ser posteriormente desenhada.

No entanto, o objeto JSON recebido pela função `build`, não é do tipo hierárquico, uma vez que um objeto deste tipo é constituído por um nó raiz e um array de nós, que representam os filhos desse nó raiz. Assim, antes de ser utilizada a função `hierarchy`, tem de ser utilizada uma outra função da biblioteca D3.js, denominada `stratify`, que transforma um objeto JSON não hierárquico, num objeto hierárquico, isto é, transforma o campo `links`, do objeto JSON recebido pela função `build`, num objeto hierárquico, que é depois utilizado pela função `hierarchy`.

De seguida, e ainda na função `build`, com os dados produzidos pela função `hierarchy`, é construída a visualização, sendo utilizada a função responsável pela construção do dendrograma ou do radial, dependendo da visualização pretendida.

Dendrograma

A função responsável pela construção deste tipo de visualização, é a função `cluster`, presente na biblioteca D3.js. Esta função percorre a árvore duas vezes, começando sempre nos nós-folha. Na primeira vez que a árvore é percorrida, são calculadas as coordenadas (x e y) iniciais de cada nó. Se um nó tiver filhos, a sua coordenada x será igual a:

$$x = meanX(children)$$

Em que `meanX` é uma função que calcula a média dos valores das coordenadas x dos filhos desse nó. A coordenada y é dada por:

$$y = maxY(children)$$

Em que `maxY` é uma função que calcula o valor máximo dos valores da coordenada y dos filhos do nó.
Se um nó não tiver filhos, significa que é um dos nós folha. Assim, a sua coordenada x será igual a:

$$x = separation(node, previousNode)$$

Em que `separation` é uma função que calcula a distância entre dois nós-folha. A coordenada y é sempre igual a 0. A figura 5.1 apresenta o pseudocódigo onde é realizado a primeira pesquisa sobre a árvore.

```
Input: T = (V,E,δ)
Output: Coordinates x, y : V → R for the nodes
Data: r ← root(T), dx ← 1, dy ← 1, separation
    // First walk, computing the initial x & y values.
    node ← root(T)
    postorder(node)
        if (r == null)
            return
        for each child v of r do
            postorder(v)
        end for

        previousNode ← null
        x ← 0
        children ← children(r);
        if children
            x(r) ← meanX(children);
            y(r) ← maxY(children);
        else
            x(r) ← previousNode ? x += separation(node, previousNode) : 0
            y(r) ← 0;
            previousNode = node;
```

Figura 5.1: Primeira pesquisa realizada sobre a árvore para construção do dendrograma.

Na segunda vez que a árvore é percorrida, são calculadas novas coordenadas para os nós, de forma a colocá-los na posição pretendida. A figura 5.2 apresenta o pseudocódigo onde é efetuada a segunda pesquisa sobre a árvore.

```
// Second walk, normalizing x & y to the desired size.
node ← root(T)
postorder(node)
    if (node == null)
        return
    for each child v of r do
        postorder(v)
    end for

    visibility(r) ← true
    x(node) ← (x(node) - x(r)) * dx
    y(node) ← (y(node) - y(r)) * dy

    if !children(node)
        leaves(node) ← Array[ node ];
    else
        leaves(node) ← Array[]
        for each child child of node do
            if !children(child)
                leaves(node).add_last(id(child))
            else
                leaves(node).add_last(...leaves(child))
        end for
```

Figura 5.2: Segunda pesquisa realizada sobre a árvore para construção do dendrograma.

Para ser realizado processamento necessário para a funcionalidade de colapsar e expandir nós, e para não ser realizada uma pesquisa extra sobre a árvore, o que aumentaria o tempo de execução do algoritmo, a função `cluster` foi redefinida, de forma a que na segunda vez que a árvore é percorrida, são calculados os nós folha de cada nó. Desta forma, foi criada a função `clusterTree`, que tem o mesmo comportamento que a função `cluster`, mas efetua a computação descrita.

O custo temporal do algoritmo para construção da visualização dendrograma é de O(2V), sendo V o número de nós da árvore.

Assim, a função `clusterTree` é executada, sendo também executada a função `nodeSize`. A função `nodeSize` pertence à função original da biblioteca D3.js para construção de dendrogramas, e define o espaço da árvore atribuído a cada nó, sendo este constituído por um `array` de dois elementos, sendo o primeiro a largura e o segundo a altura. O uso desta função tem o objetivo de impedir que os nós se sobreponham uns aos outros. Esta função coloca também a raiz da árvore na posição (0,0).

Por fim, a função `build` fica completa quando é executada a função `applyScale`. Esta é responsável por redefinir as coordenadas dos nós da árvore, tendo em consideração o tamanho de cada arco, bem como o valor da escala aplicada à árvore. A figura 5.3 apresenta o pseudocódigo desta função.

```
Input: T = (V,E,δ)
Output: Coordinates x, y : V → R for the nodes
Data: r ← root(T),
        horizontal(value, scalingFactor),
        vertical(value, scalingFactor),
        isAligned

    Q.insert(r)
    while !Q.isEmpty() do
        v ← Q.delete_first()
        for each child w of v do
            Q.insert(w)
            if id(v) equeals id(r)
                continue
            x(v) ← x(v) * value(vertical)
            if parent(v)
                if !isAligned
                    y(v) ← value(v) * value(horizontal) * scalingFactor(horizontal) + y(parent(d))
                else
                    y(v) ← depth(v) * value(horizontal) * scalingFactor(horizontal)
            }
        end for
    end while
```

Figura 5.3: Pseudocódigo da função applyScale.

Radial

Tal como mencionado no capítulo 2, a biblioteca D3.js não tem implementada uma função para a visualização radial. Desta forma, foi implementado um algoritmo, baseado no pseudocódigo presente no livro Biological Networks [4], que podemos observar na figura 5.4.

Input: $T = (V, E, \delta)$
Output: Coordinates $x, y \colon V \to \mathbb{R}$ for the nodes
Data: Queue Q, leafcount$\colon V \to \mathbb{N}^+$ {from a previous postorder traversal}

$r \leftarrow \mathsf{root}(T)$
$Q.\mathsf{insert}(r)$
$\text{rightborder}(r) \leftarrow 0$
$\text{wedgesize}(r) \leftarrow 2\pi$
$x(r) \leftarrow y(r) \leftarrow 0$
while $!Q.\mathsf{isEmpty}()$ **do**
  $v \leftarrow Q.\mathsf{delete\_first}()$
  $\eta \leftarrow \text{rightborder}(v)$
  for each child w of v **do**
    $Q.\mathsf{insert}(w)$
    $\text{rightborder}(w) \leftarrow \eta$
    $\text{wedgesize}(w) \leftarrow \frac{2\pi \cdot \text{leafcount}(w)}{\text{leafcount}(r)}$
    $\alpha \leftarrow \text{rightborder}(w) + \frac{\text{wedgesize}(w)}{2}$
    $x(w) \leftarrow x(v) + \cos(\alpha) \cdot \delta((v, w)); \; y(w) \leftarrow y(v) + \sin(\alpha) \cdot \delta((v, w))$
    $\eta \leftarrow \eta + \text{wedgesize}(w)$
  end for
end while

Figura 5.4: Algoritmo Radial.

