A Method for Connectivity Map  
Query Result Refinement and Visualization

Theodore E. Natoli

A Thesis in the Field of Biotechnology  
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University  
September 2014

Abstract

The Connectivity Map (CMap) is a database of gene expression signatures obtained from experiments in which cultured human cells are treated with pharmacologic and genomic perturbagens. A typical use case of this database is for a researcher to query with a signature of a cell state of interest and use the matching perturbagens to develop a functional hypothesis for follow-up. Current pattern matching algorithms that perform CMap queries suffer from a universal weakness – the enormous size and richness of signatures in CMap means that a query typically generates hundreds of strong connections. These connections are hard to distinguish, thereby making prioritization difficult. An interconnectivity-based method of query result refinement, whereby query results that are highly interconnected amongst themselves are highlighted over singletons, proves an effective solution to the prioritization problem. To implement this method, I built a web-based tool called QViz that displays CMap query results visually in a graph layout and helps identify highly interconnected sub-groups of signatures. Using the QViz algorithm and a set of curated queries, I have verified that biologically relevant queries yield results that are more interconnected than control or null queries. Additionally, QViz has able to characterize queries originating from different areas of biology in terms, highlighting the algorithm’s potential as a tool in understanding and further prioritizing CMap query results.

Table of Contents

List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

I. Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

The Research Problem . . . . XX

Gene Expression Profiling . . . . XX

Gene Set Enrichment Analysis (GSEA) . . . . XX

The Connectivity Map . . . . XX

Computing Connections in the Connectivity Map . . . . XX

Graphical Depictions of Biological Phenomena . . . . XX

II. Methods . . . . XX

Computing and Visualizing Connections . . . . XX

Estimating Significance . . . . XX

Other Application Features . . . . XX

Software Components . . . . XX

Front End: HTML & D3.js . . . . XX

Back End: MongoDB, R, & Node.js . . . . XX

III. Results . . . . XX

Interconnectivity Varies by Query Result . . . . XX

IV. Summary and Conclusion . . . . XX

Appendix A Source Code . . . . XX

References . . . . XX

List of Figures

1. An Example Graph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

2. QViz Application Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

3. Example Clique Densities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

4. Application Data Flow Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

5. Graphs and Clique Densities for Example Sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . XX

Chapter I

Introduction

The CMap database, built and maintained at The Broad Institute, is a compendium of gene expression signatures resulting from the treatment of cultured human cells with perturbations such as small molecule compounds (CP), short hairpin RNAs (shRNA), or over-expression constructs (OE). The utility of the CMap database is that of a gene expression search engine. Users are able to pose questions about relationships between cellular states and formulate hypotheses based on similarities or differences in the states’ gene expression signatures, or the sets of genes that are differentially regulated by a particular perturbation.

The Research Problem

Hypotheses are generated by posing search queries into the CMap database and examining the query results. A CMap query is a focused question in which a user inputs a gene expression signature, called the query, and computes the similarity, or connectivity, between his/her query and other signatures in the database. Positive connectivity indicates that two signatures’ expression changes are similar and vice versa. Researchers can use CMap to find connections between signatures within or external to the database. Hypotheses may be in the form of “the shRNA knockdown of gene X connects to shRNA knockdown signatures of pathway Y members, so X is probably a member of Y.” Or perhaps “the signature of compound Z connects to the knockdown signature of gene X, so perhaps X is the target of Z.”

Lamb et al. demonstrated a more directly therapeutically relevant use of the original incarnation of CMap when they discovered that the signature of sirolimus connected strongly to a signature of dexamethasone sensitivity. Dexamethasone is a treatment for acute lymphoblastic leukemia (ALL), but many patients eventually become resistant to it. The CMap connection between sirolimus and dexamathasone sensitivity suggested that sirolimus might be effective in reversing resistance in ALL patients who had become resistant to dexamethasone. A follow-up experiment confirmed that sirolimus conferred dexamathasone sensitivity to CEM-c1 cells, a previously dexamethasone-resistant cell line (Lamb et al., 2006).

The CMap database contains over 400,000 signatures spanning over 70 cell types. Because of this large size, interpreting and prioritizing query results has become difficult. For example, accepting only the top one percent of connections yields nearly 4,000 signatures. Follow-up on such a large number of primary hits is nearly impossible in most cases. However, within a set of initial query results, there frequently exists a set or sets of signatures that are more tightly interconnected with themselves than with other signatures. These interconnected sets are more likely to be indicative of robust biological signal and should therefore be prioritized over other singleton connections. The goal of this work was to build a web-based tool to implement an algorithm to identify subsets of high interconnectivity within lists of initial query results and to visualize the relationships between these subsets in a graph layout. This tool is useful in refining initial CMap query results into smaller, more actionable lists of connections that can be further investigated in secondary assays.

Gene Expression Profiling

Gene expression profiling is the simultaneous measure of the RNA transcript levels of many genes within a cell or group of cells. These measurements can help to provide insight into the cellular state or states of the cells in question. For example, if many cell-cycle genes are observed to be active, it could suggest that the cells are actively dividing. Conversely, if many apoptotic genes are active, the cells might be dying. Frequently, the goal of gene expression profiling is to identify genes that are differentially regulated between one or more sets of conditions. For example, one might measure expression in cells that have and have not been treated with a drug of interest, and then compare the resulting expression profiles to identify genes that are substantially up- or down-regulated in the treated cells relative to the untreated. Current technologies, such as the microarray, allow for many such gene expression experiments to be run in parallel, enabling the comparative analysis of hundreds or thousands of expression profiles corresponding to an equal number of experimental conditions. Similarly, expression profiling can be used to identify genes differentially regulated between disease and normal states. van’t Veer et al. used gene expression profiling to identify a set of genes that were predictive of breast cancer metastasis (van’t Veer et al., 2002). Because of its ability to identify such signatures, gene expression profiling is a powerful and often-used tool in contemporary biology.

Gene Set Enrichment Analysis (GSEA)

Gene Set Enrichment Analysis (GSEA) is an analytical approach designed to extract biological insight from gene expression data (Subramanian et al., 2005). It leverages groups of genes, called gene sets, that share some biological commonality (i.e. members of a cellular signaling pathway) and computes their enrichment, or trend towards the top or bottom, of a ranked list of genes generated by comparing expression profiles across two experimental classes (i.e. tumor vs. normal). For example, one might define many sets of genes, each corresponding to a cellular pathway. One could then rank-order all genes by their differential expression when comparing profiles of tumor vs. normal samples. Lastly, one could compute the enrichment of each pathway in the rank-ordered list to attempt to identify pathways that might be active in the particular tumor in question. Mechanically, GSEA computes a Kolmogorov-Smirnov (KS) statistic when comparing a given gene set to a given ranked list (Hollander & Wolfe, 1975). Effectively, this amounts to walking down the ranked list, increasing a running-sum statistic when one encounters a gene in the gene set and decreasing it for genes not in the gene set. The magnitude of the increment depends on the correlation of the gene with the phenotype. The enrichment score is the maximum deviation from zero encountered in the walk (Subramanian et al., 2005). GSEA has been used extensively for identifying coherent sets of genes that are collectively modulated under certain disease states and/or experimental conditions. In fact, a GSEA software suite and an accompanying online database exist to facilitate comparisons between novel and curated gene sets (Subramanian et al., 2005).

The Connectivity Map

The Connectivity Map (CMap) is a database containing the gene expression signatures resulting from treating cultured cells with various chemical and genomic perturbations (Lamb et al., 2006). The purpose of CMap is to serve as a lookup table of functional annotation. These annotations might be derived by comparing signatures within the CMap database or by querying the database with externally generated signatures. The database itself can be thought of as a large matrix where each row is a gene and each column is an experiment in which a particular perturbagen was profiled under a given set of conditions (i.e. cell context, dose, treatment time, etc.). The values in the matrix are differential expression measures generated by comparing the expression levels of the genes across perturbed and control states. Thus, each column of the matrix can be thought of as a given perturbagen’s expression signature.

Computing Connections in the Connectivity Map

A primary use of the CMap database is to compare the signatures of different perturbations and to assess their similarity. Perturbagens that, when used to treat cultured cells, result in similar gene expression consequences will yield similar CMap signatures. Such signatures are said to be positively connected in the CMap sense. Conversely, perturbagens that elicit inversely related expression consequences are said to be negatively connected. For a given query signature Q and a reference signature R, the weighted connectivity score (WTCS) is computed by computing and integrating two KS statistics, one each for the n most up- and down-regulated genes in Q. The algorithm proceeds as follows:

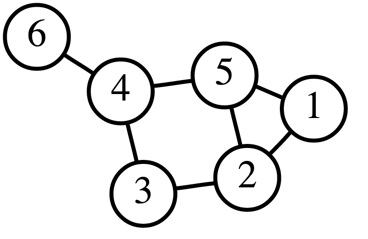
1. Order Q
2. compute ESup as the enrichment of the n most up-regulated genes in R
3. compute ESdn as the enrichment of the n most down-regulated genes in R
4. compute WTCS as
   1. 0 if ESup and ESdn share the same sign
   2. where the resulting WTCS is given the sign of ESup if ESup and ESdn are of different signs.

WTCS will be positive for signatures that are positively related and negative for those that are inversely related.

