

Meta-analyses of data from two (or more) microarray data sets.

Jeremy Miller

jeremyinla@gmail.com

Microarrays provide expression levels for thousands of genes at once, and therefore have been used extensively to study transcription in the brain. In many cases, the end point for these studies is differential expression analysis: genes A-G are increased and genes H-P are decreased in disease X. Another method for analysis, which is becoming increasingly more common, is gene coexpression analysis: genes Q-S have similar expression patterns. WGCNA is a very useful method for studying gene coexpression, and everything necessary to perform WGCNA successfully can be found at the WGCNA library website:

<http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/>

Given the large number of microarray analyses (sometimes of similar design) one question that may arise is "if group A and group B both ran microarray studies and reported some results, how compatible are these results?" There are currently no standard methods for comparing results from multiple microarray data sets, but that does not mean that it can't be done. Some methods can be found at the WGCNA website (above). Other comparison methods are listed in this document below, which is a condensed version of the analysis performed in "Miller JA, Horvath S, Geschwind DH. (2010) *Divergence of human and mouse brain transcriptome highlights Alzheimer disease pathways*. Proc Natl Acad Sci U S A. 2010 Jul 13;107(28):12698-703."

Step 1: Getting/loading what you will need for this analysis.

1) Download and install "R" from here: <http://cran.r-project.org/>

2) Start R, then install the necessary packages in R, using the following commands:

```
install.packages(c("impute", "dynamicTreeCut", "qvalue", "flashClust", "Hmisc", "WGCNA"))
```

(If you need other packages, R will tell you. If the packages don't load, try to check the permissions on your computer, or visiting the WGCNA library website above.)

3) Download "metaAnalysisFiles.zip" from [here](#), unzip it, and make this folder is your current working directory in R. In this file is "metaAnalysisData.RData", which you need to load it into R:

```
load("metaAnalysisData.RData")
```

The variables contained in this file are as follows:

- datExprA1 and datExprA2 – two data sets from the Illumina human ref-12 platform
- datExprB1 and datExprB2 – data sets from the Illumina human ref-12 and Affymetrix HG-U133A platforms, respectively (datExprA1 and datExprB1 are the same).
- probesI/A – probe set IDs for human ref-12 and HG-U133A platforms
- genesI/A – gene symbols corresponding to probesI/A

Alternatively, you can read in your own data (but make sure that in all of your expression variables, **ROWS** correspond to probes and **COLUMNS** correspond to samples).

4) You will need the following files: "tutorialFunctions.R", "exampleListInput.csv", "exampleMMInput.csv", and maybe "collapseRows_NEW.R" which should already be in your current directory.

Step 2: Pre-processing your data sets to ensure that they are comparable

Once you have your data, the first step is to preprocess your data. There are a number of ways of doing that (which will not be discussed here). All of the expression variables in "metaAnalysisData.RData" are already pre-processed.

The next step is to ensure that your variables are comparable. For this step, you will find yourself in one of two situations:

A) Your data sets of interest are from the same platform. If this is the case, congratulations, your data sets are already comparable! You can follow the suggestion in the next step, but it is not necessary.

B) Your data sets of interest are from different platforms. If this is the case, you need to match your probes in some way. The easiest way to do this is to choose one probe for each gene in each data set based on gene symbol (I = Illumina, A = Affymetrix), then use gene symbol in place of probe ID as identifiers in each data set. To do this, run this code:

```
library(WGCNA) # (Section will take ~5-10 minutes to run)
# source("collapseRows_NEW.R") # ONLY uncomment this line if you get an error with it commented
datExprB1g = (collapseRows(datExprB1,genesI,probesI))[[1]]
datExprB2g = (collapseRows(datExprB2,genesA,probesA))[[1]]
```

Once you have comparable data, you need to limit your analysis to genes/probes that are expressed in both data sets.

```
commonProbesA = intersect (rownames(datExprA1),rownames(datExprA2))
datExprA1p = datExprA1[commonProbesA,]
datExprA2p = datExprA2[commonProbesA,]

commonGenesB = intersect (rownames(datExprB1g),rownames(datExprB2g))
datExprB1g = datExprB1g[commonGenesB,]
datExprB2g = datExprB2g[commonGenesB,]
```

Now, every row in the data files for comparison corresponds to the same probe/gene.