Este algoritmo percorre a árvore uma vez, começando pelos nós-folha. Para cada nó é calculada a sua `righborder`, isto é, o ângulo correspondente ao espaço que esse nó irá ocupar na visualização, bem como o `wedesize`, que representa o perímetro do espaço ocupado pelo nó. Para cada filho do nó a ser percorrido, o ângulo correspondente será igual a:

$$wedgesize(w) = \frac{2\pi * leafcount(w)}{leafcount(r)}$$

Em que `w` é o nó a ser percorrido e o `r` é a raiz da árvore. De seguida, é calculado o α, que é igual a:

$$\alpha = rightborder(w) + \frac{wedgesize(w)}{2}$$

As coordenadas de cada nó são dadas em coordenadas polares. Assim, o x é calculado através da fórmula:

$$x(w) = x(v) + cos(\alpha) * \delta((v, w))$$

E o y é calculado através da fórmula:

$$y(w) = y(v) + sin(\alpha) * \delta((v, w))$$

Em certas árvores, os nós e arcos podem sobrepor-se uns aos outros, devido aos ângulos resultantes do cálculo descrito anteriormente. Para tentar evitar este problema, foi implementado um algoritmo de *spread*. Isto implica voltar a percorrer a árvore e

cada sub-árvore de cada nó, o que aumenta o tempo de execução do algoritmo. Assim, foi adicionada a opção de aplicar a função `spread` à árvore. A figura 5.5 apresenta o pseudocódigo do algoritmo de *spread*.

$$m(v) \leftarrow \begin{cases} \delta\left((u,v)\right), & \text{if } v \text{ is a leaf,} \\ \frac{\sum_{(v,w)}(\delta((u,v))+m(w))}{|\{\, w | w \text{ is a child of } v \,\}|}, & \text{otherwise.} \end{cases}$$

Figura 5.5: Algoritmo de spread.

O custo temporal do algoritmo para construção da visualização radial é de O(V), sendo V o número de nós da árvore.

Desenhar a árvore

Uma vez estando construída a árvore, é necessário desenhar a mesma. Para tal, foi criada a função `draw`, que recebe como primeiro argumento uma `string`, que representa o local a desenhar a árvore (container), e como segundo argumento recebe a árvore que foi construída pela função `build`. A função `draw` cria um elemento do tipo svg, adicionando-o ao elemento onde será desenhada a árvore (container). A árvore é um elemento do tipo g, isto é, é um elemento SVG que representa um contentor, de forma a agrupar outros elementos. O atributo `transform`, aplicado a este elemento, permite alterar a posição da árvore, quando existe uma transformação da mesma.

Quando são desenhadas as árvores, é também desenhada uma linha horizontal, de forma a indicar a escala aplicada à árvore, isto é, juntamente com essa linha horizontal, existe uma `string`, que indica o tamanho da linha, de forma a que quando existem transformações na árvore, esse valor reflita a escala atual. Para desenhar a linha horizontal foi criada a função `horizontalScale`. Para adicionar o valor à linha, foi criada a função `applyScaleText`.

Colpsar e expandir nós

A implementação desta funcionalidade consiste em adicionar um evento a todos os nós (do tipo `onclick`), que será ativado quando um nó é pressionado.

Cada nó tem a propriedade `visibility`, que indica se o mesmo se encontra visível ou não. Se o nó estiver visível e for pressionado, é executada a função `collapse`, de forma a colapsar a sub-árvore desse nó, caso contrário, é executada a função `expand`, para construir a sub-árvore desse nó.

A função `collapse` recebe como argumentos o nó que foi pressionado, e os seus filhos (`d.children`, em que d é o nó pressionado). Os nós filhos são percorridos, de forma a serem removidos os nós e arcos correspondentes à sub-árvore do nó pressionado. Este algoritmo realiza um percurso do tipo DFS [25] sobre a árvore, de forma a percorrer todos os nós filhos, até chegar aos nós-folha, onde termina. A propriedade `visibility` é colocada a `false` nos nós que foram colapsados. O custo temporal deste algoritmo é de O(V), sendo V o número de nós presentes na sub-árvore colapsada.

Quando uma sub-árvore de um nó é colapsada, é desenhado um triângulo nesse nó, de forma a representar a sub-árvore que foi removida. O tamanho do triângulo é

proporcional ao número de nós que foram colapsados e é apresentado na escala logarítmica. Este triângulo é um elemento do tipo `polygon` e as coordenadas do mesmo são calculadas através da função `getTriangle`. Para além do triângulo, são ainda colocadas etiquetas, que identificam o intervalo dos nós que foram colapsados, em que os identificadores dos nós são obtidos através do campo `node.leaves`, que é adicionado às propriedades da árvore no momento de construção da mesma. A figura 5.6 apresenta um dendrograma, com uma das suas sub-árvores colapsada, em que podemos observar o triângulo que representa a sub-árvore colapsada e a etiqueta que identifica os nós folha colapsados.

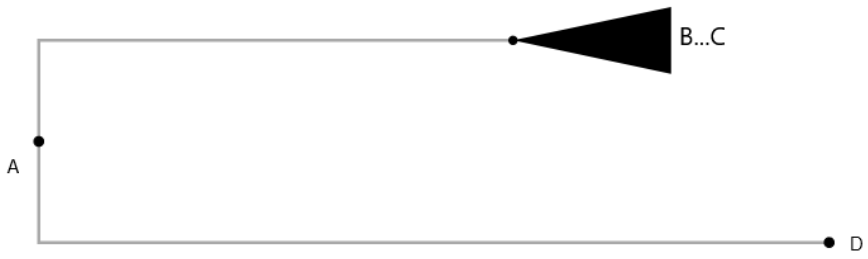


Figura 5.6: Dendrograma com uma sub-árvore colapsada.

O método `expand` executa o processo inverso, isto é, adiciona os nós e arcos que tinham sido previamente removidos na função `collapse`. A propriedade `visibility` de cada nó adicionado é colocada novamente a `true`. Quando uma sub-árvore é expandida, se uma das suas sub-árvores estiver colapsada, isto é, se existirem nós com a propriedade `visibility` a `false`, essas sub-árvores não são desenhadas, sendo novamente desenhados triângulos nas mesmas.

Zoom
Existe na biblioteca D3.js, uma função, denominada `zoom`, que permite adicionar o comportamento de zoom à árvore. Para tal, é necessário indicar os valores de x e y, bem como do atributo scale, tendo sido definida uma escala, com o valor mínimo e máximo do zoom. É também possível filtrar o tipo de comportamento a ser aplicado à árvore, tendo sido selecionados os eventos de `mousedown` e de `wheel`, ou seja, é possível, para além de aumentar e diminuir o zoom da árvore, mover a mesma através do *mouse*.