A common CMap use case is to select a given query signature Q from the database and compute its similarity to all other signatures. The remaining signatures can be ranked according to their connection strength with Q. The connections can be used to gain insight and form hypotheses about Q. For example, if Q is a signature of a novel compound and it connects strongly to signatures of compounds of a known pharmacological class, one might hypothesize that the novel compound is also a member of this class. Similarly, if Q connects strongly to the knockdown signature of gene G, one might hypothesize that G is the novel compound’s target. Q need not be a signature from the CMap database. For instance, it might be the signature of some disease and one might seek connections to genes whose modulation could be causing the disease or to compounds that could have therapeutic relevance.

Graphical Depictions of Biological Phenomena

In the fields of computer science and mathematics, a graph is a means by which to represent a set of objects and the relationships between them. It is frequently depicted as a set of nodes, where each node represents an object. Connections, where they exist, are represented by edges (Figure 1).



*Figure 1.* An Example Graph. The six nodes are connected by seven edges. All nodes other than node 6 have at least two edges. An edge between two nodes indicates an interaction or relationship between the nodes.

Although they originated in other fields, graphs have frequently been used as tools to model biological phenomena. Graphical models of protein interaction networks, gene expression networks, and other similar phenomena are commonplace. Friedman used graphical models to infer and visualize gene regulatory networks (Friedman, 2004). Lage et al. used graphical models to characterize existing and elucidate novel protein-protein interaction networks (Lage et al., 2007). Because of its widespread use and adoption, the graphical model is an appropriate, familiar, and effective means to depict connectivity between CMap perturbagens. In the graph visualization generated by QViz, CMap perturbagens are represented as nodes. Where a connection exists between to perturbagens, a line is drawn connecting their nodes. Thus, the user is able to easily see which perturbagens are connected to each other and can easily identify highly interconnected subsets of perturbagens, if they exist.

Chapter II

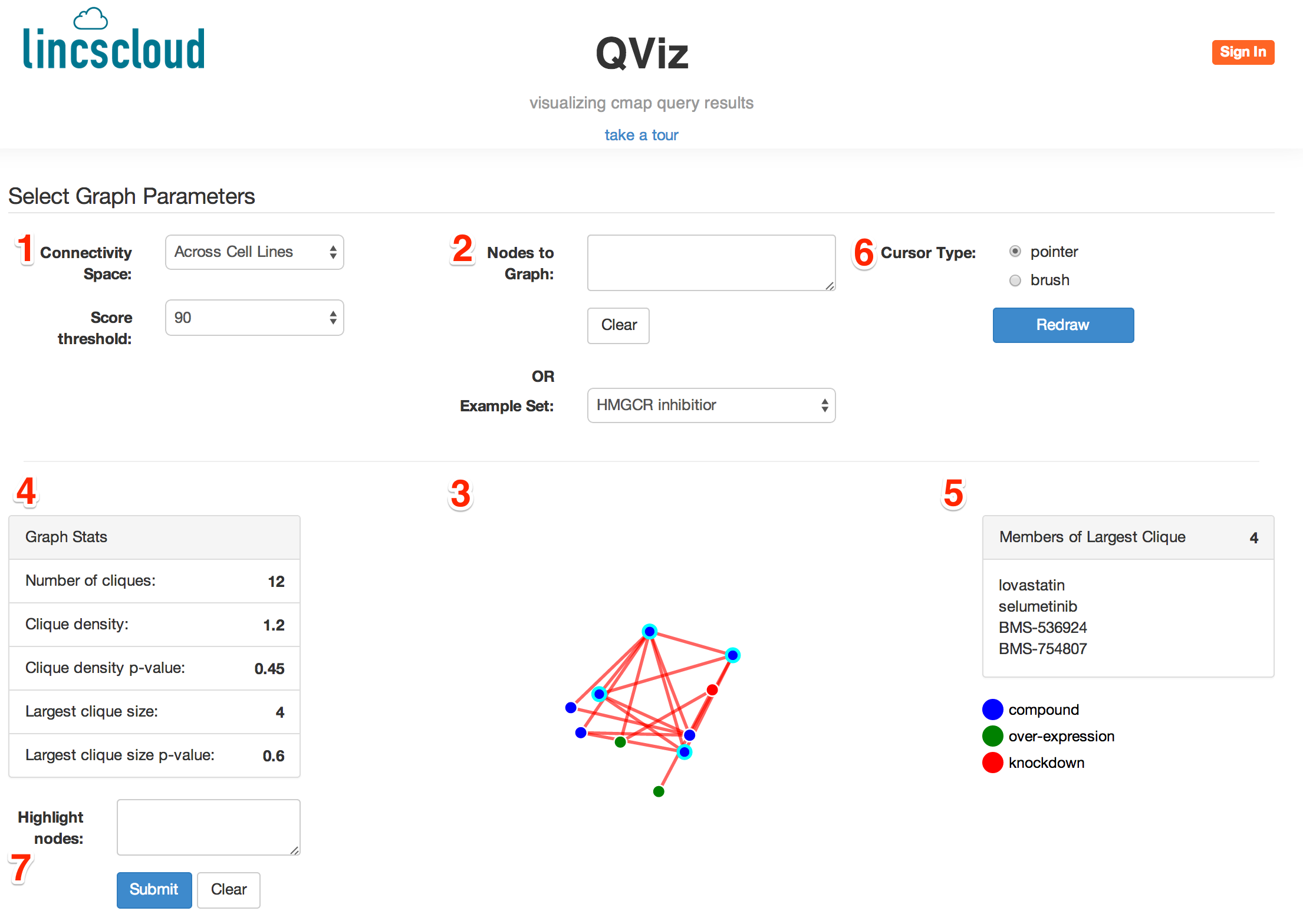
Methods

The following sections describe the implementation of the QViz application and how scientists and researchers might use it to investigate and characterize their CMap query results.

Computing and Visualizing Connections

All connections between CMap signatures were computed using WTCS. These connections were then summarized using the summly algorithm, which scores connections based on their consistency across multiple cell contexts (Subramanian et al, publication pending). The WTCS scores are converted to a rank point scale, or a normalized percentile rank, rescaled to range between -100 and 100, where 100 is completely positively connected and 100 is completely negatively connected. Connections that are maintained across many cell types receive a higher magnitude rankpoint than those that occur in only a small number. An additional benefit of applying the summly algorithm is that it abstracts away the concept of cell type and provides a more perturbagen-centric view of connectivity, thus dramatically reducing the connectivity space. The CMap database contains over 400,000 signatures, but that is reduced to a little over 7,000 perturbagens when considering only those that produced repeatable signatures in enough cell lines, in this case 4, to be considered for summly. The summly scores were precomputed and are stored in a database that the application uses to simply look up connectivity scores instead of computing them on-the-fly. Users interact with the application by inputting a list of perturbagens that have resulted from running a CMap query. QViz then searches the database for the summly scores that exist between all pairwise combinations of these perturbagens. QViz displays the perturbagens as nodes in a graph. Where the summly score between two nodes exceeds a given threshold, the application asserts that a connection exists between the perturbagens, and a line is drawn between them. Users are able to tune the summly score threshold to achieve the desired level of stringency in connection calling. Figure 2 below provides an overview of the QViz application interface.

