Final Year Project Report

A Javascript Library for Visualising Biological Networks

Séamus Ó Ceanainn

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Colm Ryan

UCD School of Computer Science
University College Dublin

August 4, 2018

Project Specification

General Information:

The bulk of the work in our cells is performed by proteins. These proteins work with each other in dense interaction networks to carry out various biological functions. Major efforts have been made over the last decade to map the interactions between all human proteins, resulting in databases containing large protein interaction networks (20,000 proteins, 300,000 interactions). These networks act as a useful scaffold for interpreting the results of large-scale biological experiments. Often biologists perform experiments that test thousands of candidate proteins for their involvement in some process (e.g. proteins whose mutation is associated with drug resistance in cancer, proteins that are associated with a particular disease). The end result of these experiments is typically a small (10-500) list of 'hit' proteins. Mapping these hits onto a protein interaction network can help biologists make sense of their experiments by revealing which of the hits interact with each other. For instance the below image displays the protein interactions between proteins identified as good drug targets in a specific type of breast cancer:

The goal of this project is to develop / adapt a javascript library to enable the visual mapping of 'hit lists' from experiments onto a protein interaction network. A biologist should be able to provide a list of protein identifiers and be presented with a javascript visualisation of the interactions between the proteins, like the above static image but dynamic and drawn in the browser. In addition to a list of proteins the user should be able to specify various parameters to customise the visualisation (node size, colour etc.).

Multiple different databases (string-db.org, thebiogrid.org) store protein interaction networks that have been curated according to differing criteria. These databases provide APIs to retrieve the interactions between user specified proteins. The library should provide a level of abstraction such that the underlying interaction network can come from any of these databases.

Core:

Evaluate different graph drawing libraries (e.g. cytoscape.js, arbor.js, D3.js) to build upon. At a minimum these libraries should facilitate custom node / edge sizes and colours and different graph drawing algorithms. Develop a library that can take a list of proteins, associated metadata (colour / size), and a database ID and return a javascript visualisation of the interactions between the proteins. Provide options for the user to specify graph drawing options (force-directed layout, circular layout) Develop a website that demonstrates the libraries functionality (e.g. allow the user input a list of proteins, retrieve the interactions from a database, and present the results to the user)

Advanced:

Integrate the resulting library with www.cancergd.org. This site currently uses statically generated images to show interactions between drug targets in cancer, but a javascript visualisation would significantly improve its usability.

Abstract

This project aims to produce a linkable Javascript library that is able to handle all aspects of biological network visualisation, including generating and parsing a user input form, calling the relevant API to retrieve the interaction network for the genes provided in user input, and finally using a graph visualisation framework to produce an SVG representation of the biological network. Encorporating this library into an existing website should be as easy as linking to the library in HTML and passing a list of selected genes to a function call, similar to as shown below. The result graph should be embedded in a user specified container and have features and functionality similar to other existing solutions to biological network visualisation. Advanced users should be able to select which features of the library they wish to use via a library API.

```
<script src="./path/to/this/library.js"></script>
<script>
    generateGraphIn("someContainerID");
</script>
```

Acknowledgments

I would like to thank my supervisor Colm Ryan for his help in developing this project. Colm was such an important source of feedback and always offered great advice on how to improve my project.

I would also like to thank the technical team behind String DB, particularly Damian Szklarczyk. When I ran into issues with their API one weekend I emailed them on a Saturday night with the issue. Damian replied to me on the Sunday morning saying he would raise the issue with the team and by Tuesday afternoon they had pushed changes to their API that allowed it to be called from external domains by web browsers.

Table of Contents

Chapter 1: **Introduction**

1.1 Biological Networks:

In terms of this project, when we are discussing biological networks we are referring specifically to gene interaction networks. Throughout the project we use the terms 'genes' and 'proteins' interchangeably. Each gene produces a unique protein, and two genes are said to interact if the proteins produced by those genes interact. Genes are represented by nodes in our graphs, and all interactions represented by edges.

1.1.1 Visualising Biological Networks:

Biological network visualisation has important applications in genetic research. Biologists use pathways identified in biological networks to differentiate between healthy and diseased states. When it comes to the problem of visualising gene interaction networks, solutions already exist, two examples being StringDB and BioGrid. Both websites have curated their own databases of known and predicted protein-protein interactions, and allow users to search for a protein or list of proteins and generate the resultant interaction network. As part of the output given on both websites users can view and interact with a visualisation of the network.

When it comes to the problem of visualising gene interaction networks on a separate webpage, no truly interactive solutions exist. The BioGrid will only allow users to output the visualised network as a PNG or get the network information in TSV or JSON format from its RESTful API service. The StringDB API does support exporting the network information in SVG (Scalable Vector Graphics) format, which allows us to call the API to retrieve the graph, however all interactivity of the graph is then lost.

1.1.2 Retrieving Network Data:

In order to produce a graph or a protein-protein interaction network, we will be dependent on the protein interactions lists returned from either the StringDB or BioGrid APIs. It is envisaged that our library will allow users to choose between both APIs as part of user input. This is because the interactions list generated by both APIs differ. The BioGrid database is a curation of experimentally proven interactions, the database is updated automatically using text mining systems that analyse new scientific publications. These systems try to predict whether or not a new interaction was proven experimentally within these publications. The StringDB database however contains a total of 7 different scores, as well as a final combined score for that interaction. This combined score is the estimated probability of there being an interaction between two proteins, as a value ranging between 0 and 1.

Chapter 2: **Background Research:**

2.1 Biological Networks:

The study of protein-protein interaction networks is a relatively new concept, and one that has the ability to have a significant impact on how we identify and treat various diseases. For a long time we have been able to identify genes that are associated to various diseases such as breast cancer susceptibility (Wooster et al. [28]), however the cause of this association between gene and disease is not always evident. This is due to the way that genes interact with each other, as well as environmental factors.

By studying the way that proteins and genes react with each other in these interaction networks, biologists can help differentiate between healthy and diseased states by identifying pathways. In fact one such study into the effects of smoking (Charlesworth et al. [17]) found that smoking related genes were over-represented in categories that correspond with "immune response, cell death, cancer, natural killer cell signaling and xenobiotic metabolism". Although that was expected, the study also unexpectedly proved that smoking actually affected protein-protein interactions within the immune-inflammatory response, which explained how smoking affects the immune system and why there are instant benefits when one quits smoking.

Generating interaction data and visualising the corresponding networks has now become an essential part of how biologists attempt to understand how the presence of certain genes and / or the presence of external environmental factors impact other genes in the network and potentially cause certain genetic diseases such as cancer.

2.2 Existing Biological Network Websites:

Given the large number of genes and proteins between which there could be interactions, it would be impossible for one single organisation to experimentally prove all possible interactions for any species. Given the importance of protein-protein interaction networks in identifying the causes of certain diseases, and given the difficulty in experimentally proving all possible interactions, two databases in particular have appeared to curate interaction data. BioGrid uses text miners to try and determine experimentally proven interactions , and StringDB takes a different approach by computing seven different scores and using them to calculate an 8th and final combined score.

2.2.1 StringDB:

StringDB is created by a consortium of four members - EMBL Heidelberg, Swiss Institute of Bioinformatics, University of Zurich and the NNF Center for Protein Research. The database consists of 9,643,763 proteins from 2,031 organisms, with 25,914,693 interactions with a confidence score of over 0.9, and 71,673,028 interactions with a confidence score of over 0.7 [13]. As mentioned above, these confidence scores are calculated from the combination of seven independent scores that each take different factors into account.

Graph Visualisation Features:

The StringDB websites allows users to search for a gene and output the network interaction graph containing all interactors for that gene. The graph displayed is an interactive SVG, that users can update by specifying a minimum threshold for the confidence score or which factors are used in calculating the score (which of the seven independent scores are used in calculating the confidence score). Users can also apply clustering to the graph.

A wide variety of data is also shown alongside the graph, including a list of predicted functional partners arranged in order of confidence score, network statistics and functional enrichments within the network. The network can be exported in many different formats, including SVG, PNG, JSON and XML. Clicking on a node brings up a comprehensive description of a node (see figure 2.2), as well a number of options such as re-centering the graph around that node, adding interactions of that node to the graph, and redirects to other sites to display the protein sequence, pathways, etc. of that node.