Escala
Foram implementadas duas escalas, linear e logarítmica. Estas escalas permitem visualizar o tamanho dos nós e arcos de acordo com o zoom aplicado à árvore, em que se estiver aplicada a escala logarítmica, o tamanho da árvore aumenta e diminui a um ritmo logarítmico, o que permite visualizar melhor árvores com uma grande quantidade de nós e arcos.

Etiquetas

Quando a árvore é desenhada, por omissão, são colocadas etiquetas apenas nos nós-filhos. No entanto, é possível também colocar etiquetas nos nós-pai e nos arcos. Para tal, foram criadas as funções `addLeafLabels`, `addInternalLabels` e `addLinkLabels`.

A função `addLeafLabels` seleciona todos os nós-folha, através da função da biblioteca D3.js, `selectAll`, adicionando um elemento do tipo `text`, com o identificador do nó, que se encontra presente no atributo `data.id` de cada nó. Nestes nós é adicionado ainda um `listener` para eventos do tipo `mouseover`, de forma a alterar a cor da etiqueta que identifica o nó, bem como colocar o atributo `font-weight` a *bold*. Estas propriedades são colocadas no nó utilizando CSS. A figura 5.7 apresenta um dendrograma, com etiquetas apenas nos nós-folha, em que é possível observar o comportamento do evento descrito no nó com o identificador E.

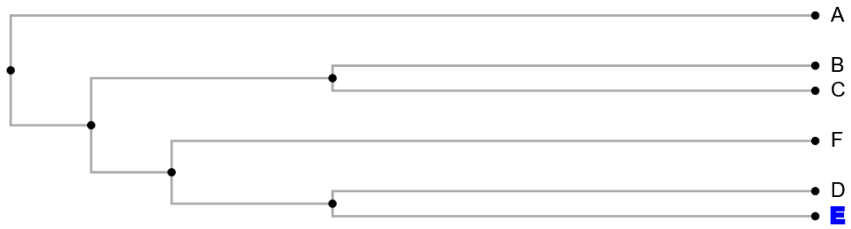


Figura 5.7: Dendrograma com a aplicação da função addLeafLabels.

A função `addInternalLabels`, semelhante à função anterior, seleciona todos os nós-pai e coloca um elemento `text` em cada um, com o identificador do nó. A figura 5.8 apresenta o mesmo dendrograma da figura anterior, mas com a aplicação da função `addInternalLabels`, bem como da função `addLeafLabels`.

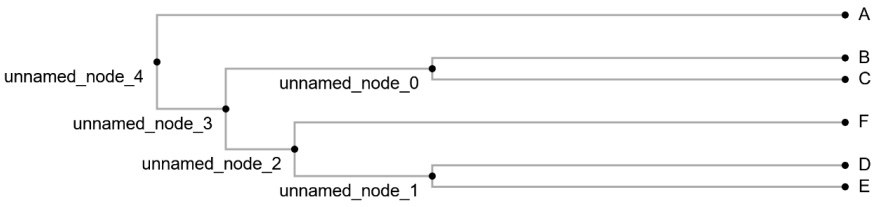


Figura 5.8: Dendrograma com a aplicação da função addInternalLabels.

Por fim, a função `addLinkLabels` seleciona todos os arcos, adicionando um elemento `text` em cada um, com o valor do tamanho do arco, presente no campo `data.data.value`,

de cada nó. A figura 5.9 apresenta um dendrograma com etiquetas nos arcos, com a aplicação da função `addLinkLabels`.

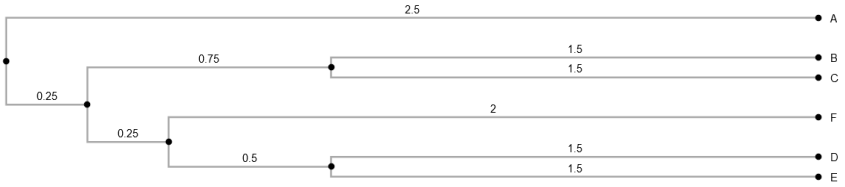


Figura 5.9: Dendrograma com a aplicação da função addLinkLabels.

Integração dos dados isolados

Quando são aplicados dados isolados às árvores, são construídos gráficos de barras nos nós folha que contêm essa informação. A função criada para o efeito tem o nome de `buildBarChart`. Esta utiliza a função presente na biblioteca D3.js, `stack`, para construir as várias secções dos gráficos de barras. Em cada nó é adicionado tantos elementos `rect`, quanto o número de isolados, em que a largura dos mesmos é calculada utilizando uma escala logarítmica.

Uma vez que a árvore pode sofrer modificações, é necessário guardar a informação relativa aos gráficos de barras, sendo adicionado o campo `barChart` a cada nó. Este campo é um objeto com as propriedades do gráfico de barras. Através da função `addBarCharts`, sempre que ocorre uma modificação na árvore depois de terem sido construídos os gráficos de barras, os mesmos são adicionados novamente à árvore.

Guardar o estado da árvore

De forma a ser guardado o estado atual de processamento da árvore, foi criada a função `save`, em que é percorrida a árvore que foi construída, sendo criado um objeto com as propriedade de cada nó, nomeadamente os campos `source`, `target` e `value` dos arcos, a cor do nó e os gráficos de barras aplicados ao mesmo. A figura 5.10 apresenta um exemplo deste objeto.

```
{
    "source": "unnamed_node_4",
    "target": "A",
    "value": 2.5,
    "barChart": [
        {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#f9bc0f",
            "x": "5",
            "y": "-5"
         }
    ]
}
```

Figura 5.10: Objeto produzido pela função save.

Esta função retorna um objeto com a árvore e as propriedades aplicadas à mesma, nomeadamente os valores do zoom e as escalas, para além dos atributos mencionados anteriormente.

Quando esta função é executada, é guardado um ficheiro com a extensão .json, com a informação retornada pela mesma. O conteúdo do ficheiro produzido encontra-se no anexo B.

Para carregar o estado da árvore, é utilizada a função `load`. Esta função atribui valores aos campos referentes às propriedades da árvore, construindo a árvore através da função `build`, e desenhando a mesma através da função `draw`.

Produção de um relatório

O relatório produzido, no formato PDF, é constituído por um título, pela árvore filogenética e, se estiverem aplicados filtros, é ainda constituído por um gráfico de setores, com estatística aplicadas à árvore. A árvore filogenética presente no documento corresponde à zona que se encontra visível na página, isto é, é feita uma cópia do documento HTML no momento em que é produzido o relatório.

Para a criação do documento PDF, é utilizada a biblioteca `jsPDF` [26, 27], que permite criar documentos PDF em Javascript. Este documento tem o tamanho de uma página A4. Para adicionar um título ao documento é utilizada a função desta biblioteca denominada `text`, onde é indicada a `string` com o texto a inserir, bem como a posição deste na página. A criação do documento e definição das propriedades do mesmo, bem como a adição do título do relatório encontram-se presentes na figura 5.11.

```
const doc = new jsPDF('p', 'pt', 'a4')
doc.setProperties({ title: "Report" })
doc.setFontSize(28)
doc.text(title, 290, 40, { align: 'center' })
```

Figura 5.11: Criação do documento PDF.