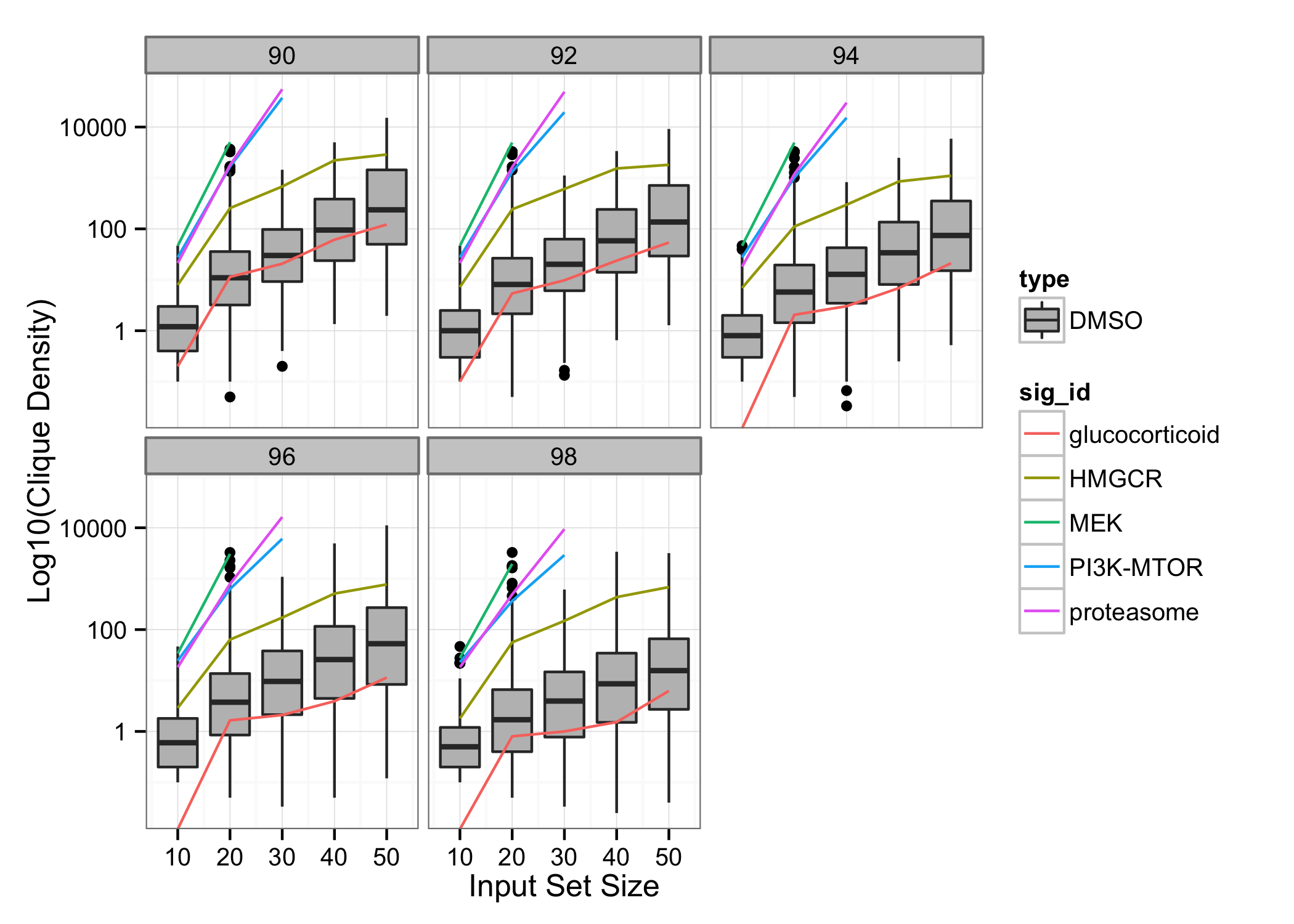
In addition to displaying the graph, QViz also computes the clique density of the graph, and the size and members of the largest clique. It supplies p-values for each of these metrics. Please see the section below for how the p-values are computed. A clique, or fully connected subgraph, is a subset of a graph whose members all share a connection to each other (Luce & Perry, 1949). The clique density is computed is the number of cliques divided by the total number of nodes, and it can be thought of as a proxy for the interconnectedness of a graph. For example, if graphs A and B have the same number of nodes but graph A has twice as many cliques as graph B, then graph A has twice the clique density and is therefore twice as interconnected. Additionally, because the clique density metric is normalized to account for the number of nodes in a graph, it can be used to compare the interconnectedness across graphs of different sizes.



*Figure 2.* QViz Application Interface. 1. The user can select the connectivity space in which to perform his or her analysis and the rankpoint threshold to use for calling connections. Currently, QViz uses only the summly dataset as its connectivity space, but in principal any arbitrary space could be used. 2. The user can input his or her own set of perturbagens or select one of the example sets to graph. 3. QViz draws a graph using the input nodes. The perturbagens are represented as nodes and edges are drawn between those nodes with a connection strength exceeding the user-defined threshold. 4. QViz computes and displays the number of cliques, the clique density, the size of the largest clique, and significance values. 5. The members of the largest clique are listed beside the graph. 6. The user is able to select a pointer cursor to hover over and drag nodes, or a brush to select multiple nodes. 7. The user can enter names of perturbagens to highlight in the graph, if they are present.

Estimating Significance

QViz computes the clique density and the size of the largest clique for each set of perturbagens input by the user, but these metrics are difficult to interpret without an accompanying measure of significance. For example, a clique density of 25 is somewhat meaningless without a sense of how frequently a value of 25 or higher is achieved in general. To assess the frequency of clique density value occurrences, it is necessary to build a reference distribution of clique density values. This reference distribution was constructed by computing the clique densities of the connections resulting from querying the CMap database with 1,000 randomly selected signatures of dimethylsulfoxide (DMSO) treatment. DMSO is a solvent that is frequently used to dissolve and deliver compound treatments to cultured cells. Hence, the signature of treating with DMSO alone serves a relevant negative control for many perturbagen signatures.

Figure 3 shows the distribution of clique density values for DMSO query results across a small range of graph sizes and rank point thresholds. Such a distribution was computed for graph sizes between 2 and 100 and rank point thresholds from 90 to 99. Each QViz analysis results in a clique density D based on the number and identity of the user’s list of input perturbagens and the rank point threshold they have selected. The application computes the p-value, or the probability of observing a clique density at least as extreme as V by computing the proportion of of clique density values more extreme than V from the DMSO distribution at the same perturbagen number and rank point threshold. For example, a density of D and a p-value of 0.05 indicates that in the DMSO distribution, 5% of the observed values were greater than equal to D. In the same fashion, the DMSO distributions are used to compute p-values for the largest clique size.

*Figure 3.* Example Clique Densities. Each panel shows the distributions of clique density for the 1,000 DMSO queries (gray) at increasing input set sizes. The clique densities for the example set queries are shown as lines. The five panels correspond to five different rankpoint cutoffs. The MEK inhibitor, PI3K/MTOR inhibitor, and proteasome inhibitor queries are much more interconnected than the DMSO queries. These three queries are so interconnected that QViz was unable to compute their clique densities at higher input set sizes, which is why their curves are truncated. However, the HMGCR inhibitor query is more connected than most, but not all of the DMSO queries and the glucocorticoid agonist query is less connected than almost all DMSO queries.

Other Application Features

At a high level, QViz allows users to input their CMap query results, visualize them as a graph, and identify subsets that may be highly interconnected. In order to maintain reasonable performance, the application allows the user to input a maximum of 100 perturbagens. Once the graph is computed and drawn, QViz highlights and lists the members of the largest clique in the graph so that the user is able to easily identify them. The user is able to hover over each graph node to see the name of the corresponding perturbagen. He or she can also change the cursor to a selection brush and can drag the mouse to highlight multiple nodes and see the names of their corresponding perturbagens. QViz provides a search box so that the user can input the names(s) of perturbagens to search for and highlight within the graph. The nodes of the graph can be dragged and repositioned so that the user can isolate particular nodes. QViz has a ‘take a tour’ feature that walks the user through each of the application’s components. It also has a list of pre-selected, curated sets of perturbagens, or example sets, that users can select to see an example use of the application. These example sets include the connections resulting from CMap queries with expression signatures of the following compounds:

1. MEK inhibitor
2. PI3K/MTOR inhibitor
3. HMGCR inhibitor
4. Glucocorticoid agonist
5. Proteasome inhibitor

These example sets were selected such that they represent a diversity of biological pathways and processes. Hence, they help give the user a feel for the level interconnectedness achievable for diverse but highly biologically relevant queries.

Software Components

The following sections describe the various software components used to build the user QViz interface and the web server that supports it.

Front End: HTML & D3.js

Hypertext markup language (HTML) is and has been the standard language for displaying information over the Internet within web browsers. HTML5, the most recent revision of the HTML standard, is used as the framework QViz. HTML5 offers many useful features for application development and is supported by most modern web browsers (W3C, 2011).

To support user interaction, the graph visualization is built using D3.js, a JavaScript library for data visualization. Created by Mozilla in 1995, JavaScript is a programming language that is interpreted by most modern web browsers and allows developers to create interactive elements within web pages (Mozilla, 2013). D3, short for Data Driven Documents, is a JavaScript library written by Mike Bostock and designed specifically to enable rich and interactive data visualizations (Bostock, 2013). D3 is particularly well-suited for visualizing CMap connections because of its ability to easily integrate and bind data to on-screen elements. It has been used in many similar projects and is capable of generating the graph of visualizations in QViz. The application also relies on the Barista CSS and and JavaScript libraries, written by CMap team member Corey Flynn (Flynn, 2014).

Back End: MongoDB, R, and Node.js

MongoDB is a database system developed by MongoDB Inc. Unlike traditional Structured Query Language (SQL) database systems that require rigid data storage schema, MongoDB’s schema is very loose and fluid. Rather than storing data in tables that may or may not be linked to each other, MongoDB stores data in “collections,” where each collection is simply a list of “documents.” Documents are simply data objects that can have any number of attributes and each document need not have the same attributes as others (MongoDB, 2013). Perhaps the main benefit of using MongoDB is that it natively stores data in JavaScript Object Notation (JSON) format (ECMA, 1999). JSON is the data object format used by JavaScript, so using MongoDB to store the connectivity data means that when the application queries MongoDB, the database will respond with data in a format the application can easily handle.