Step 3: Correlating general network properties

A quick way to assess the comparability of two data sets is to correlate measures of average gene expression and overall connectivity between two data sets. The higher the correlations of these properties, the better chance you will have of finding similarities between the two data sets at subsequent stages of the analysis.

```
softPower = 10 # (Read WGCNA tutorial to learn how to pick your power)
rankExprA1= rank(rowMeans(datExprA1p))
rankExprA2= rank(rowMeans(datExprA2p))
random5000= sample(commonProbesA,5000)
rankConnA1= rank(softConnectivity(t(datExprA1p[random5000,]),type="signed",power=softPower))
rankConnA2= rank(softConnectivity(t(datExprA2p[random5000,]),type="signed",power=softPower))

rankExprB1= rank(rowMeans(datExprB1g))
rankExprB2= rank(rowMeans(datExprB2g))
random5000= sample(commonGenesB,5000)
rankConnB1= rank(softConnectivity(t(datExprB1g[random5000,]),type="signed",power=softPower))
rankConnB2= rank(softConnectivity(t(datExprB2g[random5000,]),type="signed",power=softPower))

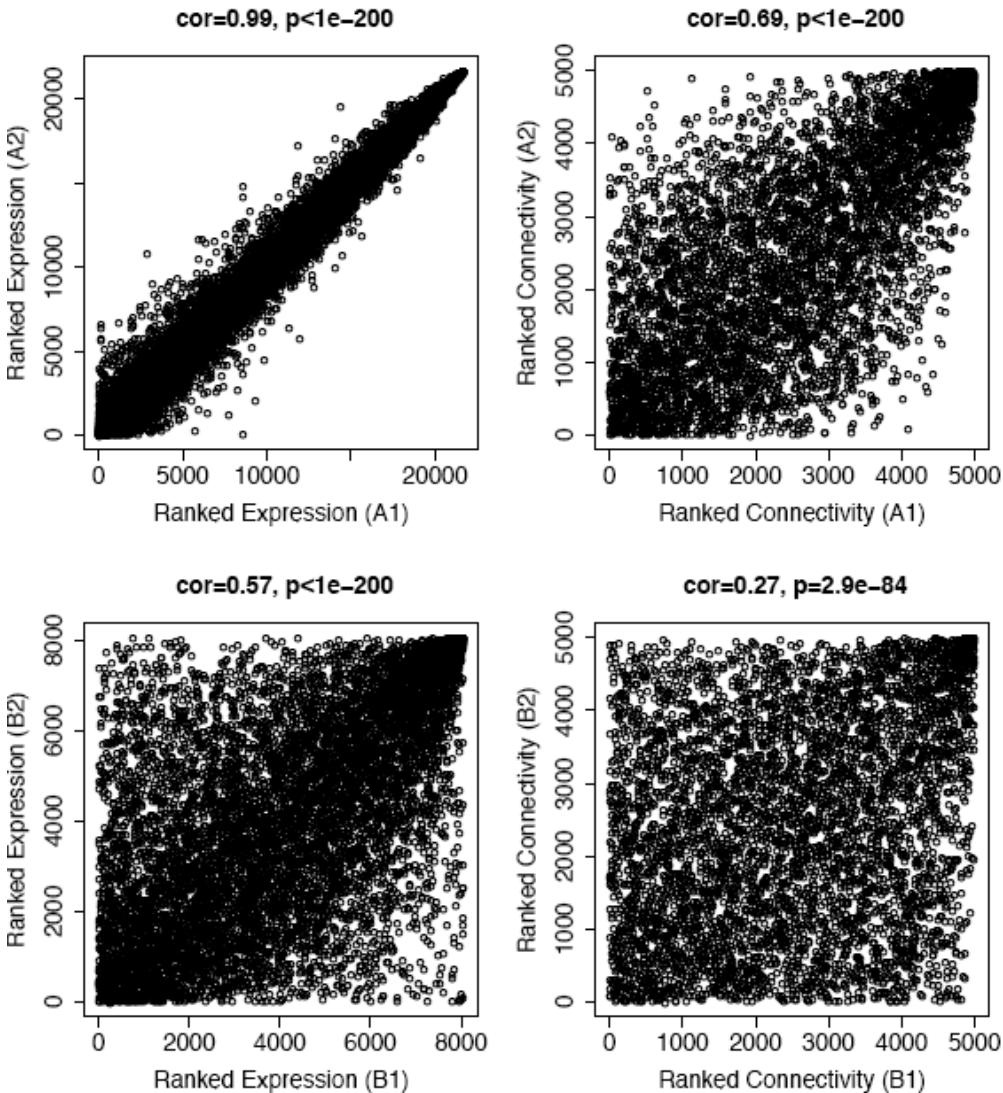
pdf("generalNetworkProperties.pdf", height=10, width=9)
```

```

par(mfrow=c(2,2))
verboseScatterplot(rankExprA1,rankExprA2, xlab="Ranked Expression (A1)",
  ylab="Ranked Expression (A2)")
verboseScatterplot(rankConnA1,rankConnA2, xlab="Ranked Connectivity (A1)",
  ylab="Ranked Connectivity (A2)")
verboseScatterplot(rankExprB1,rankExprB2, xlab="Ranked Expression (B1)",
  ylab="Ranked Expression (B2)")
verboseScatterplot(rankConnB1,rankConnB2, xlab="Ranked Connectivity (B1)",
  ylab="Ranked Connectivity (B2)")
dev.off()