Para adicionar a árvore e o gráfico de setores ao documento, é necessário utilizar

uma outra biblioteca, denominada `saveSvgAsPng` [28], que obtém um data uri, de um documento png gerado a partir do elemento svg onde está presente a árvore e o elemento svg onde está presente o gráfico de setores. Esse data uri é utilizado pela função `addImage` da biblioteca `jsPdf`, para inserir estes elementos no documento PDF no formato de uma imagem com o formato png.

Para o documento PDF ser guardado é utilizada a função `save`, também da biblioteca `jsPDF`, com o nome pretendido para o documento.

O anexo A apresenta o ficheiro PDF resultante da produção do relatório para os dados de exemplo utilizados neste capítulo bem como no anterior, nomeadamente um dendrograma, com 11 perfis alélicos e com a aplicação dos filtros país e ano.

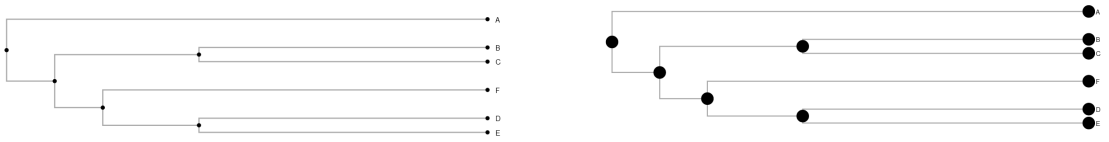
A tabela 5.1 apresenta as funções da biblioteca D3.js utilizadas nos módulos desenvolvidos, bem como alguns exemplos de funções implementadas no projeto.

Funções da D3.js	**Funções construídas no projeto**
hierarchy	build
stratify	clusterTree
zoom	radial
stack	draw
select	applyScale
selectAll	spread
-	horizontalScale
-	applyScaleText
-	collapse
-	expand

Tabela 5.1: Funções da biblioteca D3.js e funções implementadas no projeto.

Configurações visuais dos elementos da árvore

Foram implementadas funções para alterar a aparência tanto dos nós como dos arcos, nomeadamente a função `changeNodeSize`, que altera o tamanho do nó, isto é, altera o valor do raio do círculo, através do atributo `r`. A figura 5.12a apresenta um dendrograma com o tamanho dos nós por omissão, e a figura 5.12b apresenta o mesmo dendrograma depois de ter sido aumentado o tamanho dos nós.

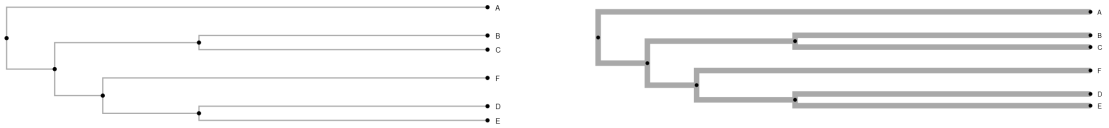


(a) Dendrograma com o tamanho por omissão para os nós

(b) Dendrograma depois de ter sido aumentado o tamanho dos nós

Figura 5.12: Resultado da função changeNodeSize

A função `changeLinkSize` altera a espessura da linha dos arcos, através do atributo `stroke-width`. A figura 5.13 apresenta um dendrograma com o tamanho dos arcos por omissão, e depois deste ter sido aumentado.

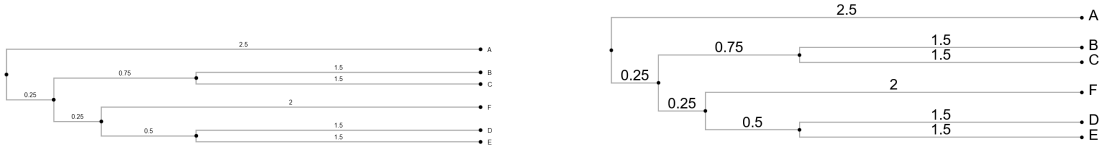


(a) Dendrograma com o tamanho por omissão para os arcos

(b) Dendrograma depois de ter sido aumentado o tamanho dos arcos

Figura 5.13: Resultado da função changeLinkSize

O mesmo foi implementado para as etiquetas, em que é possível aumentar o tamanho da letra para as diferentes etiquetas que podem ser aplicadas à árvore, nomeadamente nos nós e arcos. A função responsável por esta funcionalidade tem o nome de `changeLabelsSize`. O tamanho da letra é alterado no atributo `font`. A figura 5.14 apresenta um dendrograma com o tamanho de letra por omissão para as etiquetas, e o mesmo dendrograma com o tamanho aumentado da letra das etiquetas.

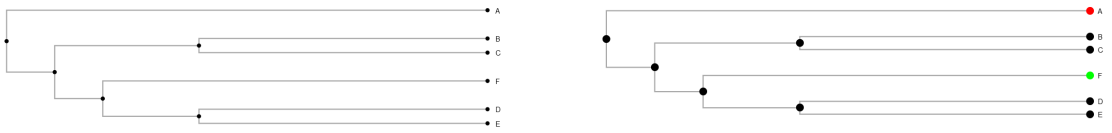


(a) Dendrograma com o tamanho por omissão para as etiquetas

(b) Dendrograma depois de ter sido aumentado o tamanho das etiquetas

Figura 5.14: Resultado da função changeLabelsSize

É ainda possível alterar a cor dos nós, através da função `changeNodeColor`. Esta função recebe o nó a alterar e a cor e, através dos atributos `fill` e `stroke`, a cor do nó é alterada. A figura 5.15 apresenta um dendrograma com os nós com a cor por omissão, isto é, pretos, e depois de terem sido alteradas as cores de dois nós, o nó A e o nó F.



(a) Dendrograma com a cor por omissão dos nós

(b) Dendrograma depois de ter sido alteradas as cores de dois nós

Figura 5.15: Resultado da função changeNodeColor

Nos exemplos anteriores foi sempre utilizado o dendrograma para exemplificar as funcionalidades implementadas, no entanto, estas aplicam-se também ao radial.

5.2 Aplicação Electron

5.2.1 data-render

O módulo `data-render` é responsável por processar os três tipos de dados utilizados pela aplicação, sendo estes a árvore filogenética, os perfis alélicos e os dados isolados. A árvore filogenética encontra-se no formato de newick ou nexus, enquanto que os perfis alélicos e os dados isolados encontram-se no formato de tabela. Este módulo é, assim, utilizado numa primeira fase, quando os dados são recebidos pela aplicação, no formato de `string`, em que é necessário processar os mesmos, e numa segunda fase, quando é necessário obter esses mesmos dados, quer seja para desenhar a árvore filogenética, e utilizando, para isso, o newick ou nexus, que foi convertido num objeto JSON, quer seja para construir as tabelas, que permitem a aplicação de filtros sobre a árvore.