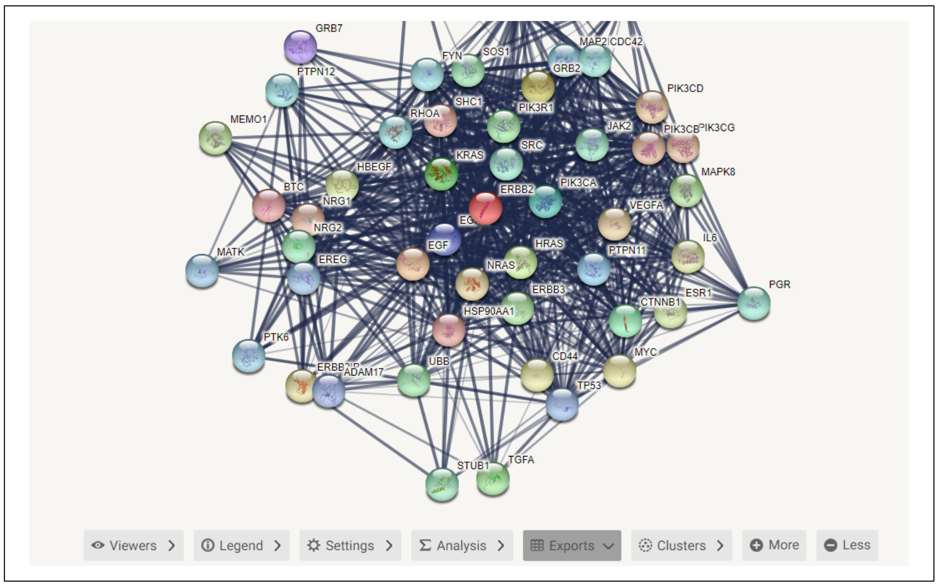
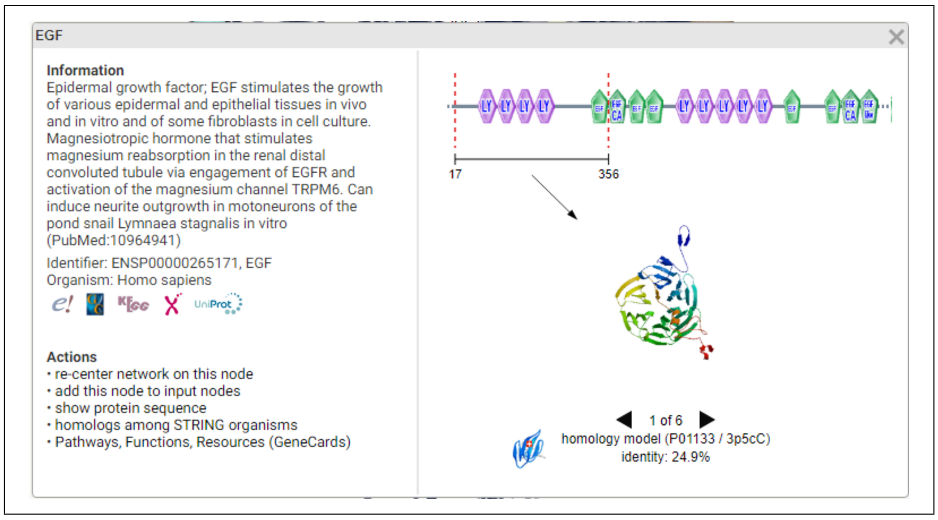


Figure 2.1: Visualisation of the Interaction Network of the ERBB2 Gene in StringDB



EGF

Information
Epidermal growth factor; EGF stimulates the growth of various epidermal and epithelial tissues in vivo and in vitro and of some fibroblasts in cell culture. Magnesiotropic hormone that stimulates magnesium reabsorption in the renal distal convoluted tubule via engagement of EGFR and activation of the magnesium channel TRPM6. Can induce neurite outgrowth in motoneurons of the pond snail Lymnaea stagnalis in vitro (PubMed:10964941)
Identifier: ENSP00000265171, EGF
Organism: Homo sapiens

Actions
- re-center network on this node
- add this node to input nodes
- show protein sequence
- homologs among STRING organisms
- Pathways, Functions, Resources (GeneCards)

Figure 2.2: Pop-up window when you click on a node in StringDB

Interaction List API:

StringDB has a RESTful API that allows users to query the database to return an interaction network containing one or more given genes. It will return the network in either SVG, TSV, JSON or PSI-MI format. The user can specify gene identifiers, species identifiers and a required score. Interestingly enough users can also specify a value 'k' for addNodes (default is 0 for a list of genes and 10 for one gene) which will add the next 'k' highest confidence interactors to the graph [11].

The result of an API call returns a list of interactions, where each interaction consists of the preferred names and the internal identifiers used by String for the proteins, the NCBI taxon identifier for the interaction, and the eight different scores for the interaction.

2.2.2 BioGrid:

The BioGrid is a project between four different organisations - Princeton University, Université de Montréal, University of Edinburgh and the Lunenfeld-Tanenbaum Research Institute. It has many partners which use their data, including StringDB [5].

The BioGrid currently stores over 1,400,000 interactions, curated from datasets, studies and through text mining systems that have identified interactions from over 57,000 publications [4]. It is affiliated to 36 different labs which investigate interactions. The BioGrid is clearly more focused on experimental evidence of interactions than StringDB which can be suitable when we need proven interactions, but the cost of this is that the BioGrid only contains 1/18th of the interactions that StringDB has.

Graph Visualisation Features:

The BioGrid allows users to search for one gene at a time. It will then present the user with a list of interactions and interactors. The user can navigate to a 'Network' tab which will then display a graph visualisation of the interaction network.

From the graph toolbar users can export the graph in a PNG format, filter out chemicals, physical experiments or genetic experiments, change between four layouts (default is force-directed) or access links to further resources or help pages. Users can set the zoom level from the toolbar but cannot pan or zoom from within the graph. Users can also select a value for minimum evidence from a dropdown bar in the toolbar, the evidence number is defined as the "number of unique curated interactions referencing the association" [10].

By left clicking on a node within the graph users can drag and pin nodes. By right clicking on a node users can display the neighbours or details of that particular node. They can also remove that node from the graph, or display the graph for that node. The pop up window for node details is no where near as comprehensive as the details given by the StringDB graph, in this case it consists of a one-line description of the gene and any known aliases for that particular gene.

Interaction List API:

The BioGrid API requires an access key but one can be acquired instantly when you register for BioGrid. They claim that this is to ensure that they can contact and update all application owners who depend on the service in the case of service failure. Once you can pass an access key it is very easy to submit queries to the RESTful API service, however the BioGrid has a lot of optional parameters that can be passed as part of the request that seem like a good idea in theory but in

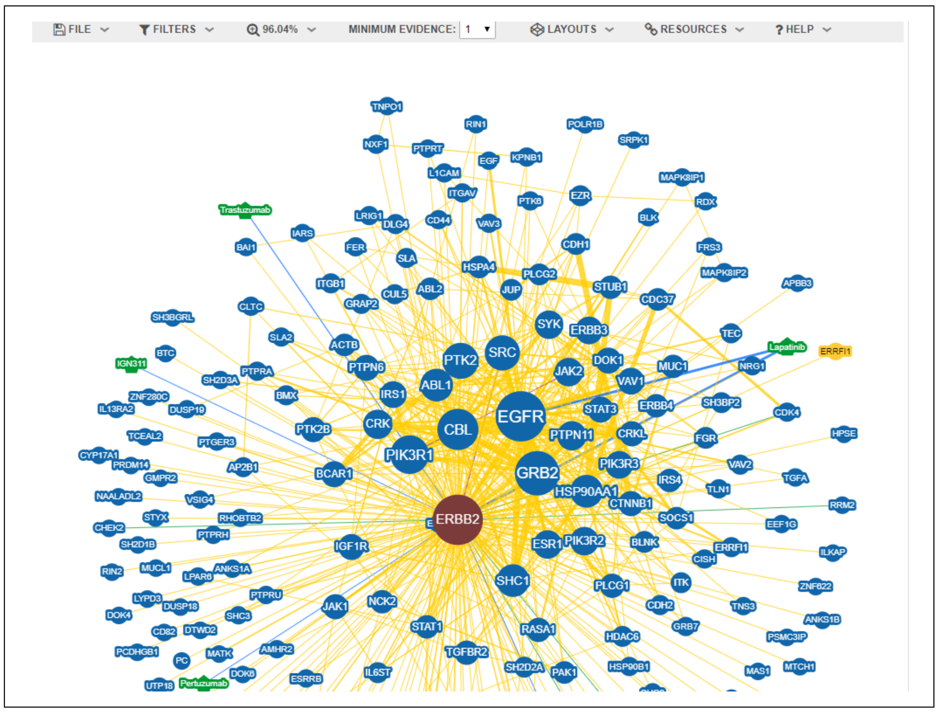


Figure 2.3: Visualisation of the Interaction Network of the ERBB2 Gene in BioGrid

practice end up making the API overcomplicated with twenty four possible input parameters.

The output is similarly a lot more complicated than it needs to be, especially compared to StringDB (which was both IDs and names, species taxon ID and then the scores). In terms of the interaction that output contains eleven data elements for the interaction itself and six data elements for each of the two proteins in the interaction [6]:

2.3 Existing Graph Visualisation Libraries:

2.3.1 Introduction:

The key requirements stemming from the project description is that the visualisation library must support different graph drawing algorithms and allow for us to map variables onto the graph (i.e. define node colour or edge thickness based on input data). The library needs to be a Javascript based library that can be a dependency of the resulting project library. Many Javascript libraries researched including Sigma.js, Ember.js and Bio.js were unsuitable as they needed to be served using Node.js.