MongoDB documents are simply JSON objects. These objects contain key-value pairs and the values can be accessed by providing the appropriate keys, or attributes. A CMap connection can be modeled as a very simple JSON object with the following attributes:

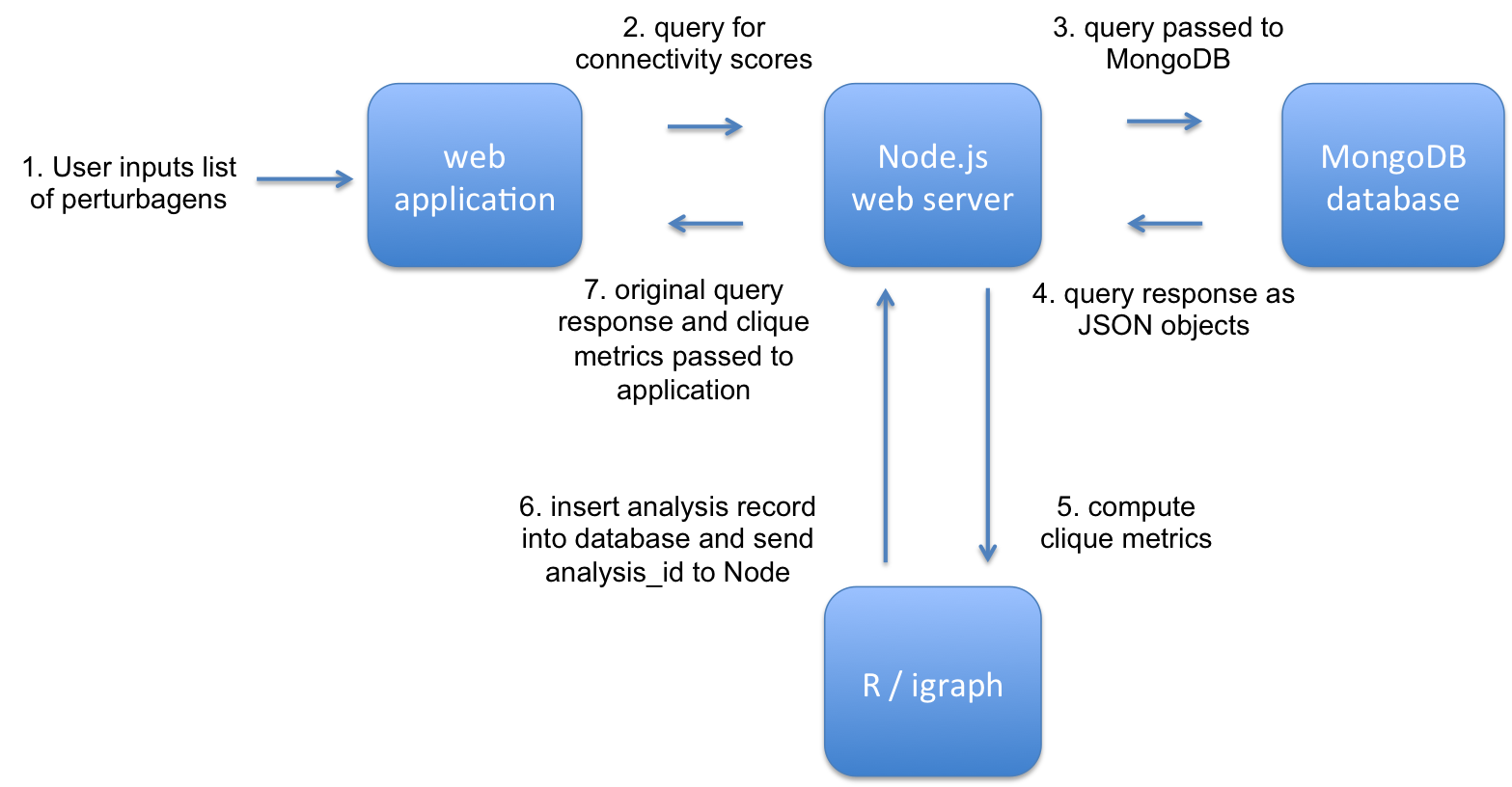
1. perturbagen 1
2. perturbagen 2
3. rankpoint

An example CMap connection stored as a JSON document might look like this:

{  
 "perturbation\_1": "vorinostat",  
 "perturbation\_2": "trichostatin-a",  
 "rankpoint": 98.0  
}

In the example above, vorinostat and trichostatin-a have a connection strength of 98.0. Providing the names of the two perturbagens is enough to uniquely identify this and any CMap summly connection. MongoDB allows for searching over the values of all documents that contain a given key or set of keys. QViz stores each CMap connection as a document in a single MongoDB collection. Based on the user’s input set of query results (perturbagen names), MongoDB is able to retrieve all connections between the query results by looking up all documents where the perturbagen\_1 and perturbagen\_2 fields are members of the input query result set and then return the results to the application as a JSON object.

R is a programming language for statistical analysis. It was developed in 1997 by Ross Ihaka and Robert Gentleman and has since become widely used in many analytical computing applications (R, 2014). It has a large community of developers who contribute packages and utilities to perform specific analyses. In this work, the ‘igraph’ package is used to compute the clique properties of the graphs generated by QViz. This package is appealing for the QViz use case because it contains built-in functions that compute the number of cliques and the size and number of the largest cliques in a graph, among other useful features (Csardi & Nepusz, 2006).

Node.js is a JavaScript-based platform for web server development. It implements an event-driven paradigm, which means that it enables writing programs built for quickly responding to inputs from a user or another application (Node, 2013). In this project, Node.js acts as the web server that handles requests from the web application and query responses from MongoDB. It acts as the middle layer that shuffles data between MongoDB, where it is stored, R, where it is analyzed, and the web application, where it is displayed.

*Figure 4.* Application Data Flow Diagram. The application receives a list of perturbagens from the user. It then sends a query for these perturbagens' connectivity scores to Node.js. Node.js receives the query, passes it to MongoDB, and waits for the response. Once the JSON response is received, Node.js passes the resulting connectivity scores to R to model the perturbagens as a graph and compute the graph's clique metrics. Once this computation is done, R stores the result as document in the database and passes the document's unique identifier to Node.js. Node.js then passes the graph object and analysis identifier to the application for visualization.

Figure 4 illustrates how data flows through the various front and back end layers of the application. Node.js is appealing for this use case because, like MongoDB and D3, it is based in JavaScript. It therefore allows for easily passing query parameters from the application to MongoDB and query results as JSON objects from MongoDB to the application.

Chapter III

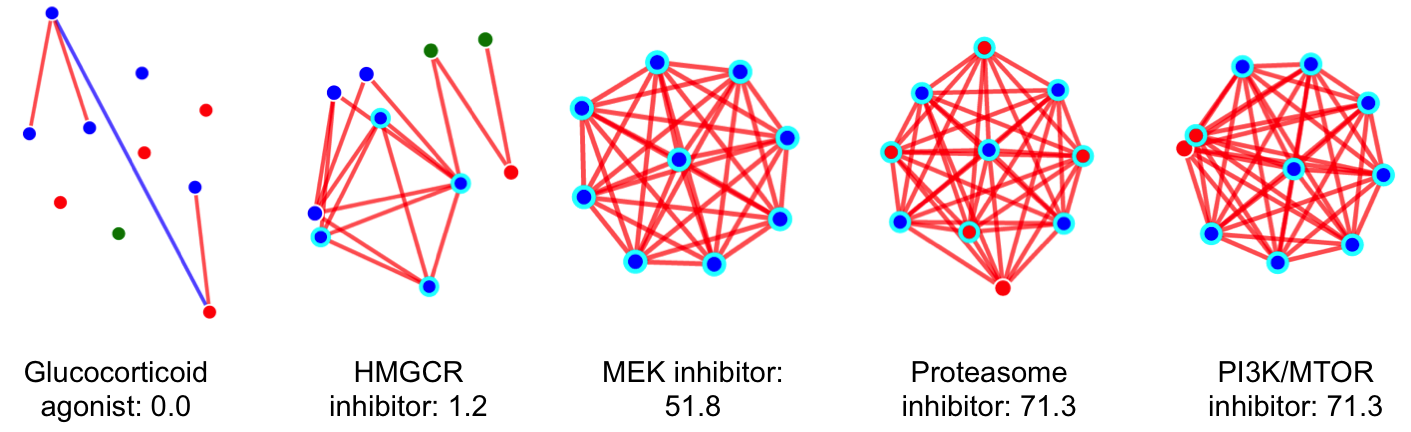
Results

QViz was used to investigate the interconnectivities of the nearly one thousand DMSO queries and the five example set query results. The sections below describe the findings.

Interconnectivity Varies by Query Result

As Figure 3 indicates, the DMSO query results have a range of clique densities. The distributions of the DMSO densities provide references against which we can compare clique densities from other query results. In contrast, the query results from the example sets have higher clique densities than the majority of the DMSO query results. This suggests that queries that contain some amount of biological signal more frequently connect to related groups of perturbagens than do queries that are less biologically meaningful. Additionally, this observation indicates that clique density is a reasonable metric for distinguishing biologically relevant and irrelevant queries.

Two trends are also evident in the DMSO query result data. The first is that right tail of the distribution, that is, the number of queries yielding highly interconnected results, is more prevalent at lower score thresholds. For a given number of input nodes, fewer connections are retained and clique density decreases as the score threshold becomes more stringent. The second trend is that the right tail of the distribution is more prevalent at higher numbers of input nodes. For a given score threshold, more connections are formed as more and more nodes are considered. This behavior is important for the user to consider when selecting input group sizes and score thresholds. The more perturbagens he or she input and the lower he or she sets the score threshold, the more cliques he or she is likely to observe simply due to chance.