Este módulo é assíncrono, ou seja, quando é obtida a árvore ou a informação dos perfis e isolados, é retornada uma `Promise`, contendo esses mesmos dados.

5.2.2 parsing

Este módulo exporta a função `parseNewick`, que é responsável por converter a árvore, que se encontra no formato de newick ou nexus, num objeto JSON. Uma vez que alguns nós intermédios, isto é, todos os nós que não são folhas, podem não ter nome, este módulo tem também a função de atribuir um identificador único a esses nós (com a designação `unnamed-node-id`), em que `id` é um identificador único atribuído a cada nó com estas características. Esta transformação é necessária, uma vez que o algoritmo utilizado posteriormente para a construção das árvores, necessita de receber nós com identificadores únicos.

5.3 Exemplos de utilização da aplicação Electron

A aplicação Electron permite realizar as ações descritas na secção 5.1. A figura 5.16 apresenta parte do menu que permite configurar as árvores, na aplicação desenvolvida para efeitos de testes, em que é possível pesquisar por um nó, centrando a árvore no mesmo (Search), adicionar e remover as etiquetas dos nós pai (parent labels), alinhar os nós (align nodes), isto é, não ter em consideração os tamanhos dos arcos, ficando

os nós alinhados por profundidade (apenas aplicável à visualização dendrograma), ou adicionar e remover as etiquetas dos arcos (link labels). É também possível alterar entre a escala linear e logarítmica, sendo que a opção por omissão é a escala linear. As setas horizontais permitem aumentar e diminuir a largura do gráfico e as setas verticais permitem aumentar e diminuir a altura do mesmo. No radial, apenas é possível aumentar e diminuir a largura da árvore.

É também possível alterar a cor de um nó, para isso é selecionado o identificador do nó e a cor pretendida, aumentar e diminuir o tamanho do nós (Node size), bem como a espessura dos arcos (Link Thickness). É ainda possível aumentar e diminuir o tamanho das etiquetas aplicadas à árvore (Labels Size). A figura 5.17 apresenta a restante parte do menu de configuração das árvores.

No *url* `https://github.com/AdrVB/Radial-Dendrogram-Visualization/wiki/Run-the-Electron-App#example-videos` encontram-se vídeos exemplificando a utilização da aplicação.

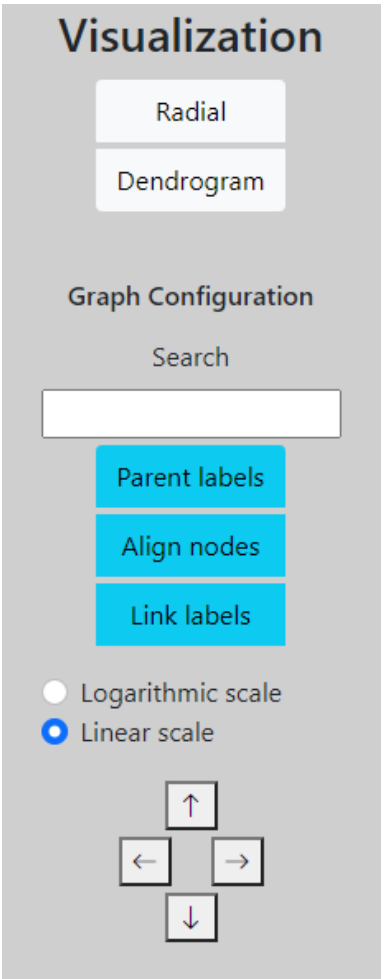


Figura 5.16: Opções de configuração das árvores.

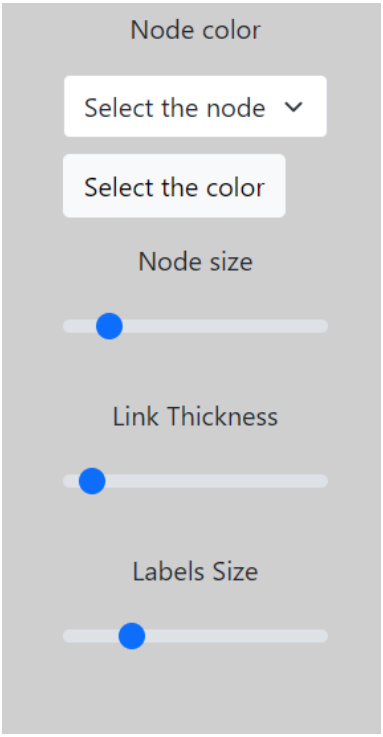


Figura 5.17: Outras opções de configuração das árvores.

Capítulo 6

Avaliação da solução

Para testar a eficiência das visualizações implementadas, foram realizados testes que visam demonstrar os tempos de construção e desenho das árvores, bem como o tempo de execução da aplicação de certas funcionalidades às mesmas. Os testes foram realizados no contexto da aplicação Electron, que integra os módulos de visualização.
Todos os testes foram realizados numa máquina com as seguintes características:

- Sistema Operativo: Windows 10
- CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- Memória: 16 GB

6.1 Avaliação Experimental

Uma vez que os módulos `data-render` e `parsing`, como mencionado nos capítulos anteriores, podem ser executados tanto no lado do servidor como no lado do cliente, foram realizados testes de comparação entre as duas abordagens, de forma a perceber se existem vantagens em processar os dados no lado do servidor, ou se todo o processamento pode ser realizado no lado do cliente.

Assim, foram registados os tempos de processamento de árvores, com diferentes números de perfis, quando o processamento é realizado no lado do servidor (versão 1) e no lado do cliente (versão 2).

De salientar que o número de perfis corresponde ao número de nós das árvores, e o número de arcos das mesmas é igual a (2 * V) - 1. em que V é o número de nós.

A Tabela 6.1 apresenta a média dos tempos de construção da árvore, quando o processamento dos dados é realizado no lado do servidor e no lado do cliente.

Número de perfis	**Versão 1**	**Versão 2**
100	2,90	2,64
6 000	54,39	51,80
11 000	75,08	71,14

Tabela 6.1: Média dos tempos de construção das árvores (em milissegundos).