A Note on the Force-Directed Layout:

The force-directed algorithm works by modeling edges as springs that attract connected nodes and repel disconnected nodes (Fruchterman and Reingold [20]). A greater edge weight would result in a shorter edge length, whereas in hierarchical graph layouts we rely on edge thickness to represent edge weight. Both manners of representing edge weight can even be combined by increasing edge thickness when edge weight is increased in a force-directed layout.

In terms of usage in this particular project, the resulting graph would have its nodes evenly distributed and that the closer two proteins are on the graph the more likely it is that they will interact. This makes it easier for users to see the difference in confidence scores for interactions. Due to the fact that this layout produces a graph that is very intuitive for a user to read, it is a layout that I came to require from the graph visualisation library that I would choose.

2.3.2 CytoScape.js:

Cytoscape.js is a graph visualisation library that was designed to be the online equivalent of the Cytoscape desktop application. Cytoscape.js can either be linked to as a Javascript library or as a Node.js package. It has impressive core functionality out of the box when it comes to graphing networks, including zoom functionality, alternate layouts (including the COSE layout which is an interpretation of the force-directed layout) and path finding algorithms built in. This is improved on by 'extensions' that add more functionality, however I found that many of these extensions are implemented for use with a Node.js implementation only.

Cytoscape.js graphs are styled using style sheets very similar to CSS, and graph rendering is very high performing since the graph is rendered in a HTML canvas. While this approach gives better performance over rendering the graph as an SVG object like other libraries, it sacrifices the ability to export an SVG object and can only export image (i.e. PNG, JPEG) or data (i.e. JSON, TSV) formats.

The combination of high core functionality and the use of HTML canvas means that Cytoscape.js is a very opinionated library. While it is highly customisable, there are limitations to how customisable it is since we can only use the Cytoscape.js API to interact with the graph. This also means that clients would have to learn the entire Cytoscape.js API in order to customise the behaviour of the graph, whereas with DOM based libraries clients could use regular Javascript to select and interact with graph elements.

2.3.3 D3.js:

D3.js is a Javascript based visualisation library that was built to handle all forms of data visualisation in the browser. Similarly to CytoScape.js it can be linked to as either a Javascript library or a Node.js package. It has no real core functionality out of the box but offers a wide arrange of functions that can be called using the D3 selector (similar to how JQuery uses the $ selector). What makes D3.js so powerful is that it can be used to define an SVG element, and then it creates and manipulates DOM elements inside of that SVG graph.

D3.js can create force directed graphs using arrays of node and edge objects, and can even read these arrays straight from JSON or as delimited separated values. It contains built in zooming and scaling functions that easily allows us to define and bind our own pan/zoom functionality to a graph. It also contains built in force functions that we can add to the graph and modify the strength of to get the desired force directed layout that we want. We can overwrite the positioning of elements by binding mouse events using the built in dragging functions, and has many timing functions to allow us to calibrate the performance of the graph animation. D3 has built in path finding algorithms, and if all that wasn't enough, includes a powerful selector so that we can easily apply a function to a certain type of element (e.g. all nodes).

Although D3.js might be lacking in out of the box functionality, it is very well documented and there are a lot of examples created to help a user starting off with the library. However with the power of the functions that are built into the library combined with the fact that all nodes / edges are implemented as DOM elements means that D3.js has the ability to implement any feature you can imagine and the only limit is the performance of the browser rendering the graph.

2.3.4 Arbor.js:

Although Arbor.js is described as a graph visualisation library, it is essentially only a physics module for a visualisation library. It has pre-built force-directed algorithms and will allow users to add and remove nodes and edges, however all drawing methods must be defined by the user [3]. This would just be far too challenging within the scope of the project. Arbor.js is essentially the starting point for a graph visualisation library with the core data structure and a force-directed algorithm implemented.

2.3.5 Alchemy.js:

Alchemy.js is a very basic Javascript visualisation library that will allow users to create force-directed graphs [1]. It allows users to add, remove or hide nodes and edges, import nodes and edges from JSON, and set the properties of the nodes and edges. To use any other layout the user would have to define their own algorithms and use that algorithm to set the X and Y properties of every node and edge. This library does not support binding events from graph elements (e.g. when a node is dragged) which is a major disadvantage in terms of interactivity.

2.3.6 Vis.js:

Vis.js is a simple and lightweight Javascript libraries that will draw networks using a force-directed layout. Unfortunately no other graph layout types are built-in and the user must define their own algorithms to use other layouts (e.g. circle, grid). In-built functionality[14] is really limited to creating and drawing nodes and edges, importing nodes or edges from JSON, a physics module to handle the force-directed layout, clustering algorithms to group nodes and methods to bind events from elements in the graph (e.g. when a node is clicked). Graphs are rendered as a HTML canvas.

2.4 Gene Data Enrichment:

In order to create a better environment for biological graph exploration, some data enrichment has to be performed. Detailed information on each interaction between genes is returned in the APIs considered for requesting gene networks. The BioGrid API returns interaction information which includes the PubMed ID corresponding to the academic paper in which the interaction was proven experimentally. For StringDB API calls the interaction information includes the seven different scores used to calculate the combined score. Unfortunately, the only gene information returned by these calls is a few basic attributes (gene name and internal gene ID) which are present in the interaction data.

2.4.1 Mapping from StringDB:

When first researching data enrichment I tried to find a way of mapping from StringDB IDs to external data sources. The downloads section of the StringDB developers website contains some files containing mappings from StringDB IDs to either Uniprot or Entrez Gene. Unfortunately these mappings can't be accessed through the StringDB API. The Uniprot API does allow users to request mappings between various internal IDs, and supports StringDB, the BioGrid, Entrez Gene and Ensembl, which were all part of my research. Since the Uniprot API would allow us to map identifiers between StringDB and various other sources, it means we could then generate requests for gene information from other sources.

2.4.2 Uniprot:

The Uniprot API also allows users to request detailed gene information in the form of an XML document. This XML document is quite large and difficult to parse, with roughly 1,200 lines per gene. It does provide a lot of information that would be useful to users, including synonyms for the gene and its associated protein, a list of all biological processes and molecular functions associated with the gene, and a short paragraph describing the gene's function.

2.4.3 Entrez Gene:

The Entrez Gene API will also return gene information in the form of an XML document. Although the NCBI website (which operates the Entrez Gene database) contains detailed information on genes, the API will either return a 20 line XML summary containing some very basic information about the gene, or it can also be used to retrieve the entire DNA sequence for a gene. Unfortunately it doesn't provide much information that would be useful for a biologist to understand the gene if they came across it whilst exploring a network.

2.4.4 Ensembl Gene:

Ensembl is a website used by biologists to visually explore the structure of a gene's DNA. It does offer some information on genes on its website, and this information is accessible through an API. Unfortunately each attribute that it stores for a gene must be requested through separate request and would require 4 or 5 separate requests to generate basic gene information.

Chapter 3: **Project Approach and Design:**

3.1 Choosing Technologies:

3.1.1 D3.js:

After careful consideration of the graphing libraries discussed in section 2.3, D3.js was ultimately chosen as the graphing library. Arbor.js was simply too much of a challenge since all drawing functions would have to be written from scratch. Vis.js and Alchemy.js were lacking the support of multiple layouts as requested by the project description, and they just weren't as powerful as the remaining libraries. Furthermore the lack of event binding methods in Alchemy.js meant that it would be an additional challenge to create an interactive graph using that library.

It was very difficult to choose between D3.js and Cytoscape.js since Cytoscape.js was built especially to support the visualisation of biological networks, but D3.js has the benefit of more extensive documentation and many more examples available online for beginners. The deciding factor ended up being the manner in which the graphs were rendered. Cytoscape.js renders the graph in a HTML canvas, which improves performance but ultimately hinders interactivity. D3.js renders the graph as an SVG, where each node and edge are DOM elements that can be selected and manipulated with Javascript in the same manner as any other DOM element.

3.1.2 StringDB:

After comparing the StringDB and the BioGrid APIs as discussed in section 2.2, I initially focused on StringDB since it provided multiple scores, including a score based purely on experimental data, as well as many other scores to predict interactions, whereas the BioGrid only used experimental data to return known interactions. The extra information on interactions returned by StringDB allowed me to add more features to the graph (e.g. filter links in the graph based on the different scores, or according to a minimum confidence threshold). The project specification requested that multiple data sources could be used so the library was ultimately expanded to also support calling the BioGrid API.

3.1.3 Uniprot:

The fact that the Uniprot API allows users to map both StringDB and BioGrid identifiers between various other external identifiers meant that calls to the Uniprot needed to be implemented to enrich our data to get detailed information on genes. The Ensembl REST API would have been too difficult to implement since each piece of information on a gene required a separate API call (which would have resulted in 4-5 requests per gene). Although Uniprot and Entrez Gene both returned information on genes in XML files, the Entrez Gene structure for the XML response would have been much easier to parse. Unfortunately the Entrez Gene API didn't return the gene summary which is a one paragraph description of the gene and is available on the website. Uniprot was chosen because it returned all the information needed for genes. The major disadvantage of the Uniprot API was that the XML file it returns is poorly structured and ended up causing the parsing function in the library for this XML file to be highly complicated and difficult to read.