 The query results from the example sets are also quite interesting. While these five queries were carefully curated and validated by the CMap team as being biologically meaningful, they also display a range of clique densities. If we consider only the top ten connections for each query result, we see that the densities range from 0 for the glucocorticoid agonist connections all the way to upwards of 70 for the PI3K/MTOR inhibitor and Proteasome inhibitor connections (Figure 5). This suggests that, depending on the biological pathway or mechanism to which the query connects, clique densities may vary widely. It could be that some pathways are more sensitive than others, and hence may be impacted by a larger number of perturbagens. Another possibility is that the variation in clique density is artifactually due to over- or under-representation of some pathways or mechanisms in the summly dataset. For example, if the summly dataset contained no members of pathway X, then a query with the signature of pathway X would likely have a low clique density, and vice versa. Because it is currently challenging to bin all perturbagens into pathways or coherent groups, it is difficult to account for any such biases in QViz, but this may become more tractable as pertubagens are better understood and their annotations improve.

*Figure 5.* Graphs and Clique Densities for Example Sets. Each panel shows the graph and the clique density of the top 10 connections from one of the five example sets at a rankpoint threshold of 90. The glucocorticoid agonist graph is entirely devoid of cliques. The HMGCR inhibitor graph is a bit more interconnected, with 12 cliques across its 10 nodes. The remaining three graphs are extremely dense and interconnected and present a striking visual contrast to the first two.

Chapter IV

Summary and Conclusions

Using QViz, I have been able to characterize an array of perturbagen sets across a small variety of CMap queries. Though these queries were derived from a diverse set of biological processes all have some level of biological relevance, they differ greatly in their levels of interconnectivity. QViz is able to visually highlight these differences and therefore aid in the interpretation of the query results.

QViz was motivated by the need to refine CMap query results. Hence, all of the analyses and examples in this work are based on finding relationships amongst sets of perturbations derived by running CMap queries. However, the QViz algorithm could in principle be applied to any arbitrary set of perturbations. One could use the application to investigate relationships between sets of perturbations derived in any number of ways. For example, one could input a list of genes that may potentially belong to the same pathway or a list of compounds that may share the same mechanism of action and obtain a measure of how interrelated the perturbations are in the space of gene expression.

Similarly, although QViz was implemented using the summly dataset, in principle any dataset could be used. This could allow for extending QViz to a variety of datasets, each potentially tailored to a specific application. For example, one could use a dataset of connectivities generated within a particular cell context of interest to identify relationships that are potentially context-specific.

There were a number of challenges in creating the QViz application. While it represents only a subset of the entire CMap matrix, the summly dataset still contains over 7,000 perturbations, which correspond to over 50 million pairwise comparisons and 50 million documents in the MongoDB database. Although MongoDB is a capable datastore, the large number of documents hamper the speed of database lookups, especially for large numbers of input query results. Additionally, the clique identification is an expensive computation and one that scales exponentially with the number of inputs. For this reason, QViz limits the maximum number of input pertubagens to 100, but performance at this size can still be sluggish, especially for input sets that are highly interconnected. Potential improvements to the application and algorithm include further optimization of database lookups and investigating alternative methods for more efficiently computing interconnectivity.

While the current version of QViz could benefit from implementation improvements, in its current state it is effective at identifying sets of interrelated perturbagens. It is also extensible to other algorithms, datasets, and inputs, and thereby has the potential to become an even more useful tool for generating biological hypotheses from CMap queries.

Appendix A: Source Code

CSS

qviz.css

/\*

Some basic styling for the QViz application.

Ted Natoli

August 2014

\*/

.node {

stroke: #fff;

stroke-width: 1.5px;

}

.selected {

stroke: #00FFFF;

stroke-width: 3px;

}

.link {

stroke: #999;

stroke-opacity: .6;

}

.dev

{

outline: #000000 dotted;

}

.brush .extent {

fill-opacity: .1;

stroke: #fff;

shape-rendering: crispEdges;

}

HTML

index.html

<!-- code for generating the Qviz visualization

Ted Natoli

August 2014 -->

<!DOCTYPE html>

<html lang="en">

<head>

<title>qviz</title>

<link rel="stylesheet" href="css/qviz.css">

<link rel="stylesheet" href="css/font-awesome.css">

<link rel="stylesheet" href="css/barista.main.min.css">

</head>

<body><!-- header view target -->

<div class="cmap-shadow">

<div id="header\_target"></div>

<div style="text-align: center">

<span><a class="cmap-link" id="take\_a\_tour">take a tour</a></span>

</div>

</div>

<div style="min-height:30px"></div>

<div class="row">

<div class="col-xs-offset-1 col-xs-10" id="content">

<div class="row" id="inputs">

<!-- inputs for redrawing -->

<form class="form-horizontal">

<fieldset>

<!-- Form Name -->

<legend>Select Graph Parameters</legend>

<div class="col-xs-4">

<!-- connectivity space -->

<div class="form-group">

<label class="col-xs-4 control-label" for="conn\_space">Connectivity Space:</label>

<div class="col-xs-6">

<select id="conn\_space" name="conn\_space" class="form-control" data-step="1" data-intro="Select the connectivity space in which to perform your analysis." data-position="right">

<option value="summly">Across Cell Lines</option>

</select>

</div>

</div>

<!-- threshold -->

<div class="form-group">

<label class="col-xs-4 control-label" for="threshold">Score threshold:</label>

<div class="col-xs-6">

<select id="threshold" name="threshold" class="form-control" data-step="2" data-intro="Set the threshold above which to call a connection." data-position="right">

<option value="90">90</option>

<option value="91">91</option>

<option value="92">92</option>

<option value="93">93</option>

<option value="94">94</option>

<option value="95">95</option>

<option value="96">96</option>

<option value="97">97</option>

<option value="98">98</option>

<option value="99">99</option>

</select>

</div>

</div>

</div>

<div class="col-xs-4">

<!-- input nodes -->

<div class="form-group">

<label class="col-xs-4 control-label" for="nodes">Nodes to Graph:</label>

<div class="col-xs-8">

<textarea class="form-control" id="nodes" name="nodes" data-step="3" data-intro="Paste in up to 100 names of perturbagens to graph." data-position="right"></textarea>

</div>

</div>

<!-- Button -->

<div class="form-group">

<label class="col-xs-4 control-label" for="singlebutton"></label>

<div class="col-xs-8">

<button name="singlebutton" class="btn btn-default" id="clear\_nodes" type="button">Clear</button>

</div>

</div>

<div class="row">

<label class="col-xs-4 control-label" for="nodes">OR</label>

</div>

<!-- poscon set -->

<div class="form-group">

<label class="col-xs-4 control-label" for="poscon">Example Set:</label>

<div class="col-xs-8">

<select id="poscon" name="poscon" class="form-control" data-step="4" data-intro="OR select from example sets of query results." data-position="right">

<option value="mek">MEK inhibitor</option>

<option value="pi3k\_mtor">PI3K/MTOR inhibitior</option>

<option value="hmgcr">HMGCR inhibitior</option>

<option value="glucocorticoid">Glucocorticoid agonist</option>

<option value="proteasome">Proteasome inhibitor</option>

</select>

</div>

</div>

</div>

<div class="col-xs-4">

<!-- radio buttons -->

<div class="form-group">

<label class="col-xs-4 control-label" for="radios">Cursor Type:</label>

<div class="col-xs-4">

<div class="radio">

<label for="pointer">

<input type="radio" name="cursor" id="pointer" value="pointer" checked="checked" data-step="9" data-intro="Set the cursor to a pointer to drag and reposition nodes..." data-position="left">

pointer

</label>

</div>

<div class="radio">

<label for="brush">

<input type="radio" name="cursor" id="brush" value="brush" data-step="10" data-intro="OR to a brush to highlight multiple nodes." data-position="left">

brush

</label>

</div>

</div>

</div>

<!-- Button -->

<div class="form-group">

<button id="redraw" name="redraw" class="btn btn-primary col-xs-4 col-xs-offset-4" data-step="11" data-intro="Click this button to re-draw the graph if after changing any parameters." data-position="left">Redraw</button>

</div>

</div>

</fieldset>

</form>

</div>

<hr>

<div style="min-height:30px"></div>

<!-- the vis components -->

<div class="row">

<!-- loading group -->

<div id="loading\_group" class="row">

<div class="cmap-spacer-large"></div>

<div class="col-xs-offset-1 col-xs-10">

<div class="col-xs-offset-5 col-xs-2 cmap-loading">

</div>

</div>

</div>

<div class="row" id="graph\_container" style="display: block; visibility: hidden">

<!-- clique stats -->

<div class="col-xs-3">

<div class="row">

<div class="col-xs-12" id="clique\_stats" data-step="6" data-intro="Stats on the graph are displayed here." data-position="right">

<div class="panel panel-default">

<!-- Default panel contents -->

<div class="panel-heading">Graph Stats</div>

<!-- List group -->

<ul class="list-group">

<li class="list-group-item">Number of cliques: <span style="float: right"><strong id="num\_cliques"></strong></span></li>

<li class="list-group-item">Clique density: <span style="float: right"><strong id="clique\_density"></strong></span></li>

<li class="list-group-item">Clique density p-value: <span style="float: right"><strong id="clique\_density\_p"></strong></span></li>

<li class="list-group-item">Largest clique size: <span style="float: right"><strong class="largest\_clique\_size"></strong></span></li>

<li class="list-group-item">Largest clique size p-value: <span style="float: right"><strong id="largest\_clique\_size\_p"></strong></span></li>