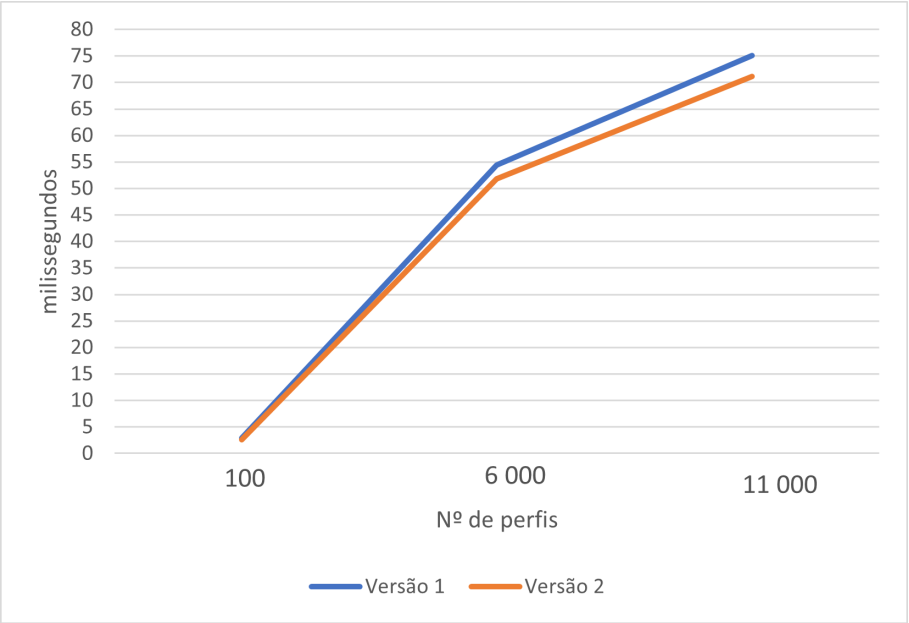


Figura 6.1: Representação gráfica da tabela 6.1.

A Tabela 6.2 apresenta a média dos tempos de desenho da árvore, quando o processamento dos dados é realizado no lado do servidor e no lado do cliente.

Número de perfis	**Versão 1**	**Versão 2**
100	11,46	11,02
6 000	205,83	211,94
11 000	400,15	387,31

Tabela 6.2: Média dos tempos de desenho das árvores (em milissegundos).

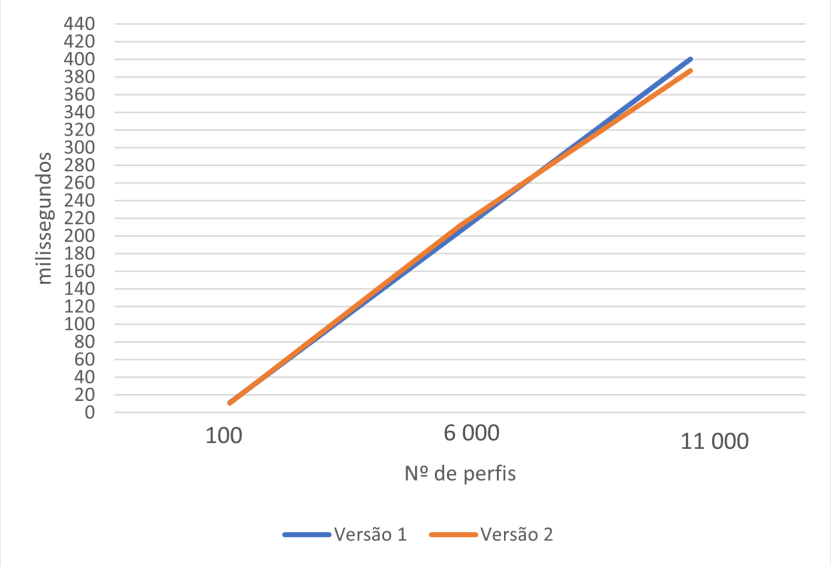


Figura 6.2: Representação gráfica da tabela 6.2.

A Tabela 6.3 apresenta a média dos tempos de colapsar a raiz da árvore, isto é, é colapsada a árvore na totalidade, quando o processamento dos dados é realizado no lado do servidor e no lado do cliente.

Número de perfis	**Versão 1**	**Versão 2**
100	4,33	3,89
6 000	96,99	98,81
11 000	152,35	155,81

Tabela 6.3: Média dos tempos de colapsar a árvore (em milissegundos).

A Tabela 6.4 apresenta a média dos tempos de expandir a raiz da árvore, quando o processamento dos dados é realizado no lado do servidor e no lado do cliente.

Número de perfis	**Versão 1**	**Versão 2**
100	21,6	22,7
6 000	684,19	709,89
11 000	660,71	605,89

Tabela 6.4: Média dos tempos de expandir a árvore (em milissegundos).

O tempo de expandir a árvore é consideravelmente superior ao tempo de colapsar a mesma, uma vez que enquanto na funcionalidade de colapsar a árvore, os nós e arcos são removidos, quando esta é expandida, todos os seus nós e arcos são desenhados novamente, aumentando, desta forma, o tempo de processamento.

Para testar o tempo de processamento dos dados no lado do servidor em comparação com o processamento no lado do cliente, os módulos `data-render` e `parsing` foram executados nestes dois casos, sendo registados os tempos obtidos. A Tabela 6.5 apresenta o tempo de processamento dos dados no lado do servidor e no lado do cliente.

Número de perfis	**Versão 1**	**Versão 2**
100	0,42	0,87
6 000	22,48	20,24
11 000	27,93	29,63

Tabela 6.5: Média dos tempos de processamento dos dados (em milissegundos).

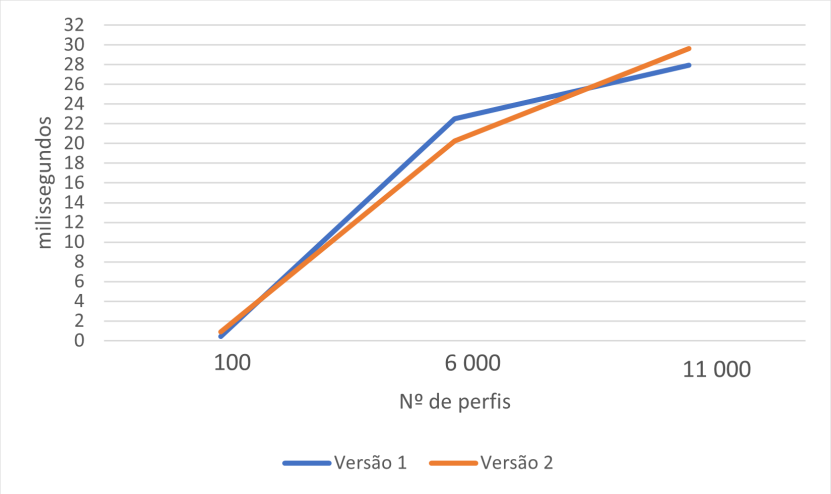


Figura 6.3: Representação gráfica da tabela 6.5.

Não se verificou uma diferença significativa no tempo de execução na construção das visualizações, quando o processamento dos dados é realizado no lado do servidor em comparação com o processamento dos dados no lado do cliente.

6.2 Testes de Usabilidade

Foram criados testes de usabilidade à aplicação Electron, de forma a avaliar a interface gráfica com o utilizador, bem como a utilização das funcionalidades disponibilizadas pela aplicação.