3.2 Project Development:

An iterative approach was taken when developing this project. A basic version of the library was created to coincide with the submission of the interim report. Following the initial version of the library many new features were added iteratively to meet the project specifications. My project supervisor helped me to decide what features to add to the library and in what order to implement them.

3.2.1 Initial Design:

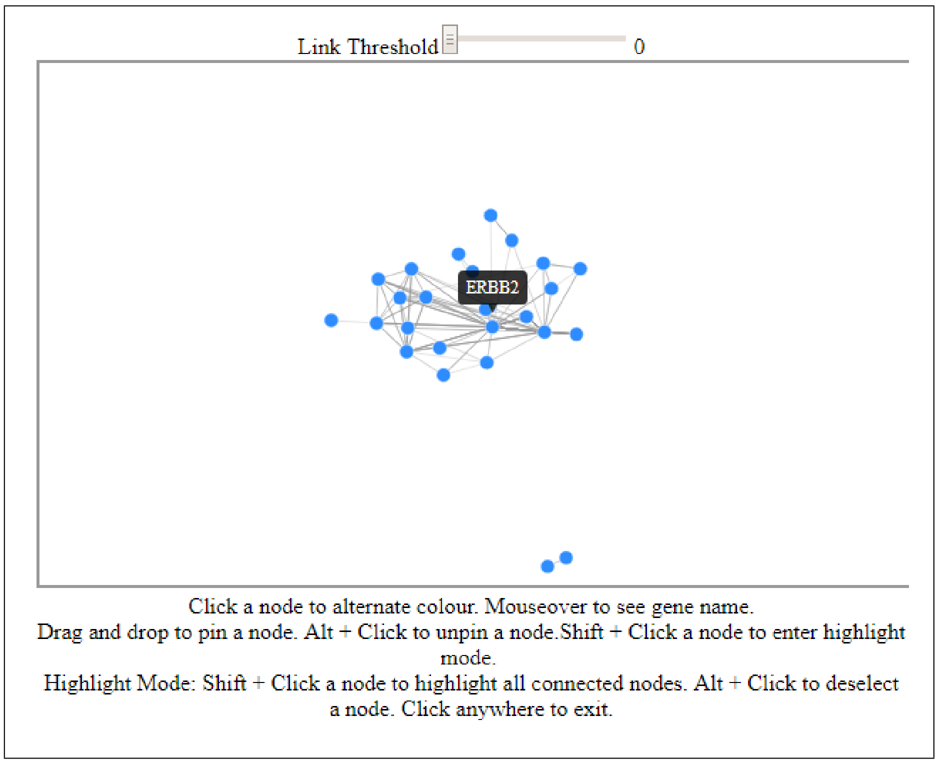
The initial version of the library contained enough code to parse a HTML form with user input, and use this input to call the StringDB API. The StringDB request returned a JSON array of interactions, from which the library also generated an array of genes. The arrays of genes and their interactions were then passed to D3.js which generated a force directed layout graph using these arrays as nodes and links.

The graph produced implemented various features to aid in network exploration. The graph was very dense so node labels (gene IDs) were only displayed on mouseover. It was also difficult at times to spot what nodes were connected due to the graph, so a feature was added that would highlight all nodes connected to a specified node. Nodes could also be dragged and pinned when exploring the graph. Nodes were initially coloured blue but would alternate colours when clicked to track visited nodes. Interactions could be filtered according to a minimum threshold confidence score by adjusting a sliding bar at the top of the graph.

This initial version also returned CORS errors in the browser and had to be run in Google Chrome with security disabled. CORS stands for Cross-Origin Resource Sharing and it requires APIs to specify in their response headers if they want to allow calls coming from browsers in other domains. Although the issue was initially marked by Chrome as an issue with CORS headers not being specified for requests coming from the "localhost" domain, StringDB didn't have CORS headers specified for any domains. I contacted the development team behind StringDB about the issue and they implemented CORS headers across all of their APIs following my feedback.

3.2.2 Feature Improvements:

The initial graph produced a highly dense graph with small unlabeled nodes. This made it very hard to explore large graphs and to find the appropriate genes. For this reason the nodes needed to be drawn larger, text labels had to be added and the strength of the forces used by the force directed layout had to be adjusted. Changing the node objects of the graph from circles to circles with labels broke a lot of my code regarding displaying nodes (e.g. updating their position in the graph according to the animation or changing the node color when clicked). This minor change actually ended up requiring a lot of refactoring of existing code. Once labels were added to the nodes it became difficult to identify which label was associated with which node so the library was then updated to color nodes in alternating colors according to a 3 color scheme.



Click a node to alternate colour. Mouseover to see gene name.
Drag and drop to pin a node. Alt + Click to unpin a node.Shift + Click a node to enter highlight mode.
Highlight Mode: Shift + Click a node to highlight all connected nodes. Alt + Click to deselect a node. Click anywhere to exit.

Figure 3.1: Visualisation of the Interaction Network of the ERBB2 Gene in the Initial Version of the Library

Development then progressed to offering further functionality to users. A function was added to allow users to filter what score was used to display links in the graph (i.e. to allow users to toggle between the combined score and the experimental score returned by StringDB). Autocomplete functionality was added as an option if users were specifying species as part of an input form for the API calls.

The modal component was added to give a pop-up window effect and 'onclick' events were added to the link and node objects to display detailed gene and interaction information inside the modal component. Interaction information was already stored from the result of the StringDB API call, but gene information needed to be retrieved from an external source. The node on click event calls a function which makes two requests to the Uniprot API, the first maps the StringDB ID to a Uniprot ID, the second request retrieves an XML file containing gene information for that Uniprot ID.

Since the NCBI website (which hosts the Entrez Gene database) seems to be the go to website for biologists when exploring genes, the Uniprot API is also used to map to the Entrez Gene ID and a link to the relevant page on the NCBI website is generated and added to the detailed node information pop-up window.

Once 'onclick' events were added to links my supervisor realised it was difficult to select the correct link in dense parts of the graph. Mouse over events were then added to links so that it was easier to hover over the correct link and so that users would know what link they were currently hovering over.

Once the modal element was added, it didn't make sense to have help text displayed underneath the graph as seen in figure 3.1. A help button was added to the 'toolbar' at the top of the graph and once clicked it displays the help text inside of the modal component.

When the modal component was first developed it was developed in such a way that the modal container would be filled with the relevant HTML content (e.g. the content generated from parsing the Uniprot gene information XML file), and the modal would then be displayed. However this lead to some delays between when the node was clicked and when the modal actually displayed, and allowed for a user to make multiple requests to Uniprot by clicking on a node a few times before the modal actually displayed. To overcome this the modal was redesigned so that it would be displayed instantly, with the contents replaced with a loading SVG animation. Once the HTML content for the modal is ready the SVG is then replaced with the appropriate content.

3.2.3 Configuration Options:

Once an interactive and usable graph was developed, work started on making that graph more configurable for users. Users could choose between using the BioGrid and StringDB APIs, and the functions to generate the graph and to display detailed node and interaction information were abstracted to support the differences in input coming from the two different API responses. Once this abstraction was performed it actually allowed for users to specify their own data for the graph and completely bypass calling either BioGrid or StringDB for network information.

To give users greater control over the graph style, options were added to allow users to choose a color scheme for the graph, or to specify their own coloring or sizing functions for graphs. Options were also added to support other styling elements, such as whether or not to center node labels or what elements to include in the graph 'toolbar'. This resulted in a highly configurable implementation of an interactive force directed graph with most of the features required by biologists to explore the graph.