```

If you now open up this pdf file, you will see the following plots:



Notice three things:

- 1) The correlations are positive and the p-values are significant in all cases. This suggests that the data sets are comparable.
- 2) The correlations and p-values are better for expression than for connectivity. This is consistent with many studies.
- 3) The correlations and p-values for A are better than for B. This is because the two A data sets were run using the same platform. Thus data sets from different platforms are less comparable than data sets from the same platform, *but they are still comparable*.

From now through step 7, the analysis is the same for data sets run on the same platform as those run on different platforms. Since the results are more significant for the within-platform comparisons, the remainder of this tutorial (through step 7) will focus on data sets A1 and A2.

Step 4: Run WGCNA on the data sets.

So computational reasons and for simplicity we first will choose the top 5000 most expressed probes in data set A1 (*normally you wouldn't do this*) and then keep only 1 probe per gene (as above), leaving a total of 4746 genes.

```
keepGenesExpr = rank(-rowMeans(datExprA1p)) <= 5000
datExprA1g    = datExprA1p[keepGenesExpr,]
datExprA2g    = datExprA2p[keepGenesExpr,]
keepGenesDups = (collapseRows(datExprA1g, genesI, probesI)) [[2]]
datExprA1g    = datExprA1g[keepGenesDups[,2],]
datExprA2g    = datExprA2g[keepGenesDups[,2],]
rownames(datExprA1g) <- rownames(datExprA2g) <- keepGenesDups[,1]
```

Next, we will calculate all of the necessary values to run WGCNA. For a tutorial on what all of these variables mean, please see the WGCNA website (listed above).

```
adjacencyA1 = adjacency(t(datExprA1g), power=softPower, type="signed");
diag(adjacencyA1)=0
dissTOMA1    = 1-TOMsimilarity(adjacencyA1, TOMType="signed")
geneTreeA1   = flashClust(as.dist(dissTOMA1), method="average")

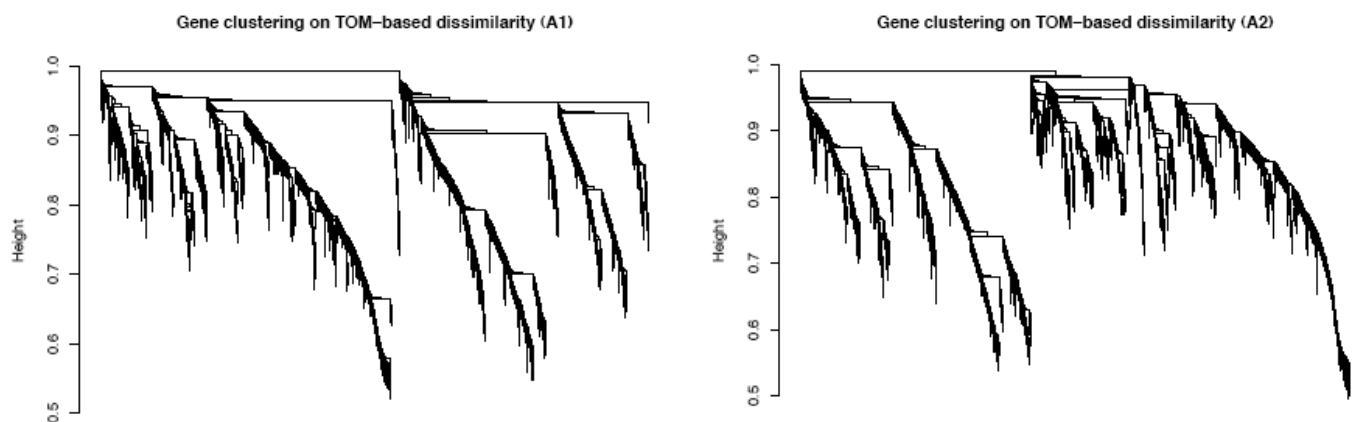
adjacencyA2 = adjacency(t(datExprA2g), power=softPower, type="signed");
diag(adjacencyA2)=0
dissTOMA2    = 1-TOMsimilarity(adjacencyA2, TOMType="signed")
geneTreeA2   = flashClust(as.dist(dissTOMA2), method="average")

save.image("tutorial.RData") # (Section will take ~5-15 minutes to run)
# NOTE: This file will be ~700GB and is not required, but if your computer crashes, you can type:
# ' load("tutorial.RData") ' to restart at this point.
```