Capítulo 7

Conclusões

Através da realização deste projeto, percebemos a importância dos estudos epidemiológicos e genéticos de populações microbianas, uma vez que fornecem informações importantes no controlo de doenças infeciosas e na evolução das mesmas, bem como a importância de visualizar esses estudos da melhor forma possível, através de algoritmos de visualização.

A biblioteca e respetivos módulos de visualização desenvolvidos neste projeto, permitem enriquecer os estudos epidemiológicos, uma vez que permitem visualizar árvores filogenéticas nos formatos de dendrograma e radial, bem como a aplicação de diferentes filtros sobre as árvores, através da integração de dados complementares, que contêm informação importante relativamente à amostra em estudo. A biblioteca de visualização permite também:

- Colapsar e expandir sub-árvores da árvore em estudo, para permitir uma melhor visualização da árvore filogenética.
- Adicionar etiquetas aos nós e arcos da árvore, para identificar os mesmos.
- Produzir estatísticas aplicadas à árvore.
- Alterar os tamanhos e cores dos elementos da árvore.
- Guardar o processamento da árvore em estudo num dataset, para estudos estudos.
- Produzir um relatório como resultado do estudo da árvore.

Estes módulos, ao terem sido construídos de forma genérica, podem ser integrados no contexto de uma aplicação na área da filogenia, como serão integrados na aplicação PHYLOViZ, adicionando funcionalidades não existentes até ao momento, como é o caso da possibilidade de colapsar nós e a produção de um relatório.

É possível adicionar facilmente um outro tipo de visualização estática, se necessário, mantendo o mesmo contrato implementado pelas visualizações radial e dendrograma.

No futuro poderia ainda ser implementada a funcionalidade de adicionar dinamicamente uma linha aos dados isolados, atualizando as visualizações de acordo com a

nova informação, nomeadamente, a construção dos gráficos de setores e dos gráficos de barras aplicados aos nós da árvore.

Poderia também ser adicionada a funcionalidade de visualização da evolução de uma árvore filogenética ao longo do tempo, isto é, visualizar as várias fases de evolução de uma espécie ao longo do tempo. Isto permitiria uma melhor compreensão da evolução das espécies, enriquecendo assim, os estudos epidemiológicos.

A aplicação Electron desenvolvida para efeitos de teste pode ser melhorada no futuro, de forma a tornar-se numa aplicação com vários clientes, a ser utilizada na área dos estudos filogenéticos, para visualização dendrograma e radial de árvores filogenéticas, com a possibilidade de persistir os dados armazenados num dataset.

Referências

[1] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[2] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[3] Wikipedia. Dendrograma. URL `https://pt.wikipedia.org/wiki/Dendrograma`. Consultado em 07-04-2021.

[4] Christian Bachmaier, Ulrik Brandes, and Falk Schreiber. *Biological networks*. Taylor & Francis, 2014.

[5] Se Hang Cheong and Yain Whar Si. Force-directed algorithms for schematic drawings and placement: A survey. *Information Visualization*, 19:147387161882174, 01 2019.

[6] D3.js. URL `https://d3js.org/`. Consultado em 07-04-2021.

[7] Vivagraphjs. URL `https://github.com/anvaka/VivaGraphJS`. Consultado em 07-04-2021.

[8] A.Francisco, C.Vaz, P.Monteiro, J.Melo-Cristino, M.Ramirez, and J.A.Carriço. PHYLOViZ. URL `http://www.phyloviz.net/`, 2014. Consultado em 29-03-2021.

[9] 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.

[10] PHYLOViZ online. URL `http://online.phyloviz.net/index`. Consultado em 11-06-2021.

[11] Electron. URL `https://www.electronjs.org/`. Consultado em 19-05-2021.

[12] Projeto de investigação, desenvolvimento, inovação e criação artística (IDI&CA). 6ª edição. Instituto Politécnico de Lisboa. Diva: From data integration to visualization in large scale phylogenetic analysis.

[13] grapeTree. URL `https://github.com/achtman-lab/GrapeTree`. Consultado em 08-07-2021.

[14] phylo.io. phylo.io. URL `https://github.com/DessimozLab/phylo-io`. Consultado em 08-07-2021.

[15] Elijah Meeks. *D3. js in Action*. Manning Shelter Island, NY, 2015.

[16] Cytoscape.js. URL https://js.cytoscape.org/. Consultado em 11-05-2021.

[17] sigmajs. URL http://sigmajs.org/. Consultado em 07-04-2021.

[18] James Lamar Williams. *Learning html5 game programming: A hands-on guide to building online games using Canvas, SVG, and WebGL*. Addison-Wesley Professional, 2012.

[19] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse Bloomington, 2000.

[20] Michael Heydt. *D3. js by example*. Packt Publishing Ltd, 2015.

[21] Wikipedia. newick. URL https://en.wikipedia.org/wiki/Newick_format. Consultado em 06-06-2021.

[22] David R Maddison, David L Swofford, and Wayne P Maddison. Nexus: an extensible file format for systematic information. *Systematic biology*, 46(4):590–621, 1997.

[23] Luana Silva, Cátia Vaz, and Alexandre Francisco. Library of efficient algorithms for phylogenetic analysis. *arXiv preprint arXiv:2012.12697*, 2020.

[24] d3. URL https://devdocs.io/d3~6/. Consultado em 08-07-2021.

[25] dfs. URL https://en.wikipedia.org/wiki/Depth-first_search. Consultado em 17-07-2021.

[26] jsPDF. URL https://artskydj.github.io/jsPDF/docs/jsPDF.html. Consultado em 07-07-2021.

[27] jspDF. URL https://github.com/MrRio/jsPDF. Consultado em 08-07-2021.

[28] savePNG. URL https://github.com/exupero/saveSvgAsPng. Consultado em 08-07-2021.

Simbologia

JSON JavaScript Object Notification

NGS Next Generation Sequencing

PDF Portable Document Format

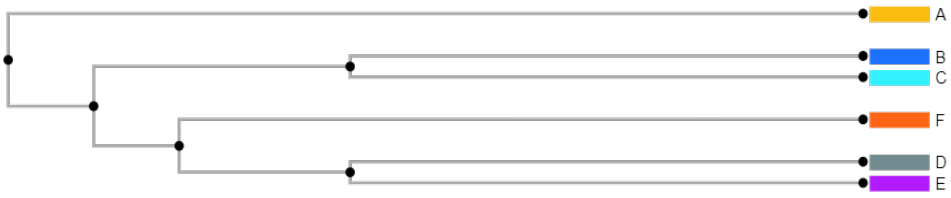
SVG Scalable Vector Graphics

T Táxon

Anexo A

Exemplo do Relatório produzido para uma árvore filogenética com 11 perfis e com os filtros país e ano aplicados