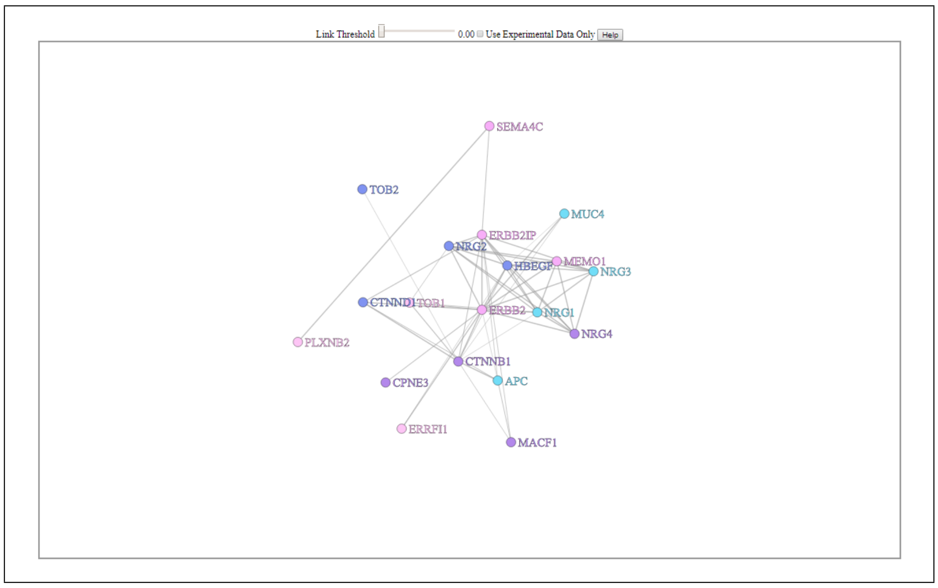


Figure 3.2: Visualisation of the Interaction Network of the ERBB2 Gene in the Current Version of the Library

In order to add to the library at this stage another graph layout was implemented. The hierarchical layout is an adapted version of D3's cluster layout. The cluster layout is normally used to represent trees in a graph, but in the hierarchical adaptation all nodes are drawn at the same level, and only leaf nodes are displayed. The hierarchical layout in this case is essentially a circular layout, with curved paths between each node. To make it easier to see interacting genes mousing over one node highlights all connected nodes. Clicking on a node will display detailed gene information similar to the force directed layout, but clicking on a path between nodes does not display interaction information due to the fact that in the hierarchical layout the path between nodes is a line object and not an instance of a link object as it is in the force directed layout.

Chapter 4: **Detailed Design and Implementation:**

This chapter is divided into two sections: gene interaction network data and graph drawing. The first section will discuss how user input is parsed, how the library retrieves gene interaction network data through API calls, and how the library uses another API for data enrichment to retrieve detailed gene information. The graph drawing section discusses the different layouts supported by the library, the features implemented in these graphs, and how client customisation of these graphs is supported.

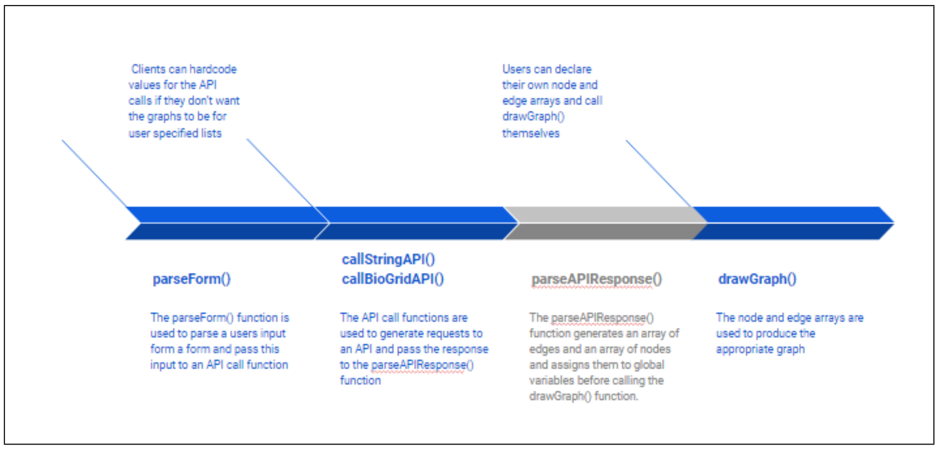
4.1 Gene Interaction Network Data:

4.1.1 Gene Selection:

The library supports three different ways of selecting genes for the network visualisation. The client can include a form on the page and pass the appropriate form element IDs to a parsing function in the library.

The parseForm() function in the library takes the ID of the form element containing the gene list, the ID of the form elemenent containing the species ID (can be null), a boolean flag for whether the StringDB or BioGrid API is to be called (default is BioGrid), and finally an additional argument 'arg', which for StringDB is a minimum confidence score (can be null), and for the BioGrid is a user identifier (required).

Alternatively, the client can hard-code these values and pass them to the API calls themselves, or completely bypass the API calls and use their own gene interaction data. The structure of the pipeline for passing information to the graph drawing function and the entry points for clients is shown in figure 4.1.



Clients can hardcode values for the API calls if they don't want the graphs to be for user specified lists

Users can declare their own node and edge arrays and call drawGraph() themselves

The parseForm() function is used to parse a users input form a form and pass this input to an API call function

The API call functions are used to generate requests to an API and pass the response to the parseAPIResponse() function

The parseAPIResponse() function generates an array of edges and an array of nodes and assigns them to global variables before calling the drawGraph() function.

The node and edge arrays are used to produce the appropriate graph

Figure 4.1: Pipeline for Passing Information to the Graph Drawing Function

4.1.2 API Calls for Network Retrieval:

The library contains both a callStringAPI() and a callBioGridAPI() function, which each take different parameters. The callStringAPI() function takes a list of proteins as a string, a species ID which defaults to 9606 (human), and a required score which defaults to 0 as arguments. The callBioGridAPI() function takes a list of proteins as a string, a species ID which defaults to 9606 (human), and a user access key (required) as arguments. Both functions generate the appropriate request based on the arguments use that request to call the API.

To aid with API calls, a protocol() function is also defined in the library to help create requests. If the current page is served by "file" or "HTTP" protocol the function will return HTTP, if the current page is served over HTTPS protocol the function will return HTTPS. This is used to avoid security errors in the browser when some content but not all is loaded using encryption.

Once the API call is returned, the data is passed to a parseAPIResponse() function which takes as arguments the reponse data and a flag for whether the data comes from StringDB or the BioGrid. The data is parsed into an array of edges (representing interactions between genes) by iterating over the interactions, with additional attributes added for graph drawing (specifically adding a source, target and value attribute).

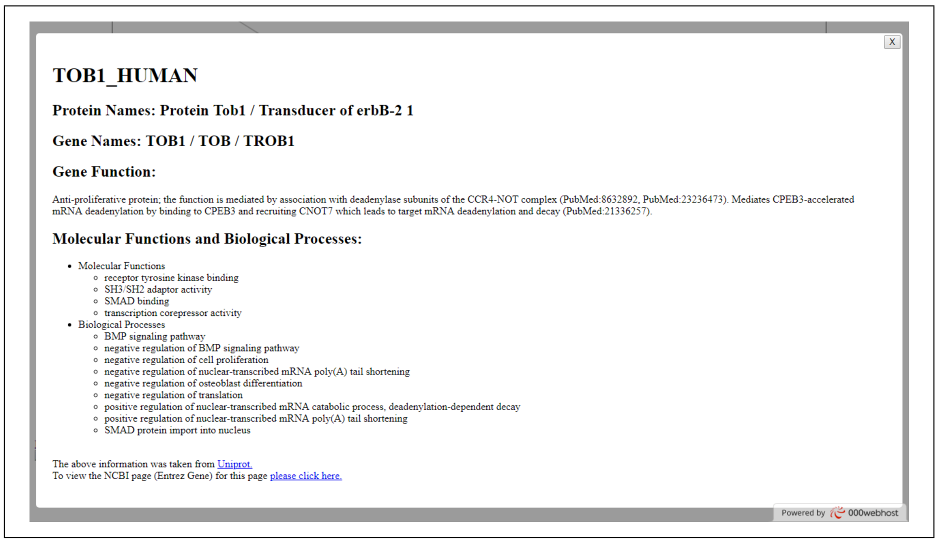
While iterating over the interactions a set of strings is generated using the Javascript 'Set' data structure. The strings added to the 'Set' are semicolon delimited values containing the gene name, internal gene ID and the organism ID for both StringDB and the BioGrid data. The parseAPIResponse() function then iterates over this set and splits these strings to create a unique array of nodes. If the array of nodes is not unique we will end up with several unconnected nodes and some unwanted behaviour in the graph. Once the array of nodes and the array of links are generated the graph can then be drawn.

4.1.3 Data Enrichment:

The displayNodeInformation() function takes a node as an argument and uses the Uniprot API to map the ID of this node (from StringDB/the BioGrid) to a Uniprot ID. If successful this Uniprot ID is then used to generate another request for detailed gene information from Uniprot which is returned in an XML format. This XML data is passed to a parseUniprotXML() function which parses the data to appropriate HTML content and fills the modal component with this content. The displayNodeInformation() function is triggered by node "onclick" Javascript events to request and display detailed gene information whenever a gene is clicked in the graph.

4.2 Graph Drawing:

The graph drawing function creates a responsive SVG element, and inside this SVG creates node and line DOM elements based on the input data (arrays of nodes and links). The only constraints placed on input data is that nodes must have an ID attribute and edges must have a value attribute, as well as source and target attributes that correspond to an existing node ID. In a force directed layout genes are represented by labeled circles connected by straight lines, and in a hierarchical layout genes are represented by labels placed in a circular layout and linked by curved paths.



X

TOB1_HUMAN

Protein Names: Protein Tob1 / Transducer of erbB-2 1

Gene Names: TOB1 / TOB / TROB1

Gene Function:

Anti-proliferative protein; the function is mediated by association with deadenylase subunits of the CCR4-NOT complex (PubMed:8632892, PubMed:23236473). Mediates CPEB3-accelerated mRNA deadenylation by binding to CPEB3 and recruiting CNOT7 which leads to target mRNA deadenylation and decay (PubMed:21336257).

