# Bipartite Graph Epi-Stroma

## First Step

### Goals

We want to be able to:

1. Represent a decent sized graph and see the interactions from epi to stroma directly.
2. Select a node and view only its first neighbours
3. Select a node and view its second neighbours as well. This will be somewhat tricky when it comes to drawing the edges from the first neighbours to the second neighbours
4. Show the degree of a node based on its size
5. Once we get home we should refactor the code as to have separate controllers for the graph and the sidenav. we’ll need to use a sharedservice in order to keep track of the necessary variables
6. I’m thinking that we should cache some of the output. We should also perhaps create the initial config on the server side and then send it to the client. Since we will be changing the layout of the graph depending on which gene a user selects, we’ll have a back button that then goes back to the server and grabs the initial config to restore the graph state.
7. Now that we have the initial config cached on the server side,
8. The performance is being impacted by the event handlers that we attached to the graph. Therefore, we will stop these events for the time being. Instead, we are going to have 2 dropdowns. The first will allow us to select either an epi or stroma gene and give us the corresponding interactions with the stroma and epi respectively. Then the second dropdown will have only epi or only stroma depending on what our choice for the first dropdown was. All of the processing for this will be made on the server and will result in new configs being created every time.
9. Had to reorganize the back end logic to include caching of configs. Furthermore, I moved the firstNeighbour logic to the back end from the front end since having select handlers on the front end is what was responsible for the crashing of the application.
10. Improved performance by enabling certain settings in the config that disable the visibility of edges and labels when moving.
11. So far we have been sticking to the current model of returning configurations from the server. Let’s continue to stick with that model for the second dropdown and see how much bandwidth it uses. In order to do this, we need to send the current config from the user to the server so that we can add some further nodes and edges to it.

Organization of Code:

There should be a service that houses the config for the graph. The reason for this is that the config needs to be maniupulated by both the sidenav as well as the graph itself. Our graph controller should really only have things in it that are to do with user interaction with the graph directly such as selecting nodes.

# Back End Logic

## Technical Specifications

* The server being used is a node.js server and it is written in JavaScript. Because of this, the R scripts that are being run by the server need to send back JSON in order for the server to be able to understand the results

## Communication

* The server always sends a cytoscape configuration to the client in order to minimize the computation done on the client side

## Initialization

* The correlation matrices, after the p value test (for values of 0.001, 0.01, 0.05, 0.1), are computed and stored in Rdata files. The degrees of the genes are also computed and stored in RData files. Note that the degrees are also based on the correlation matrices obtained AFTER the p value test
* Then, the initial configurations of the big networks are created and cached on the server (in memory) for later access. This way, requesting the overall graph for a given p value (an operation that takes much longer than work with the subgraphs) can be done quickly.

## Requests

### Overall Graph

* When a user wants the entire network, they are simply sent the cached configuration that was created on server initialization

### First Neighbours

* The first neighbours are computed as follows:
  + The R script findCorrelations.R is run and the correlation matrix (after the p value test) and degrees that were previously stored in R data files are loaded into memory
  + The gene name and side is parsed from the command line arguments and some simple processing is done to determine what the neighbours of that gene are, as well as what their degrees are
  + A new cytoscape configuration is created with the new information and is then sent to the user

### Second Neighbours

* This process is almost identical to that of obtaining the first neighbours except that client sends its current configuration to the server, and the server then appends some new nodes and edges to that configuration

.001 .01 .05 .1

\*\*\*What we need to add is some table that shows various connections. Like a gene name, then arrow, repeat, etc. This will be searchable and can quickly show bioloigsts what pathways there are in a more condensed manner.

\*\*\*Need to prevent the user from messing with the p value once the first neighbours are shown

\*\*\*Need to show the total number of interactions, the number of interactions from epi to stroma, the number of interactions from stroma to epi. We should stick with the current model of computing everything in R and sending the final result back to the server

\*\*\*Background color

---Gene names

\*\*\*Show info about edges and nodes3

\*\*\*Make a better and more biological layout

\*\*\*Bipartite layout with panels, then a large network with colored nodes

\*\*\*Think about how to handle the selection of more than one gene

\*\*\*need to have 5 tabs.

1. The non bipartite view of the network

2. The bipartite view of the network

3. The tables showing self loops, first, and second neighbours

4. Analysis

5. Results

Okay so we need to come up with some more general functions that are in charge of the layout of the graph.

Our createNodes function should really only be creating the data for the nodes, and NOT doing anything with the style or position of them.

Instead of storing configs, we are going to instead store collections of elements. When a user makes a request to the server, we will then create an appropriate config based on the paramters of that request. This will keep us from having to store many different configurations just for different layouts and color schemes.