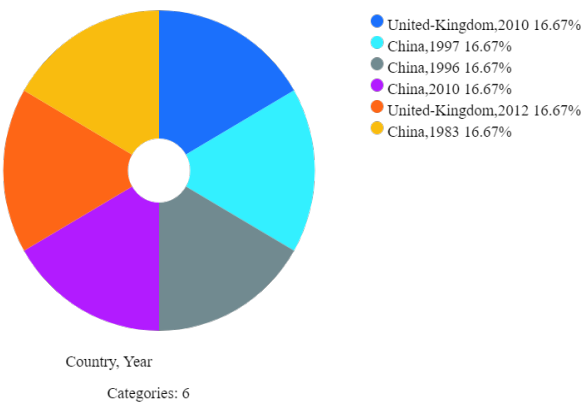
Phylogenetic tree example



0.59

Total number of profiles: 11

Number of isolates: 6



Anexo B

JSON produzido pelo módulo save para os dados de exemplo


```
{
  "type": "dendrogram",
  "data": {
    "tree": [
      {
        "source": null,
        "target": "unnamed_node_4",
        "value": 0
      },
      {
        "source": "unnamed_node_4",
        "target": "A",
        "value": 2.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#f9bc0f",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_4",
        "target": "unnamed_node_3",
        "value": 0.25
      },
       {
        "source": "unnamed_node_3",
        "target": "unnamed_node_0",
        "value": 0.75
      },
```

```
{
        "source": "unnamed_node_0",
        "target": "B",
        "value": 1.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#1b70fc",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_0",
        "target": "C",
        "value": 1.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#33f0ff",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_3",
        "target": "unnamed_node_2",
        "value": 0.25
      },
      {
        "source": "unnamed_node_2",
        "target": "F",
        "value": 2,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#fe6616",
            "x": "5",
            "y": "-5"
          }
        ]
      },
```

```
{
        "source": "unnamed_node_2",
        "target": "unnamed_node_1",
        "value": 0.5
      },
      {
        "source": "unnamed_node_1",
        "target": "D",
        "value": 1.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#718a90",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_1",
        "target": "E",
        "value": 1.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#b21bff",
            "x": "5",
            "y": "-5"
          }
        ]
      }
    ],
    "input": {
      "schemeGenes": [
        "ST",
        "Gene_1",
        "Gene_2",
        "Gene_3",
        "Gene_4",
        "Gene_5"
      ]
```

```
"nodes": [
        {
          "key": "B",
          "profile": [
            "B",
            "5",
            "4",
            "4",
            "2",
            "15"
          ],
          "isolates": [
            [
              "B",
              "United-Kingdom",
              "2010"
            ]
          ]
        },
        {
          "key": "C",
          "profile": [
            "C",
            "5",
            "3",
            "4",
            "6",
            "2"
          ],
          "isolates": [
            [
              "C",
              "China",
              "1997"
            ]
          ]
        },
        {
          "key": "D",
          "profile": [
            "D",
            "2",
            "2",
            "4",
            "8",
            "7"
          ]
```

```
      "isolates": [
        [
          "D",
          "China",
          "1996"
        ]
      ]
    },
    {
      "key": "E",
      "profile": [
        "E",
        "2",
        "2",
        "1",
        "1",
        "12"
      ],
      "isolates": [
        [
          "E",
          "China",
          "2010"
        ]
      ]
    },
    {
      "key": "F",
      "profile": [
        "F",
        "1",
        "3",
        "1",
        "1",
        "1"
      ],
      "isolates": [
        [
          "F",
          "United-Kingdom",
          "2012"
        ]
      ]
    },
```

```
    {
      "key": "unnamed_node_1"
    },
    {
      "key": "unnamed_node_0"
    },
    {
      "key": "unnamed_node_2"
    },
    {
      "key": "A",
      "profile": [
        "A",
        "10",
        "6",
        "6",
        "12",
        "13"
      ],
      "isolates": [
        [
          "A",
          "China",
          "1983"
        ]
      ]
    },
    {
      "key": "unnamed_node_3"
    },
    {
      "key": "unnamed_node_4"
    }
  ],
  "links": [
    {
      "source": null,
      "target": "unnamed_node_4",
      "value": 0
    },
```

```
    {
      "source": "unnamed_node_4",
      "target": "A",
      "value": 2.5,
      "barChart": [
        {
          "width": "44.45378125959109",
          "height": "11",
          "fill": "#f9bc0f",
          "x": "5",
          "y": "-5"
        }
      ]
    },
    {
      "source": "unnamed_node_4",
      "target": "unnamed_node_3",
      "value": 0.25
    },
    {
      "source": "unnamed_node_3",
      "target": "unnamed_node_0",
      "value": 0.75
    },
    {
      "source": "unnamed_node_0",
      "target": "B",
      "value": 1.5,
      "barChart": [
        {
          "width": "44.45378125959109",
          "height": "11",
          "fill": "#1b70fc",
          "x": "5",
          "y": "-5"
        }
      ]
    },
```

```
{
        "source": "unnamed_node_0",
        "target": "C",
        "value": 1.5,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#33f0ff",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_3",
        "target": "unnamed_node_2",
        "value": 0.25
      },
      {
        "source": "unnamed_node_2",
        "target": "F",
        "value": 2,
        "barChart": [
          {
            "width": "44.45378125959109",
            "height": "11",
            "fill": "#fe6616",
            "x": "5",
            "y": "-5"
          }
        ]
      },
      {
        "source": "unnamed_node_2",
        "target": "unnamed_node_1",
        "value": 0.5
      },
```

```
{
          "source": "unnamed_node_1",
          "target": "D",
          "value": 1.5,
          "barChart": [
            {
              "width": "44.45378125959109",
              "height": "11",
              "fill": "#718a90",
              "x": "5",
              "y": "-5"
            }
          ]
        },
        {
          "source": "unnamed_node_1",
          "target": "E",
          "value": 1.5,
          "barChart": [
            {
              "width": "44.45378125959109",
              "height": "11",
              "fill": "#b21bff",
              "x": "5",
              "y": "-5"
            }
          ]
        }
      ],
      "metadata": [
        "ST",
        "Country",
        "Year"
      ]
    }
  },
  "graph": {
    "nodeSize": [
      15,
      1
    ],
```

```
"style": {
     "align": false,
     "leafLabels": true,
     "linkLabels": false,
     "parentLabels": false,
     "labels_size": 12,
     "links_size": 2,
     "nodes_size": 3,
     "barChart": true
   },
   "scale": {
     "linear": {
       "vertical": {
         "value": 1,
         "limits": [
           0.1,
           10
         ],
         "scalingFactor": 1,
         "step": 0.02
       },
       "horizontal": {
         "value": 1,
         "limits": [
           0.1,
           10
         ],
         "scalingFactor": 1000,
         "step": 0.01
       }
     },
     "log": {
       "vertical": {
         "value": 1,
         "limits": [
           1,
           100
         ],
         "scalingFactor": 0.01,
         "step": 0.02
       },
```

```
  "horizontal": {
          "value": 30,
          "limits": [
            1,
            100
          ],
          "scalingFactor": 15,
          "step": 2
        }
      }
    }
  },
  "canvas": {
    "zoom": {
      "x": 100,
      "y": 200,
      "scale": 1
    }
  }
}
```