Molecular Functions and Biological Processes:

- Molecular Functions
  - receptor tyrosine kinase binding
  - SH3/SH2 adaptor activity
  - SMAD binding
  - transcription corepressor activity
- Biological Processes
  - BMP signaling pathway
  - negative regulation of BMP signaling pathway
  - negative regulation of cell proliferation
  - negative regulation of nuclear-transcribed mRNA poly(A) tail shortening
  - negative regulation of osteoblast differentiation
  - negative regulation of translation
  - positive regulation of nuclear-transcribed mRNA catabolic process, deadenylation-dependent decay
  - positive regulation of nuclear-transcribed mRNA poly(A) tail shortening
  - SMAD protein import into nucleus

The above information was taken from Uniprot.
To view the NCBI page (Entrez Gene) for this page please click here.

Powered by 000webhost

Figure 4.2: Detailed Gene Information Generated for the TOB1 Gene

4.2.1 Force Directed Layout:

For the force directed layout three unique forces are defined and their strengths are set: a centering force that tries to force each node towards the center of the graph, a repelling force between each node, and an attraction force between connected nodes whose strength depends on the value for a given interaction. The graph is an animated self-balancing physics model that tries to balance these forces, and if the strength for each force is set appropriately we should end up with a graph that is centered, well-distributed and which has connected nodes drawn closer together.

The force directed layout is drawn by the drawGraph() function of the library. The strengths for the three forces are set and stored in the global variable 'simulation'. The SVG DOM element is created and stored in the global variable 'svg' and node and link objects are created in this SVG by reading data from the global variables 'nodes' and 'links'.

The ticked() function is called regularly by D3 to animate the graph. Each 'tick' updates the positions of nodes and links in the graph to the current position assigned to them by the simulation. Global variables are needed to allow the various functions such as ticked() to access these variables since we can't explicitly pass them as arguments.

Graph Features:

The force directed layout has a lot of additional features added to extend the basic functionality offered by the D3 library. Custom functions have been added to allow users to drag and pin nodes using the mouse, or to Alt + Click to unpin a node. Users can highlight connected nodes through Shift + Click on a particular node. Clicking on a node or link will display detailed gene or gene interaction information in a pop up modal, and mousing over a link will draw the link thicker and with a darker colour to make it easier for users to click on a given link.

There are also functions to add a toolbar containing any of the following: a help button, a sliding bar to adjust the minimum score threshold for links, and a filter to allow users to change which score is used to draw links (e.g. for StringDB API calls users can change between using the combined score or the experimental score only).

4.2.2 Hierarchical Layout:

For the hierarchical layout a 'root' node is added to the existing array of nodes passed to the library. A "parent" attribute is added to each existing node, and the value of this attribute is the ID of the root node. An "interactions" attribute is also added to each node, which is an array containing the IDs of all nodes that interact with that particular node. D3 contains a "stratify" function that needs to be called on the updated array of nodes, which automatically selects the root node and sorts the remaining nodes according to their position in the hierarchy. Once this has been performed a customised cluster diagram is then drawn with curved lines between all nodes.

The resulting graph displays each gene at regular intervals in a circular layout. The Javascript "mouseover" event for a node triggers a function that highlights all connected nodes (and the links connecting to those nodes). The Javascript "onclick" event for a node triggers a function that displays detailed node information in a pop up modal, similar to the force directed layout.

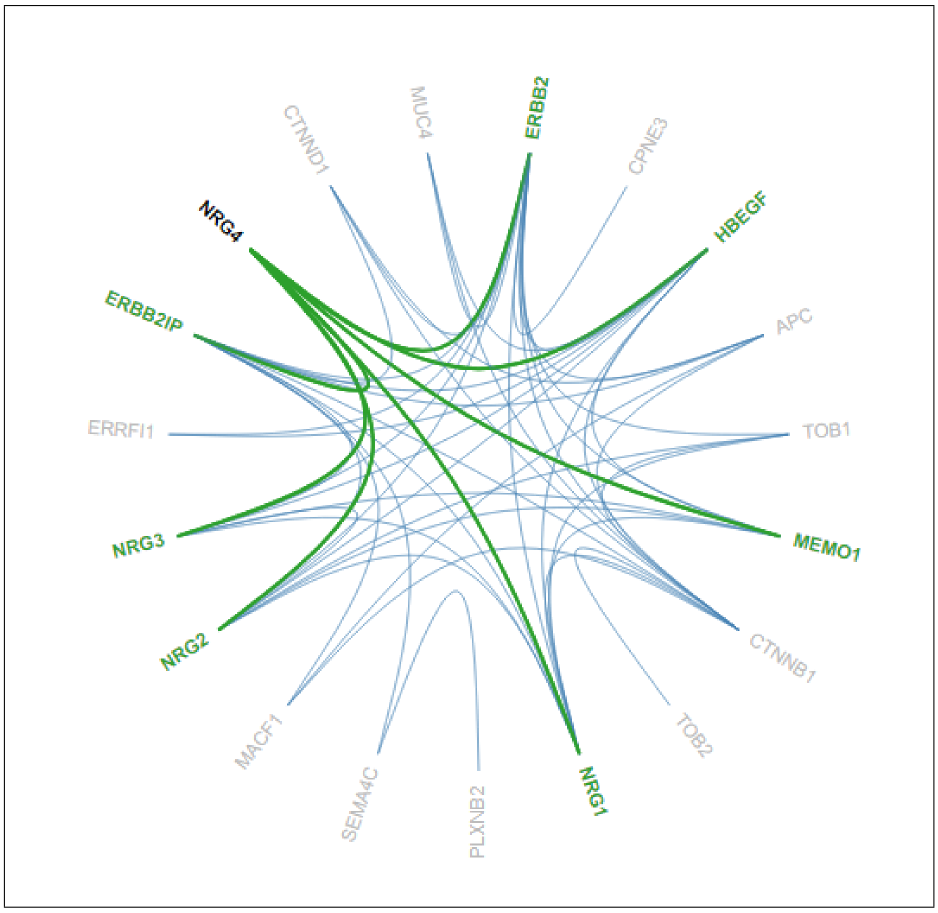
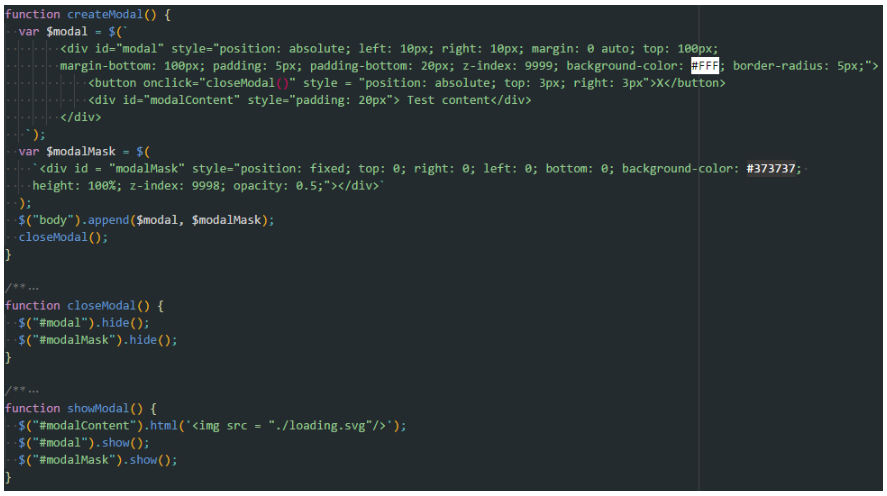


Figure 4.3: Visualisation of the Interaction Network of the ERBB2 Gene using the Hierarchical Layout

4.2.3 Modal Component:

In order to allow various information screens to pop up the library provides a 'modal' component. The component consists of a modal and a modal mask. The modal mask is a div element that is placed over the entire web page, and is coloured grey with low opacity. This gives the desired effect of the web page appearing greyed out. The modal sits in front of the modal mask and is a closable container containing the relevant HTML. The library contains a createModal() function to create the relevant HTML components which are not displayed when the page is first loaded, and a closeModal() function that hides the modal components whenever the close button in the modal is clicked.

The library also contains a "showModal()" function that displays the modal and the modal mask, placing a loading SVG icon in the modal to avoid old information being shown. The modal is then filled with the relevant information asynchronously which will overwrite the loading SVG icon. Once the modal is closed the web page returns to its normal state.