Here is code to display the networks visually:

```
pdf("dendrogram.pdf", height=6, width=16)
par(mfrow=c(1,2))
plot(geneTreeA1, xlab="", sub="", main="Gene clustering on TOM-based dissimilarity (A1)",
     labels=FALSE, hang=0.04);
plot(geneTreeA2, xlab="", sub="", main="Gene clustering on TOM-based dissimilarity (A2)",
     labels=FALSE, hang=0.04);
dev.off()
```

If you now open up this pdf file, you will see the following plots:



These are "good" data, since there are a lot of distinct branches.

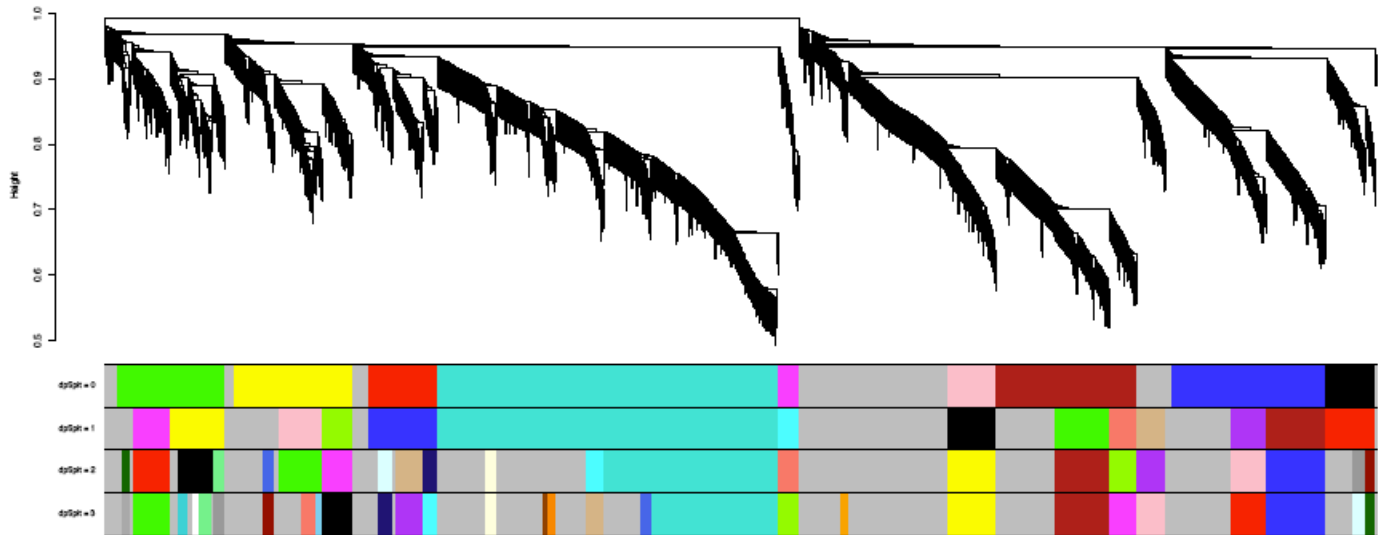
Next we will determine modules based on data set A1 (in a real situation, you would usually determine modules in your "control" data set). We use this code to automatically display four different module splits that we can choose from.

```

mColorh=NULL
for (ds in 0:3){
  tree = cutreeHybrid(dendro = geneTreeA1, pamStage=FALSE,
    minClusterSize = (30-3*ds), cutHeight = 0.99,
    deepSplit = ds, distM = dissTOMA1)
  mColorh=cbind(mColorh,labels2colors(tree$labels));
}
pdf("Module_choices.pdf", height=10,width=25);
plotDendroAndColors(geneTreeA1, mColorh, paste("dpSplt =",0:3), main = "",dendroLabels=FALSE);
dev.off()
modulesA1 = mColorh[,1] # (Chosen based on plot below)

```

If you now open up this pdf file, you will see the following plots:



I chose to use `deepsplit=0` (the top row) so that there will be a small number of large modules. Depending on the purpose of the analysis, it is sometimes preferable to choose a larger number of small modules—in such a case, `deepsplit` values of 1-3 would be chosen.