</ul>

</div>

</div>

<!-- inputs for highlighting -->

<div class="col-xs-12" id="textarea\_input">

<form class="form-horizontal">

<fieldset>

<!-- Textarea -->

<div class="form-group">

<label class="col-xs-4 control-label" for="highlight">Highlight nodes:</label>

<div class="col-xs-8">

<textarea class="form-control" id="highlight" name="highlight" data-step="8" data-intro="Paste or type in names of pertubagens to highlight in the graph." data-position="right"></textarea>

</div>

</div>

</fieldset>

<!-- Button -->

<div class="form-group">

<label class="col-xs-4 control-label" for="singlebutton"></label>

<div class="col-xs-8">

<button name="singlebutton" class="btn btn-primary" id="submit" type="button">Submit</button>

<button name="singlebutton" class="btn btn-default" id="clear" type="button">Clear</button>

</div>

</div>

</form>

</div>

</div>

</div>

<!-- the graph -->

<div class="col-xs-6" id="graph" data-step="5" data-intro="The graph of interconnectivity is displayed here. Click nodes to reposition them. Hover over nodes to see their identities." data-position="right"></div>

<!-- largest clique members -->

<div class="col-xs-3">

<div class="panel panel-default" data-step="7" data-intro="The members of the largest clique are listed here." data-position="left">

<!-- Default panel contents -->

<div class="panel-heading">Members of Largest Clique <span style="float: right"><strong class="largest\_clique\_size"></strong></span></div>

<div class="panel-body" id="largest\_clique">

</div>

</div>

<div class="row">

<div class="col-xs-12">

<!-- legend {"trt\_cp": "blue", "trt\_oe": "green", "trt\_sh.cgs": "red"}; -->

<svg height="100" width="200">

<circle cx="10" cy="10" r="10" stroke="none" stroke-width="3" fill="blue" /><text x="25" y="15" fill="black">compound</text>

<circle cx="10" cy="35" r="10" stroke="none" stroke-width="3" fill="green" /><text x="25" y="40" fill="black">over-expression</text>

<circle cx="10" cy="60" r="10" stroke="none" stroke-width="3" fill="red" /><text x="25" y="65" fill="black">knockdown</text>

</svg>

</div>

</div>

</div>

</div>

</div>

</div>

<!-- hidden textarea to store pert\_inames -->

<textarea style="display: none" id="pert\_inames"></textarea>

</div>

<div class="row"></div>

<div id="footer\_target"></div>

</body>

<script src="libraries/barista.main.min.js"></script>

<script type="text/javascript">

var example\_queries = {

"mek": [

"selumetinib",

"AS-605240",

"PD-0325901",

"U0126",

"U-0126",

"PD-0325901",

"AS-703026",

"AZ-628",

"MEK1-2-inhibitor",

"PD-184352"

],

"pi3k\_mtor": [

"KU-0063794",

"AZD-8055",

"NVP-BEZ235",

"PP-110",

"PI-828",

"TGX-115",

"PI-103",

"MTOR",

"AFF4",

"BMS-536924"

],

"hmgcr": [

"lovastatin",

"selumetinib",

"PIK3R1",

"MSH2",

"neratinib",

"BMS-536924",

"BMS-754807",

"TXNIP",

"simvastatin",

"5-nonyloxytryptamine"

],

"glucocorticoid": [

"UBL5",

"TPCA-1",

"fluticasone",

"ETV1",

"CCDC92",

"RHO-kinase-inhibitor-III[rockout]",

"COPS5",

"dexamethasone",

"GAMT",

"forskolin"

],

"proteasome": [

"PSMD1",

"bortezomib",

"PSMA1",

"MLN-2238",

"VCP",

"HSPA5",

"radicicol",

"PSMB2",

"MG-132",

"tosedostat"

]

};

// define analysis\_id

var analysis\_id = "query\_" + new Date().getTime();

console.log(analysis\_id);

// get the connection space

var conn\_space = $("#conn\_space").val();

// get list of pert\_inames from database

$.ajax({

url: "/get\_inames/" + conn\_space,

type: "GET",

dataType: "json",

success: function(response) {

$("#pert\_inames").val(response.members);

},

error: function(err) {

console.error(err);

}

})

// set up some functions

function draw\_graph(edge\_color, node\_color, nodes, edges) {

// clear the svg and brush, if any

$("#graph svg").remove();

// draw the force-directed graph

var width = Math.max($("#graph").width(), 380); // 600

var height = Math.max($("#graph").height(), 380);

var force = d3.layout.force()

.charge(-120)

.size([width, height]);

var svg = d3.select("#graph").append("svg")

.attr("width", width)

.attr("height", height);

force

.nodes(nodes)

.links(edges)

.linkDistance(function(d) { return (100 - d.score + 100); }) // assume scores are rankpoints

.start();

var link = svg.selectAll(".link")

.data(edges)

.enter().append("line")

.attr("class", "link")

.style("stroke", function(d) { return edge\_color[d.direction]; })

.style("stroke-width", 3);

var node = svg.selectAll(".node")

.data(nodes)

.enter().append("circle")

.attr("class", "node")

.attr("id", function(d) { return d.id; })

.style("fill", function(d) { return node\_color[d.pert\_type]; })

.attr("r", 6)

.call(force.drag);

node.append("title")

.text(function(d) { return d.id; });

// set up graph behavior when reaching equilibrium

force.on("tick", function() {

link.attr("x1", function(d) { return d.source.x; })

.attr("y1", function(d) { return d.source.y; })

.attr("x2", function(d) { return d.target.x; })

.attr("y2", function(d) { return d.target.y; });

node.attr("cx", function(d) { return d.x; })

.attr("cy", function(d) { return d.y; });

});

// enable brushing

var brush = svg.append("g")

.attr("class", "brush")

.call(d3.svg.brush()

.x(d3.scale.identity().domain([0, width]))

.y(d3.scale.identity().domain([0, height]))

.on("brush", function() {

var extent = d3.event.target.extent();

node.classed("selected", function(d) {

if(extent[0][0] <= d.x && d.x < extent[1][0] && extent[0][1] <= d.y && d.y < extent[1][1]) {

return true;

}

else {

return false;

}

});

var selected = [];

$(".selected").each(function(i, el) {

selected.push($(el).attr("id"));

$("#highlight").val(selected.join("\n"));

})

if(selected.length===0) {

$("#highlight").val("");

}

}));

// hide the brush initially. user can show it by clicking radio button

$(".brush").hide();

}

function show\_graph\_and\_stats(cliques) {

$("#loading\_group").hide(500);

$("#graph\_container").css("visibility", "visible");

$("#num\_cliques").html(cliques.num\_cliques);

$("#clique\_density").html(Math.round((cliques.num\_cliques / node\_collection.length) \* 100) / 100);

$("#clique\_density\_p").html(cliques.clique\_density\_p);

$(".largest\_clique\_size").html(cliques.largest\_clique\_size);

$("#largest\_clique\_size\_p").html(cliques.largest\_clique\_size\_p);

$("#largest\_clique").html(''); // clear previous values

$("#largest\_clique").html(cliques.largest\_cliques["1"].members.join("<br>")); // choose the first largest clique

// highlight the largest clique members

cliques.largest\_cliques["1"].members.forEach(function(member) {

d3.select("#" + member)

.attr("class", "selected");

})

}

// basic header and footer views

var CMapHeaderView = new Barista.Views.CMapHeaderView({el: $("#header\_target"),title: "QViz",subtitle: "visualizing cmap query results"});

var CMapFooterView = new Barista.Views.CMapFooterView({el: $("#footer\_target")});

// set up collection(s)

var node\_collection = new Backbone.Collection([]);

var edge\_collection = new Backbone.Collection([]);

// set up some variables for force-directed graph

var node\_color = {"trt\_cp": "blue", "trt\_oe": "green", "trt\_sh.cgs": "red"};

var edge\_color = {"pos": "red", "neg": "blue"};

// fetch data and draw the graph upon page load

var collection = $("#conn\_space").val();

var threshold = parseFloat($("#threshold").val());

var example\_set = $("#poscon").val();

var query = $.ajax({

url: "/get\_connections/" + collection + "?thresh=" + threshold + "&analysis\_id=" + analysis\_id,

type: "GET",

dataType: "json",

data: {nodes: example\_queries[example\_set]}

}).done(function(response) {

node\_collection.reset(response.nodes);

edge\_collection.reset(response.edges);

draw\_graph(edge\_color, node\_color, node\_collection.toJSON(), edge\_collection.toJSON());

console.log(analysis\_id);

get\_cliques(analysis\_id, threshold, example\_queries[example\_set].length);

});

get\_cliques = function(analysis\_id, threshold, sample\_size) {

// fetch the corresponding cliques

$.ajax({

url: "/get\_cliques/" + analysis\_id,

type: "GET",

dataType: "json",

success: function(cliques) {

console.log(cliques);

// get the p-values

$.ajax({

url: "/get\_p\_values",

type: "GET",

dataType: "json",

data: {

"sample\_size": sample\_size,

"threshold": threshold,

"num\_cliques": cliques.num\_cliques,

"largest\_clique\_size": cliques.largest\_clique\_size

},

success: function(p\_values) {

console.log(p\_values);

cliques["clique\_density\_p"] = p\_values.clique\_density\_p;

cliques["largest\_clique\_size\_p"] = p\_values.largest\_clique\_size\_p;

show\_graph\_and\_stats(cliques);

},

error: function(err) {

console.error(err);

}

})

},

error: function(err) {

console.error(err);