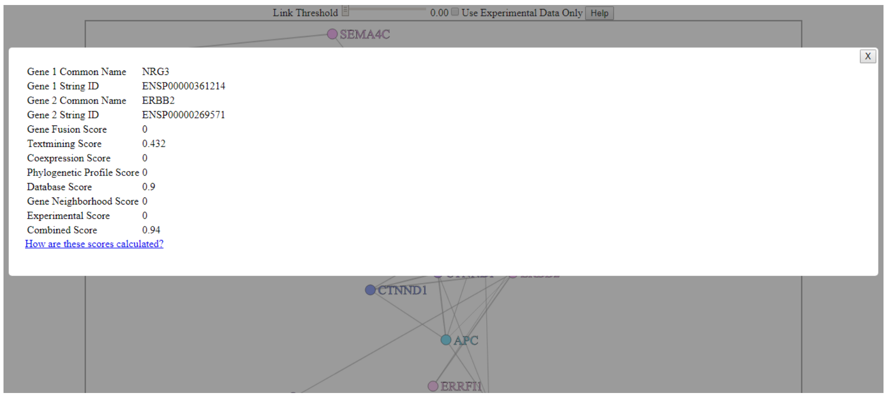


```
function createModal() {
  var $modal = $(`
        <div id="modal" style="position: absolute; left: 10px; right: 10px; margin: 0 auto; top: 100px;
        margin-bottom: 100px; padding: 5px; padding-bottom: 20px; z-index: 9999; background-color: #FFF; border-radius: 5px;">
            <button onclick="closeModal()" style = "position: absolute; top: 3px; right: 3px">X</button>
            <div id="modalContent" style="padding: 20px"> Test content</div>
        </div>
    `);
  var $modalMask = $(
     `<div id = "modalMask" style="position: fixed; top: 0; right: 0; left: 0; bottom: 0; background-color: #373737;
     height: 100%; z-index: 9998; opacity: 0.5;"></div>`
  );
  $("body").append($modal, $modalMask);
  closeModal();
}

/**...
function closeModal() {
  $("#modal").hide();
  $("#modalMask").hide();
}

/**...
function showModal() {
  $("#modalContent").html('<img src = "./loading.svg"/>');
  $("#modal").show();
  $("#modalMask").show();
}
```

Figure 4.4: Source Code for the Implementation of the Modal Component



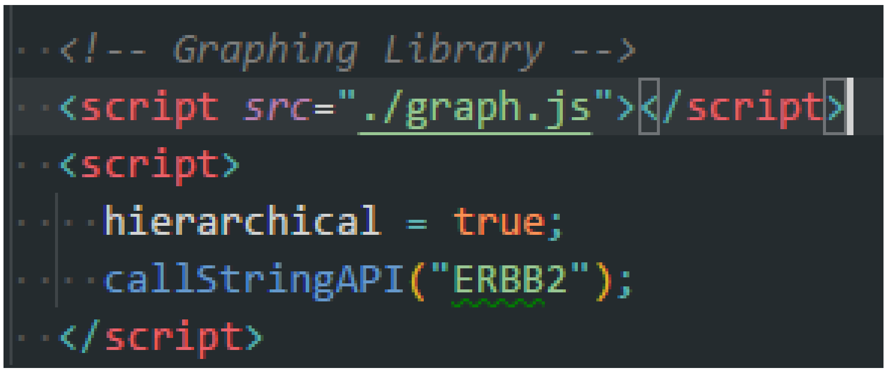
Gene 1 Common Name	NRG3
Gene 1 String ID	ENSP00000361214
Gene 2 Common Name	ERBB2
Gene 2 String ID	ENSP00000269571
Gene Fusion Score	0
Textmining Score	0.432
Coexpression Score	0
Phylogenetic Profile Score	0
Database Score	0.9
Gene Neighborhood Score	0
Experimental Score	0
Combined Score	0.94

How are these scores calculated?

Figure 4.5: Interaction Information Displayed in the Modal Component

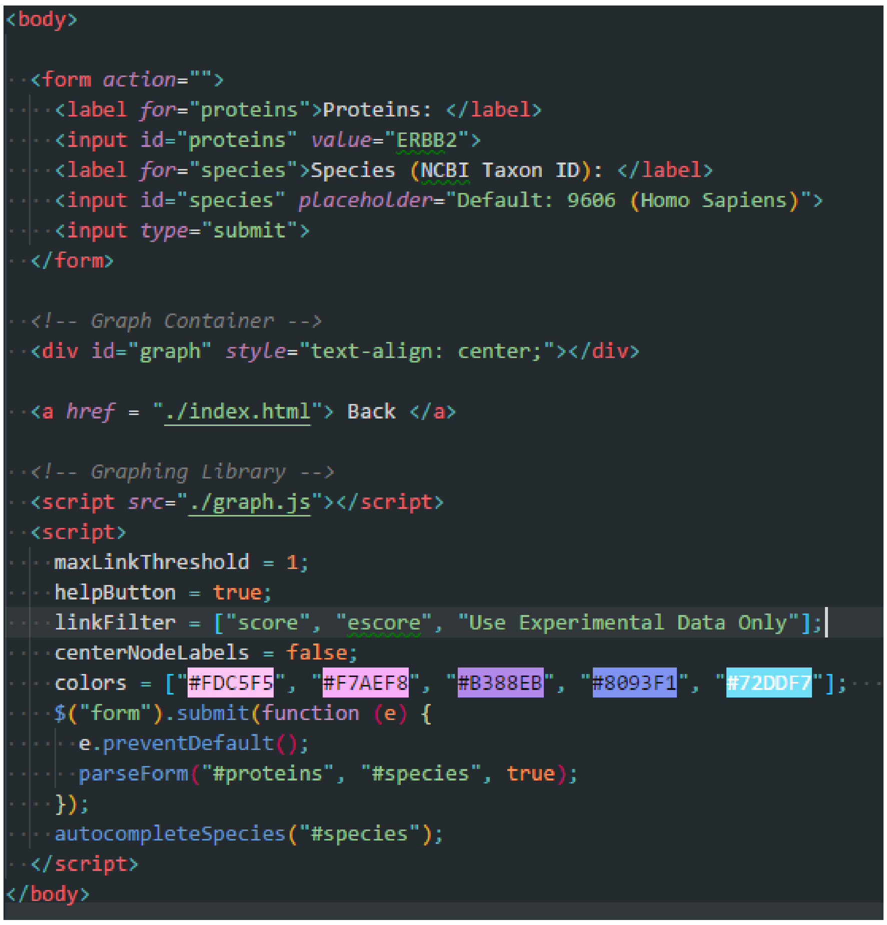
4.2.4 Graph Customisation:

The use of global variables couldn't be avoided in this project since they are needed by D3 to pass graph components (e.g. the node and line objects drawn in the graph) between various functions. Since global variables were necessary for graph drawing, global variables are also used for graph configuration and customisation to avoid having to pass a large number of arguments to the library functions. The table above details the variables that can be set to change the appearance of the graph. The global variables needed for graph drawing functions can also be accessed by advanced users to completely alter the graph, including setting custom strengths to the forces used in the force directed layout, or changing how nodes or links are represented in a graph.



```
  <!-- Graphing Library -->
  <script src="./graph.js"></script>
  <script>
    hierarchical = true;
    callStringAPI("ERBB2");
  </script>
```

Figure 4.6: Source Code Required to Produce Graph Shown in Figure 4.3



```
<body>

  <form action="">
    <label for="proteins">Proteins: </label>
    <input id="proteins" value="ERBB2">
    <label for="species">Species (NCBI Taxon ID): </label>
    <input id="species" placeholder="Default: 9606 (Homo Sapiens)">
    <input type="submit">
  </form>

  <!-- Graph Container -->
  <div id="graph" style="text-align: center;"></div>

  <a href = "./index.html"> Back </a>

  <!-- Graphing Library -->
  <script src="./graph.js"></script>
  <script>
    maxLinkThreshold = 1;
    helpButton = true;
    linkFilter = ["score", "escore", "Use Experimental Data Only"];
    centerNodeLabels = false;
    colors = ["#FDC5F5", "#F7AEF8", "#B388EB", "#8093F1", "#72DDF7"];
    $("form").submit(function (e) {
      e.preventDefault();
      parseForm("#proteins", "#species", true);
    });
    autocompleteSpecies("#species");
  </script>
</body>
```

Figure 4.7: Source Code Required to Produce Graph Shown in Figure 3.2

Chapter 5: **Testing and Evaluation:**

Since this library was developed using Javascript it doesn't need any special software or testing environment. Notepad++ was initially used for development but as the library grew a better IDE was needed. I used Visual Code Studio to help keep consistent formatting. Visual Code Studio also has a plugin that displays colours used in the source code which was a major advantage when working on design aspects of the project. The library is stored in a single Javascript file with a website showcasing it built around it. Git was used for version control, and Bintray is being considered as a CDN if the library can be made open source.

Testing was done extensively in the browser (Google Chrome V65 on Windows 10) using the showcase website. Testing was done using the files stored on my local machine but the showcase website is currently hosted online at `http://14433252fyp.000webhostapp.com`. Some light testing was done on FireFox on Windows 10 and Safari on Mac as well. Selenium was considered to automate testing using Mozilla Firefox as a browser, but was ultimately considered unnecessary since most of the work going in to the project has been largely cosmetic since it is a visualisation based project.