Next we calculate the principle components for visualizations (as well as for the quantitative assessments, which are presented later). The first PC is referred to as the module eigengene (ME), and is a single value that represents the highest percent of variance for all genes in a module. In other words, if we show the ME for module X doing something, there is a good chance that most genes in module X also do that same thing.

```

PCs1A = moduleEigengenes(t(datExprAlg), colors=modulesA1)
ME_1A = PCs1A$eigengenes
distPC1A = 1-abs(cor(ME_1A,use="p"))
distPC1A = ifelse(is.na(distPC1A), 0, distPC1A)
pcTree1A = hclust(as.dist(distPC1A),method="a")
MDS_1A = cmdscale(as.dist(distPC1A),2)
colorsA1 = names(table(modulesA1))

save.image("tutorial.RData")

pdf("ModuleEigengeneVisualizations.pdf",height=6,width=6)
par(mfrow=c(1,1), mar=c(0, 3, 1, 1) + 0.1, cex=1)

plot(pcTree1A, xlab="", ylab="", main="", sub="")

plot(MDS_1A, col= colorsA1, main="MDS plot", cex=2, pch=19)

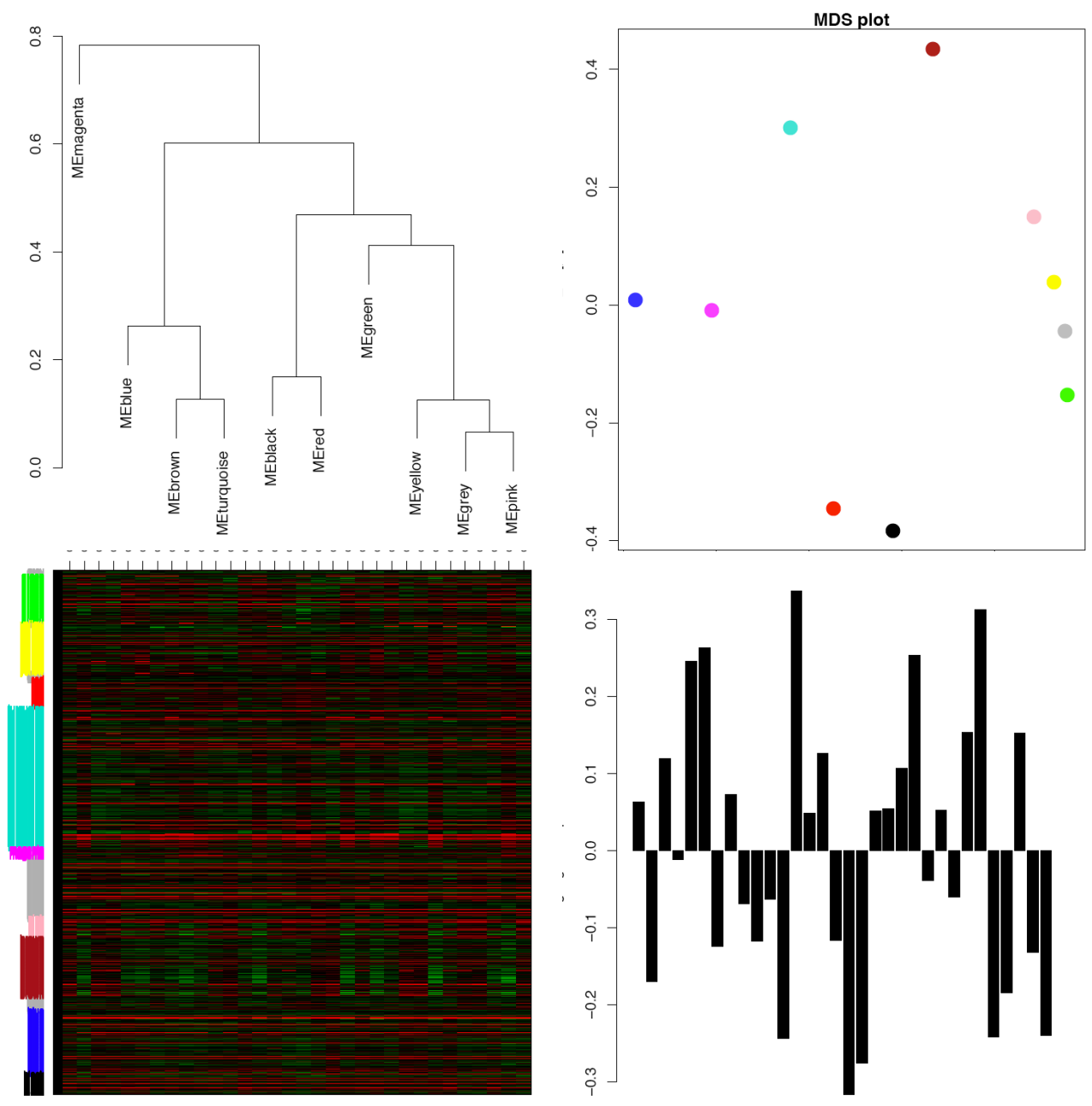
ordergenes = geneTreeA1$order
plot.mat(scale(log(datExprAlg[ordergenes,])) , rlabels= modulesA1[ordergenes], clabels=
colnames(datExprAlg), rcols=modulesA1[ordergenes])