## May 9

Before we go and start doing in depth testing of the correctness of the display, let’s first make some changes to the front end in terms of aylout, controller organization, partials.

What we need to do now before tweaking the concentric layout is making all of the current methods that query the server send over what kind of layout they want.

I guess since all of this will be moving to another tab, we should work on that tab as well. But for now, we’ll keep all the controls in the first tab on the sidebar to speed up the testing of the layout requesting

Furthermore, we need to fix the neighbour-general path since it is now busted for when the gene that is selected in the second dropdown is not the same as the node selected in the first dropdown.

Once we get all of these done, we can add a slider to filter based on the weight of the edges in the graph. We need to test this thoroughly yo determine whether or not the filtering is something that should be done using ctysocape, or something that should be done using R on the back end.

setwd('C:/Users/Alex/Documents/EpiStroma/R\_Scripts')

load('TE-LGenes.RData')

load('TS-LGenes.RData')

View(edataTE.ERNeg.LGenes)

CorTES.LGenes <- cor(t(edataTE.ERNeg.LGenes),t(edataTS.ERNeg.LGenes))

Signif.TESLGenes<-r.test(54,CorTES.LGenes)

Transparency

Tabs

Correlation slider

Input correlation matrix

tbp ube2c

Also, we need to figure out how to handle showing multiple self-loops when we give the user the ability to select multiple genes

Here is the issue that we are having with the first neighbours

Say we select a gene from epi TBP

Then we select a gene from stroma UBE2C

Our correlation matrix says that

Okay so before we move on to testing and such, let’s take a look at the server from a high level perspective and extract as many methods as possible. We need to make things ultra general and reusable since there wil be many features to add later on. As such, we might need to start making use of the async library.

## Current Server Analysis

We are currently storing collections of cytoscape edges and nodes on the server which will be retierved as requested by the client. This is the most dynamic thing I can think of since when a client requests information, different styles and layouts can be applied to these elements.

Let’s take a look at the methods along this pathway and see if we can generalize anything even further.

### Initialization Path

1. The method initializeServer() is called.
2. initializeSever() then simply calls createAndStoreCorrelationsAndDegrees() with createOverallElements() as a callback function.
3. createAndStoreCorrelationsAndDegrees() is the method that calls the R script which creates the correlation matrices, and the degree lists and stores them in R data files. So I suppose what we could do is remove the callback functionality and instead use the async library.
4. The createOverallElements() is what runs a loop that calls the cacheElementsForPValue() method, as well as the createListOfSelfLoopGenes() method for each of the p values.
5. So cacheElementsForPValue() has a notion of epiDegrees and stromaDegrees. Venkata said that later on we might even be doing graph that are epi-epi. So we might need to make this more genera, but for now I think it’s fine.
6. cacheElementsForPValue() also calls getWeightsAndDegreesFromROutput() which is responsible for parsing the raw JSON output created by the R script and creating a clear object out of that. The issue is the different levels at which we have to index the raw JSON output in order to get the values that we desire. I think that it would be wise to develop some kind of recursive method that iterates over the properties of the outputted object and determines what should be done. I.e., if one of the properties is a list, then we should go one level further and extract the elements of that list. Like when we access the degrees, we go one level further and get the epiDegrees and stromaDegrees from the object.
7. The result of the getWeightsAndDegreesFromROutput() method is then used in the createElements() method. Let’s see if this one can be generalized more.
8. Honestly, it all looks decently flexible. We need to ask Venkata about the cases when we would have epi-epi and stroma-stroma graphs so we can modify our method to support that.

Okay so since we are returning a list from the R output, that list has an attribute called names which we can iterate over. These are the names of the list elelemetns as defined in our R script. These names should be in the same order as the corresponding elements in the values property of the list object returned form the R output.

Instead of returning so many different things from the getWeights and degrees script, perhaps it would be better to create another script that specializes in getting the min and max weights. That way, our code will be much more modular in the future, and our features will be much more easy to test since they will be clearly isolated /separated. However, there is a very unfortunate efficiency tradeoff when having to load the RData files over and over again.

Based on our timing analysis in R, it is the case that dput() takes the most amount of time by far when calling our createAndStoreCorrelatiosnAndDegrees script.

For our 978x978 correlation matrices, it takes about 9.7 seconds to load just a single matrix into memory from an R data file. Again, this is what takes the longest time in our getWeightsAndDegrees script. Therefore, separating the getWeightsAndDegrees script into different scripts might be too computationally expensive if all of the scripts load in the weight matrix.

It looks like the loading of the correlation matrices into memory on Venkata’s computer works very fast. So long as it is the case that the server which we will be hositng the website on will have similar or better performance than his machine, we need not worry about this bottleneck we are having.

