# Week of May 16

## Monday

### Morning

We will dedicate the morning to testing out the following functionalities:

1. Test first neighbours

2. Test second neighbours

3. Test the genes of interest

4. Test the filtering by edge weight

5. Test varying the layout

6. Test varying the p value

We need to do this for multiple choices of P Values.

### Afternoon

WE might want it to be the case that the edge weight filtering applies to not only the main graph, but the children, as well as the genes of interest graph.

In fact, it might be a good idea to slowly phase out the main graph since we will not be using it later on at all due to the complexity of the 20k by 20k matrix.

We have generalized the neighbor general method so that it now allows you to keep asking for neighbours and doesn’t limit you to only the 2nd neighbours. We need to extend the front end in order to permit the user to keep on exploring neighbours. It is still however made to work for only an epi-stroma correlation matrix. We need to come up with a scheme that will work got epi-epi as well.

IT would be nice to have the circular layouts done by the time that Venkata gets back.

In order to phase out the overall graph and start focusing on the graph that shows the genes of interest, we need to create a path on the server that is simply responsible for getting a list of genes as well as their degrees.

Okay so let’s create the R script and the necessary server side code in order to accomplish this.

Then we will move on to changing the front end so that the filtering happens on the graph that is returned to the client after they select a list of genes of interest.

Let’s see what the performance of the md-autocomplete control is like when it contains 20,000 genes

Let’s test out the functionality of getting beyond just the second neighbours. We will implement this by having a list similar to the genes of interest list. WE’ll leave it to the ng-repeat to tell the user which nodes are selected.

Unfortunately, our logic breaks down after a few levels since of the conditions we implemented to ensure that there won’t be duplicate nodes in the graph are preventing necessary edges from being added. What we can do about this is add a check that

Goes back and searches the previous neighbours to see if one of our exclusions for the current neighbor is in there. IF so, we’ll add an edge from one of those previous neighbours to the current node of interest. The only issue with this approach is that it will make the graph somewhat messy. Furthermore, this approach doesn’t work under the current framework since we are only ever using a single source node at each level. Let’s come back to this later.

One thing we can do right now is make a layout for the selected genes graph. Okay so here is the ideal layout:

We need to highlight the genes that the user selected. Unfortunately, we can’t just have a fixed policy that these genes are to go in the middle of the graph. Our position policy should be based on minimizing edge overlap. I’m not sure if it’s even possible to have a layout where no edges cross eacht other for this kind of graph. From a theoretical point of view, I don’t think it’s possible to completely avoid edge overlap in the genes of interest graph. Having said that, we can try to minimize the amount of overlapping edges. Since we have control over node position and size, there probably isn’t a unique way in which to accomplish this task. Let’s try making some sort of concentric layout and see where that takes us.

Another issue that has emerged is that our self-loop logic is currently flawed due to the fact that we are no longer caching graph elements. We can simply move this logic to the getRelevatnSubmatrix R script and send a list of genes back to the client in addition to the cytoscape config that we are sending back. However, we would also have to do this for any other script that we are to write in the future. Already, the list that we are returning from R to the server is quite complicated and adding more elements to it will lead to confusion in the future as well as a lack of maintainability. We should leave the self-loop computation to the client side.

Now that we’ve added the self-loop computation to the client side in addition to adding filtering to the self -oops as well as interactions, let’s clean up the code on the server and also come up with a concentric layout for our genes of interest graph.

Removed some of the unecesary paths for the overall graph which is now obsolete. Now it’s time to make the code that calls the exec() function cleaner. The first thing we can do for this is create a function that builds an arugment string given an array of arguments.

Before moving on to making the clustered layout, we need to decide how we’re going to reconcile the different ways of creating nodes and edges currently. At the moment, we have two ways: one way uses a concept of source nodes and creating edges from that node to specified neighbouring nodes. The other way makes use of a correlation matrix and generates nodes based on the column and row names of that matrix, and it creates edge wherever there is an entry in the matrix not equal to zero. Now obviously this correlation matrix approach makes more sense due to the fact that it stays true to the original representation of the network in R as a correlation matrix. However, it limits us because we don’t know anything about the neighbours. For example, when we specify some genes of interest and get back a correlation matrix, we wouldn’t be able to create a clustered layout. This is because we only know the nodes that the user selected, but we don’t know which of the nodes in the graph are their first neighbours, and which are their second neighbours.