}

})

}

// parse input from textarea

$("#submit").click(function(event) {

var lines = $("#highlight").val().split("\n");

lines.forEach(function(entry) {

d3.select("#" + entry)

.attr("class", "selected");

})

})

clear\_selections = function() {

$("#highlight").val("");

d3.selectAll(".selected")

.attr("class", "node");

}

// clear any selections

$("#clear").click(function(event) {

clear\_selections();

})

clear\_nodes = function() {

$("#nodes").val("");

}

// clear any selections

$("#clear\_nodes").click(function(event) {

clear\_nodes();

})

$("#poscon").change(function(event) {

clear\_nodes();

})

// listen to radio buttons

$("[name=cursor]").click(function() {

var hide\_brush = $("input:radio:checked").val() === "pointer";

if (hide\_brush) {

$(".brush").hide();

}

else {

$(".brush").show();

}

})

// process input from poscon set and query space dropdown

$("#redraw").click(function(event) {

event.preventDefault();

clear\_selections();

// hide the graph and show the loading group

$("#graph\_container").css("visibility", "hidden");

$("#loading\_group").show(500);

// update the analysis\_id

analysis\_id = "query\_" + new Date().getTime();

// fade out the current nodes and edges

d3.selectAll(".node").transition().attr("r", 0).remove();

d3.selectAll(".link").transition().style("stroke", "white").remove();

var example\_set = $("#poscon").val();

var collection = $("#conn\_space").val();

var threshold = parseFloat($("#threshold").val());

var nodes = [];

$("#nodes").val().split("\n").forEach(function(node) {

if (node !== "") nodes.push(node);

})

// use input nodes if any, else example set

nodes = nodes.length > 0 ? nodes : example\_queries[example\_set];

// use only 100 nodes even if they input more

nodes = nodes.length <= 100 ? nodes : nodes.slice(0, 101);

// check if any nodes are not in the pert\_iname set

var known = $("#pert\_inames").val().split(",");

var unrecognized = \_.difference(nodes, known);

if (unrecognized.length > 0) {

alert("These inputs were not recognized:\n" + unrecognized.join("\n"));

}

var query = $.ajax({

url: "/get\_connections/" + collection + "?thresh=" + threshold + "&analysis\_id=" + analysis\_id,

type: "GET",

dataType: "json",

data: {"nodes": nodes}

}).done(function(response) {

node\_collection.reset(response.nodes);

edge\_collection.reset(response.edges);

draw\_graph(edge\_color, node\_color, node\_collection.toJSON(), edge\_collection.toJSON());

console.log(analysis\_id);

// fetch the corresponding cliques

get\_cliques(analysis\_id, threshold, nodes.length);

});

})

// take a tour

$("#take\_a\_tour").click(function(){

introJs().start();

});

</script>

</html>

JavaScript

server.js

// code for defining the web server that will interact with

// the MongoDB database and, call R to compute clique density,

// and correspond with the QViz web-based visualization.

// Ted Natoli

// August 2014

var express = require('express');

var fs = require('fs');

var spawn = require('child\_process').spawn;

var app = express();

var MongoClient = require('mongodb').MongoClient;

var format = require('util').format;

app.use(express.logger('dev'));

app.use(express.bodyParser({ keepExtensions: true }));

app.use(express.static(\_\_dirname + "/app"));

var dbloc = "mongodb://127.0.0.1:27017/thesis";

var dump\_dir = "/Users/tnatoli/github/thesis/thesis/code/app/data/tmp";

var clique\_script\_path = "/Users/tnatoli/github/thesis/thesis/code/utils/find\_cliques.R"

// curated sets that should be interconnected

var poscons = {

"HDAC\_simple": [

"dacinostat",

"trichostatin-a",

"vorinostat",

"panobinostat"

],

"HDAC\_inhibitor": [

"dacinostat",

"apicidin",

"KM-00927",

"ISOX",

"HC-toxin",

"merck-ketone",

"trichostatin-a",

"vorinostat",

"belinostat",

"panobinostat",

"THM-I-94",

"scriptaid"

],

"HDAC\_plus\_random": [

"dacinostat",

"apicidin",

"KM-00927",

"ISOX",

"HC-toxin",

"merck-ketone",

"trichostatin-a",

"vorinostat",

"belinostat",

"panobinostat",

"THM-I-94",

"scriptaid",

"BRD-K14395166",

"PASK"

],

"HDAC\_plus\_PI3K\_inhibitor": [

"dacinostat",

"apicidin",

"KM-00927",

"ISOX",

"HC-toxin",

"merck-ketone",

"trichostatin-a",

"vorinostat",

"belinostat",

"panobinostat",

"THM-I-94",

"scriptaid",

"XL-147",

"AS-605240",

"GSK-1059615",

"idelalisib",

"AS-604850",

"honokiol",

"wortmannin",

"TGX-221",

"NU-7441",

"NVP-BEZ235",

"LY-294002",

"buparlisib",

"AZD-6482",

"PI-103"

],

"PI3K\_inhibitor": [

"XL-147",

"AS-605240",

"GSK-1059615",

"idelalisib",

"AS-604850",

"honokiol",

"wortmannin",

"TGX-221",

"NU-7441",

"NVP-BEZ235",

"LY-294002",

"buparlisib",

"AZD-6482",

"PI-103"

],

"small\_cluster": [

"BRD-K97534490",

"erbstatin-analog",

"BRD-K58479490",

"tozasertib",

"BRD-K62056344",

"fenbendazole",

"BRD-A16820783",

"VU-0365117-1",

"ST-001903",

"BRD-K34351329",

"CYT-997",

"fatostatin",

"SB-225002",

"scoulerine"

],

"HDAC\_with\_small\_cluster": [

"dacinostat",

"apicidin",

"KM-00927",

"ISOX",

"HC-toxin",

"merck-ketone",

"trichostatin-a",

"vorinostat",

"belinostat",

"panobinostat",

"THM-I-94",

"scriptaid",

"BRD-K97534490",

"erbstatin-analog",

"BRD-K58479490",

"tozasertib",

"BRD-K62056344",

"fenbendazole",

"BRD-A16820783",

"VU-0365117-1",

"ST-001903",

"BRD-K34351329",

"CYT-997",

"fatostatin",

"SB-225002",

"scoulerine"

],

};

function make\_nodes\_and\_edges(docs, connection\_thresh) {

// take a list of docs and create an array of nodes and edges to be

// sent back to client

var seen\_perts = [];

var seen\_combos = [];

var nodes = [];

var edges = [];

var indices = {};

for (var i = 0; i < docs.length; i++) {

var doc = docs[i];

var score = doc.score;

var pert\_iname\_x = doc.pert\_iname\_x;

var pert\_type\_x = doc.pert\_type\_x;

var pert\_iname\_y = doc.pert\_iname\_y;

var pert\_type\_y = doc.pert\_type\_y;

if (pert\_iname\_x !== pert\_iname\_y) {

var combo = [pert\_iname\_x, pert\_iname\_y].sort().join(":");

if (seen\_combos.indexOf(combo) == -1) {

// haven't seen this combo yet, continue on and form a node/edge if appropriate

seen\_combos.push(combo);

if (seen\_perts.indexOf(pert\_iname\_x) == -1) {

// pert\_iname\_x hasn't been seen yet

seen\_perts.push(pert\_iname\_x);

nodes.push( {"id": pert\_iname\_x, "pert\_type": pert\_type\_x} );

indices[pert\_iname\_x] = nodes.length - 1;

}

if (seen\_perts.indexOf(pert\_iname\_y) == -1) {

// pert\_iname\_y hasn't been seen yet

seen\_perts.push(pert\_iname\_y);

nodes.push( {"id": pert\_iname\_y, "pert\_type": pert\_type\_y} );

indices[pert\_iname\_y] = nodes.length - 1;

}

if (Math.abs(score) >= connection\_thresh) {

// score is high enough, add an edge

var edge = {

"source": indices[pert\_iname\_x],

"target": indices[pert\_iname\_y],

"score": score,

"direction": score > 0 ? "pos" : "neg"

}

edges.push(edge);

}

}

}

}

return( {"nodes": nodes, "edges": edges} );

}

// dump a set of edges to a file

function dump\_edges(nodes, edges, file\_name, outpath) {

var data = "source\ttarget\n";

edges.forEach(function(edge) {

data += nodes[edge.source].id + "\t" + nodes[edge.target].id + "\n";

})

fs.writeFile(outpath + "/" + file\_name, data, function(err) {

if(err) throw(err);

console.log("edges saved\n");

});

}

// set up some routes

app.get('/get\_connections/:collection', function(req, res) {

// sends back an object with keys for nodes and edges

// allows specifying a poscon set to return

console.log(req.query);

console.log(req.body);

var collection\_name = req.params.collection;

var limit = parseInt(req.query.limit);

var thresh = req.query.thresh ? req.query.thresh : 90;

var skip = req.query.skip ? parseInt(req.query.skip) : 0;

var nodes = req.query.nodes ? req.query.nodes : false;

if (!nodes) {

res.send(500, "no nodes specified");

}

// make a time-stamp directory

var ts\_dir = req.query.analysis\_id ? req.query.analysis\_id : "query\_" + new Date().getTime();

var outdir = dump\_dir + "/" + ts\_dir;

var edge\_file = outdir + "/" + "edges.txt"

fs.mkdir(outdir, function(err) {

if(err) throw(err);

console.log("created " + outdir);