Unfortunately, due to the immense size of the correlation matrices when we have 20,000 by 20,000, it will be infeasible to cache the cytoscape elements in memory. Just the correlation matrix itself is 3.75GB. i

Normal

Tumor

Differntial

Epi-Epi

Stroma-Stroma

Epi-Stroma

IF the positive checkbox is checked, then we will be filtering by the value specified in the textbox. If at the same time the negative checkbox isn’t checked,

IF the positive checkbox is checked, we want to get the positive values greater than specified. If it is not checked, then we don’t get any of the positives.

So the last thing that remains to do before we move on to full tilt testing is filtering the graph by degree. The issue that occurs with this is that we probably won’t get the same amount of epi and stroma nodes when we do this. So we’ll have to change our current framework to not just look at the dimnames of the weight matrix. Instead what we need to do is return a list of epi gene names and stroma gene names from the R script.

a e f m

r

t

v

If our current algo were to be fed this matrix, let’s see what the output would be

It would create the edge re, av, rm, ra,

It looks like the algo would work okay. So let’s now modify the getWeightsAndDegreesFilterByWeight function to include filtering by degree. Let’s say that we will be doing this filtering by removing columns and rows from the weight matrix. Once we remove one column, we end up potentially decreasing the degrees of the other nodes. Suppose that we end up removing so many columns that the degrees of our epi genes are now below the threshold. What do we do now?

Say that in the above matrix, the 4th row was removed due to it only containing zeros. Now suppose that we remove columns f and m in order to get rid of genes having degree less than 2.

a e

r

t

v

However, now gene t has only one interaction.

Venkata was right in saying that if either the epiDegree OR stromaDegree for a gene is less than the cutoff, we should remove that gene from both the epi and stroma. This still doesn’t rid us of our problem. We might need a lookahead function of some sort.

It could be best to do this on the client side actually. Let’s see the performance of that.

## Neighbour General Revisited

We need to rewrite the second neighbours function so that it doesn’t need the client to send over the cytoscape elements that were created for the first neighbour. The reason being that if there are a lot of first neighbours, the client has to upload a lot of data which is something that we want to avoid. We changed this so no the first neighbours are recomputed on the server side when the second enighbours are requested.

## Client Side Shared Methods

* The issue with the desire to have multiple tabs with a similar interface is that a lot of code will be repeated. If even one thing in a controller has to be slightly different, a new controller has to be made and therefore a lot of code duplication will occur in terms of all of the methods.
* Obviously if we want to change something on the server side which results in a sclient side change, or if we just want to add an additional feature on the client side, we would have to go into every single controller file and modify code over and over again.
* The only way to avoid this is to make the basic methods shared via a service. Unfortunately, the $scope is not available in services. Additionally, I don’t know if passing around scopes would work, and even if it did it seems like a very shady practice. Therefore, I think the best thing to do would be to create some shared services and instead of having so many variables defined on the scope of each controller, we’ll have the variables defined on the shared service. So instead of $scope.minDegree, we would set a variable like $scope.sharedData = SharedService.data. Then we could access the min degree via $scope.sharedData.minDegree.

Okay so it looks like we are able to pass scope objects to methods in a sharedService, or any type of service quite frankly, and set scope variables within a method in the sharedService.

As far as I can tell, what we gain from doing this is not having to access members on the front end via data.whatever. What we lose by doing this is having to pass the scope to each method which is no big deal really.

## Important AngularJS Shared Methods Info

So there are basically 2 ways of using a service in angularJS in order to allow the sharing of methods across multiple controllers. One way would be to add a parameter to the method that will be the scope of the calling controller. Therefore inside the method, if you want to change variables and have those changes reflected on the front end, all you have to do is scope.variable (notice the missing $).

The other way of doing this would be to create variables in the shared service that will then be used by the controllers. For example, say that we have two tabs, a main tab and a neighbour tab. The main tab uses the MainController while the neighbour tab uses the NeighbourController. Our shared service can have a mainData variable and a neighbourData variable. Each of these variables will be initialized to the same model such as {selfLoops: [], selfLoopsCount: 0}. The method in the shared service that is responsible for modifying the correct data structure will need to take in a flag that specifies whose data we want to modify. Furthermore, on the front end, we would have to set $scope.data = SharedService.mainData in the MainController. Moreover, in the html markup, if we want to access selfLoops, we need to write data.selfLoops.

I think that the first option makes things slightly cleaner on the front end, although you have to keep on defining the same variables for each controller.

You need to do

$scope.selfLoops = [];

$scope.selfLoopsCount = 0;

in every single controller. The second method saves you from having to do this, although you could have a method that can be used for each controller that defines the basic variables that all controllers will be making use of.

We will opt for option 1.

For the degree filter, we’ll just do a table. Epi table and stroma table.