In order to properly evaluate my project I would have to find a considerable number of biologists in this particular field who would be willing to use the showcase website and then complete a feedback form evaluating the website. Unfortunately I didn't have the resources to complete this type of evaluation. However my supervisor did offer plenty of feedback as a potential user, which lead to the implementation of many changes to the graph, including changes to the color scheme and animating the mouseover of links to make it easier to select the correct link when exploring the graph.

Chapter 6: **Conclusions and Future Work:**

Conclusions:

The library produced is capable of producing highly interactive and customisable graphs using minimal amounts of code. Clients can choose custom node sizes and colours and they currently have a choice of two graph drawing algorithms. A website has been developed to demonstrate the libraries functionality, and so all core specifications for the project have been successfully fulfilled.

The advanced specification for this project was to implement this library with `http://cancergd.org`. Although the library has not been implemented with CancerGD directly, it has been implemented on another instance of CancerGD running at `http://cgenetics.pythonanywhere.com/` and so I would consider this advanced specification to also be fulfilled.

The library developed does have some limitations, based on two contributing factors: a limit on data returned by API calls, and the computing power of the users machine. The BioGrid will only return up to 10,000 interactions, and although StringDB recommend using POST requests for larger requests, sending requests to StringDB as POST requests sometimes resulted in empty return values. The issue turned out to be a difference in how gene identifiers were mapped between GET and POST requests. Although we are constrained in terms of the size of the API calls we can make, in practice this number is much higher than what most users computers can handle. In most machines the animation of the force directed layout will start to lag to an unusable level at around 2000-3000 nodes [23], and the hierarchical layout will either be too dense or have too large a diameter to facilitate exploration by this number of nodes.

Future Work:

My hope for this project in the future would be to make it open source and to host it on a free content delivery network such as Bintray to make it available to the public. If this became open source then a formal set of tests using Selenium would need to be developed to ensure any changes didn't cause any problems with any of the libraries functionality.

A lot of improvements could also be made on the implementation of the library on the CancerGD instance. It was difficult to implement even though the library requires very little set up because CancerGD is a Python based web application and the owner of CancerGD required the graph to display in a pop up container, which was difficult to develop in the existing web application. Had time not been a constraint I would have liked to improve this implementation to the point where the owner of CancerGD would have been satisfied to include it on the real CancerGD website.

More layouts could be added to the library in the future to give clients more choice for their graphs, and the data enrichment for genes could be improved by using the Uniprot API to map gene IDs to IDs from more external sites. This would allow the library to access more data sources for gene information instead of using only Uniprot for data enrichment.

Bibliography

[1] Alchemy.js, . URL http://graphalchemist.github.io/Alchemy/#/docs.

[2] API Documentation, . URL https://www.ensembl.org/info/docs/api/index.html.

[3] arbor.js reference, . URL http://arborjs.org/reference.

[4] BioGRID REST Service | BioGRID, . URL https://wiki.thebiogrid.org/doku.php/biogridrest.

[5] BioGRID Partners | BioGRID, . URL https://wiki.thebiogrid.org/doku.php/partners.

[6] BioGrid API Call for a Single Interaction, . URL http://webservice.thebiogrid.org/interactions/?searchNames=true&geneList=MDM2&taxId=9606&includeInteractors=true&includeInteractorInteractions=true&accesskey=0e9267057ee3b4250c265745d40a92dd&format=json&max=1.

[7] BioJS, . URL http://biojs.net/.

[8] @ember/application - 2.17 - Ember API Documentation, . URL https://emberjs.com/api/ember/2.17/modules/@ember%2Fapplication.

[9] Ensembl Rest API - Ensembl REST API Endpoints, . URL https://rest.ensembl.org/.

[10] Frequently Asked Questions | BioGRID, . URL https://wiki.thebiogrid.org/doku.php/network_viewer:frequently_asked_questions.

[11] Help/STRING: functional protein association networks, . URL http://string-db.org/cgi/help.pl?subpage=api.

[12] How can I access resources on this website programmatically?, . URL http://www.uniprot.org/help/api.

[13] STRING: functional protein association networks, . URL https://string-db.org/cgi/about.pl.

[14] vis.js - Network documentation., . URL http://visjs.org/docs/network/.

[15] D3.js Tooltips by VACLab, October 2017. URL https://github.com/VACLab/d3-tip. original-date: 2016-07-21T00:24:00Z.

[16] d3: Bring data to life with SVG, Canvas and HTML., December 2017. URL https://github.com/d3/d3. original-date: 2010-09-27T17:22:42Z.

[17] Jac C. Charlesworth, Joanne E. Curran, Matthew P. Johnson, Harald HH Gring, Thomas D. Dyer, Vincent P. Diego, Jack W. Kent, Michael C. Mahaney, Laura Almasy, Jean W. MacCluer, Eric K. Moses, and John Blangero. Transcriptomic epidemiology of smoking: the effect of smoking on gene expression in lymphocytes. *BMC Medical Genomics*, 3:29, July 2010. ISSN 1755-8794. doi: 10.1186/1755-8794-3-29. URL https://doi.org/10.1186/1755-8794-3-29.

[18] Andrew Chatr-aryamontri, Rose Oughtred, Lorrie Boucher, Jennifer Rust, Christie Chang, Nadine K. Kolas, Lara O'Donnell, Sara Oster, Chandra Theesfeld, Adnane Sellam, Chris Stark, Bobby-Joe Breitkreutz, Kara Dolinski, and Mike Tyers. The BioGRID interaction

database: 2017 update. *Nucleic Acids Research*, 45(Database issue):D369–D379, January 2017. ISSN 0305-1048. doi: 10.1093/nar/gkw1102. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5210573/.

[19] Max Franz, Christian T. Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D. Bader. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2): 309–311, January 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv557. URL https://academic.oup.com/bioinformatics/article/32/2/309/1744007.

[20] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by forcedirected placement. *Software: Practice and Experience*, 21(11):1129–1164, November 1991. ISSN 1097-024X. doi: 10.1002/spe.4380211102. URL http://onlinelibrary.wiley.com/doi/10.1002/spe.4380211102/abstract.

[21] Mileidy W. Gonzalez and Maricel G. Kann. Chapter 4: Protein Interactions and Disease. *PLOS Computational Biology*, 8(12):e1002819, December 2012. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1002819. URL http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002819.

[22] Alexis Jacomy. sigma.js: A JavaScript library dedicated to graph drawing, December 2017. URL https://github.com/jacomyal/sigma.js. original-date: 2012-03-11T15:17:40Z.

[23] Curran Kelleher. HTML5examples: Examples of HTML5 in action (best results in Chrome), February 2018. URL https://github.com/curran/HTML5Examples. original-date: 2012-08-01T21:16:07Z.

[24] Eric Sayers. *E-utilities Quick Start*. National Center for Biotechnology Information (US), November 2017. URL https://www.ncbi.nlm.nih.gov/books/NBK25500/.

[25] Matthew Suderman and Michael Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, October 2007. ISSN 1367-4803. doi: 10.1093/bioinformatics/btm401. URL https://academic.oup.com/bioinformatics/article/23/20/2651/229837.

[26] Damian Szklarczyk, John H Morris, Helen Cook, Michael Kuhn, Stefan Wyder, Milan Simonovic, Alberto Santos, Nadezhda T Doncheva, Alexander Roth, Peer Bork, Lars J. Jensen, and Christian von Mering. The STRING database in 2017: quality-controlled proteinprotein association networks, made broadly accessible. *Nucleic Acids Research*, 45(Database issue):D362–D368, January 2017. ISSN 0305-1048. doi: 10.1093/nar/gkw937. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5210637/.

[27] Rui Wang, Yasset Perez-Riverol, Henning Hermjakob, and Juan Antonio Vizcano. Open source libraries and frameworks for biological data visualisation: A guide for developers. *Proteomics*, 15(8):1356–1374, April 2015. ISSN 1615-9853. doi: 10.1002/pmic.201400377. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4409855/.

[28] Richard Wooster, Graham Bignell, Jonathan Lancaster, Sally Swift, Sheila Seal, Jonathan Mangion, Nadine Collins, Simon Gregory, Curtis Gumbs, Gos Michlem, Rita Barfoot, Rifat Hamoudi, Sandeep Patel, Catherine Rice, Patrick Biggs, Yasmin Hashim, Amanda Smith, Frances Connor, Adelgeir Arason, Julius Gudmundsson, David Ficenec, David Kelsell, Deborah Ford, Patricia Tonin, D. Timothy Bishop, Nigel K. Spurr, Bruce A. J. Ponder, Rosalind Eeles, Julian Peto, Peter Devilee, Cees Cornelisse, Henry Lynch, Steven Narod, Gilbert Lenoir, Valdgardur Egilsson, Rosa Bjork Barkadottir, Douglas F. Easton, David R. Bentley, P. Andrew Futreal, Alan Ashworth, and Michael R. Stratton. Identification of the breast cancer susceptibility gene BRCA2. *Nature*, 379(6567):749, February 1996. ISSN 1476-4687. doi: 10.1038/379749a0. URL https://www.nature.com/articles/379749a0.