})

MongoClient.connect(dbloc, function(err, db) {

if(err) console.error(err);

var collection = db.collection(collection\_name);

// request has specified specific nodes to search for, use them

collection.find({

"$and": [

{"pert\_iname\_x": {"$in": nodes}},

{"pert\_iname\_y": {"$in": nodes}}

]

}).toArray(function(err, results) {

if (err) {

console.error(err);

} else {

console.log(results.length + " results");

nodes\_and\_edges = make\_nodes\_and\_edges(results, thresh);

dump\_edges(nodes\_and\_edges.nodes, nodes\_and\_edges.edges, "edges.txt", outdir);

var clique = spawn("/usr/bin/Rscript", [clique\_script\_path, edge\_file, ts\_dir]);

clique.stdout.on("data", function(data) {

console.log(String(data));

})

clique.stderr.on("data", function(data) {

console.error(String(data));

})

clique.on("exit", function(code) {

console.log(code);

res.send(nodes\_and\_edges);

})

}

})

})

})

app.get('/get\_cliques/:analysis\_id', function(req, res) {

// sends back all cliques given the current analysis id

var analysis\_id = req.params.analysis\_id;

MongoClient.connect(dbloc, function(err, db) {

if(err) throw(err);

var collection = db.collection('cliques');

collection.find({"analysis\_id": analysis\_id}).toArray(function(err, results) {

if (err) {

console.error(err);

}

else {

res.send(results[0]);

}

})

})

})

app.get('/get\_inames/:space', function(req, res) {

var space = req.params.space;

MongoClient.connect(dbloc, function(err, db) {

if(err) console.error(err);

var collection = db.collection('members');

collection.find({"space": space}).toArray(function(err, results) {

if (err) {

console.error(err);

}

else {

res.send(results[0]);

}

})

})

})

app.get('/get\_p\_values', function(req, res) {

// based on a clique density, largest clique size, score threshold, and number of nodes,

// compute significance from null collection

var threshold = parseInt(req.query.threshold);

var sample\_size = parseInt(req.query.sample\_size);

var num\_cliques = parseInt(req.query.num\_cliques);

var largest\_clique\_size = parseInt(req.query.largest\_clique\_size);

console.log(threshold, sample\_size, num\_cliques, largest\_clique\_size);

MongoClient.connect(dbloc, function(err, db) {

if (err) console.error(err);

var collection = db.collection("null\_cliques");

collection.find({"sample\_size": sample\_size, "score": threshold}).toArray(function(err, results) {

if (err) console.error(err);

else {

console.log(results);

num\_instances = results.length;

var ge\_clique\_density = results.filter(function(x) {

return x.num\_cliques >= num\_cliques;

});

var ge\_largest\_clique\_size = results.filter(function(x) {

return x.largest\_clique\_size >= largest\_clique\_size;

});

var clique\_density\_p = Math.round(100 \* (ge\_clique\_density.length / num\_instances)) / 100;

var largest\_clique\_size\_p = Math.round(100 \* (ge\_largest\_clique\_size.length / num\_instances)) / 100;

res.send({"clique\_density\_p": clique\_density\_p, "largest\_clique\_size\_p": largest\_clique\_size\_p});

}

})

})

})

app.listen(8080);

R

find\_cliques.R

## A script to find cliques in a graph

## and insert them into a mongodb collection

## as part of the QViz application backend.

## Ted Natoli

## August 2014

# load required packages

require("igraph", lib.loc="/Users/tnatoli/github/thesis/thesis/code/R")

require("rmongodb", lib.loc="/Users/tnatoli/github/thesis/thesis/code/R")

# set some global variables

host <- "localhost"

db <- "thesis"

collection <- "cliques"

namespace <- paste(db, collection, sep=".")

# get arguments

args <- commandArgs(trailingOnly=T)

infile <- args[1]

analysis\_id <- args[2]

min\_size <- args[3] # optional

# make connection to mongo

mongo <- mongo.create(host=host, db=db)

# set min\_size to 3 if not supplied

if (is.na(min\_size)) {

min\_size <- 3

}

# read in the data

d <- read.delim(infile)

d <- d[, c("source", "target")]

# make the graph object

g <- graph.data.frame(d, directed=F)

# find the cliques

graph\_cliques <- cliques(g, min=min\_size)

largest\_cliques <- largest.cliques(g)

if(nrow(d) == 0) {

graph\_cliques <- list()

largest\_cliques <- list()

largest\_clique\_size <- 0

} else {

if(length(largest\_cliques) != 0) {

if (length(largest\_cliques[[1]]) > 2) {

# need at least 3 members in the largest clique

largest\_clique\_size <- length(largest\_cliques[[1]])

}

else {

largest\_clique\_size <- 0

largest\_cliques <- list()

}

} else {

largest\_clique\_size <- 0

}

}

# format the cliques for insertion into mongo

mongo\_cliques <- list()

mongo\_largest\_cliques <- list()

for (gc in graph\_cliques) {

m <- g[gc]

members <- rownames(m)

mongo\_cliques[[length(mongo\_cliques) + 1]] <- list(size = length(members), members = members)

}

for (lc in largest\_cliques) {

m <- g[lc]

members <- rownames(m)

mongo\_largest\_cliques[[length(mongo\_largest\_cliques) + 1]] <- list(size = length(members), members = members)

}

# make a bson object

b <- mongo.bson.from.list(

list(

analysis\_id=analysis\_id,

num\_cliques = length(graph\_cliques),

cliques = mongo\_cliques,

largest\_clique\_size = unlist(largest\_clique\_size),

largest\_cliques = mongo\_largest\_cliques

)

)

print(str(b))

# do the insertion

mongo.insert(mongo, namespace, b)

References

Bostock, M. (2013, November). D3: Data driven documents [computer

software]. Available from http://d3js.org

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex

network research [computer software]. Available from http://igraph.org

ECMA. (1999). JSON [computer software]. Available from

http://www.json.org

Flynn, C. (2014). Barista [computer software]. Available from

https://github.com/cmap/barista

Friedman, N. (2004). Inferring cellular networks using probabilistic graphical

models. *Science, 303 (5659*), 799–805.

[http://www.sciencemag.org/cont](http://www.sciencemag.org/content/303/5659/799)ent/303/5659/799

Hollander, M., & Wolfe, D. A. (1975). Nonparametric

statistical methods. *Biometrische Zeitschrift ,*

*17 (8),* 526–526.

http://dx.doi.org/10.1002/bimj.19750170808

Lage, K., Karlberg, E. O., Storling, Z. M., Olason, P. I.,

Pedersen, A. G., Rigina, O., Brunak, S. (2007). A human

phenome-interactome network of protein complexes

implicated in genetic disorders. *Nature Biotechnology , 25*

*(3),* 309–316.

http://www.nature.com.ezp-

prod1.hul.harvard.edu/nbt/journal/v25/n3/full/nbt1295.html

Lamb, J., Crawford, E. D., Peck, D., Modell, J. W., Blat, I. C.,

Wrobel, M. J., Golub, T. R. (2006). The connectivity map:

Using gene-expression signatures to connect small

molecules, genes, and disease. *Science , 313 (5795),* 1929–

1935. http://www.sciencemag.org/content/313/5795/1929

Luce, R. D., & Perry, A. D. (1949). A method of matrix

analysis of group structure. *Psychometrika, 14 (2),* 95-116.

[http://link.springer.com/article/](http://link.springer.com/article/10.1007/)10.1007/BF02289146

MongoDB. (2013). MongoDB [computer software]. Available from

http://www.mongodb.org

Mozilla. (2013). JavaScript [computer software]. Available from

https://developer.mozilla.org/en-US/docs/Web/JavaScript

Node. (2013). Node.js [computer software]. Available from http://www.nodejs.org

Ihaka, R. & Gentleman, R. (2014). R [computer software]. Available from

http:/www.r-project.org

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L.,

Gillette, M. A., Mesirov, J. P. (2005). Gene set enrichment analysis: A

knowledge-based approach for interpreting genome-wide expression

rofiles. *Proceedings of the National Academy of Sciences of the United*

*States of America , 102 (43),* 15545–15550.

[http://www.pnas.org/content/1](http://www.pnas.org/content/102/43/15545)02/43/15545

Tissing, W. J. E., Meijerink, J. P. P., den Boer, M. L., & Pieters, R. (2003).

Molecular determinants of glucocorticoid sensitivity and resistance in

acute lymphoblastic leukemia. *Leukemia , 17 (1),* 17–25.

van ’t Veer, L. J., Dai, H., van de Vijver, M. J., He, Y. D., Hart, A. A. M., Mao,

M., Friend, S. H. (2002). Gene expression profiling predicts clinical

outcome of breast cancer. *Nature , 415 (6871),* 530–536.

[http://www.nature.com/nature/jo](http://www.nature.com/nature/journal/v415/n6871/abs/415530a.html)urnal/v415/n6871/abs/415530a.html

World Wide Web Consortium (W3C). (2011). HTML5 differences from HTML4.

Retrieved from [http://www.w3.org/TR/2011/WD-htm](http://www.w3.org/TR/2011/WD-html5-diff-20110405/)l5-diff-20110405/