```

```

for (which.module in names(table(modulesA1))){
  ME = ME_1A[, paste("ME",which.module, sep="")]
  barplot(ME, col=which.module, main="", cex.main=2,
    ylab="eigengene expression",xlab="array sample")
};
dev.off();

```



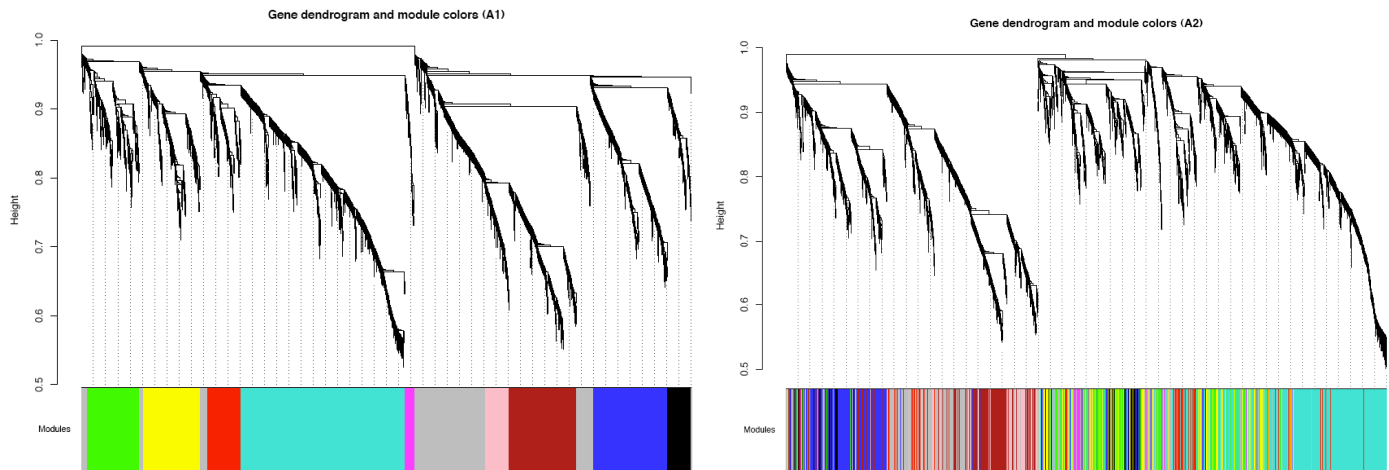
The upper left box is a dendrogram of the module eigengenes: modules in the same branches have relatively similar expression (but not as similar as the expression of genes within a single module). The upper right box is a multidimensional scaling (MDS) plot of the module eigengene, where the x-axis is the first PC and the y-axis is the second PC: modules that group together have relatively similar expression. The bottom left plot is a heat map of all of the genes sorted as they were clustered above (y-axis). Each column represents a sample (x-axis), which could be sorted based on phenotypic information or based on clustering, but in this case aren't sorted at all. Finally, the bottom right plot shows the ME expression for the black module (there is a similar plot for each of the other modules). As with the heat map, the samples (bars) could be ordered and labeled, but aren't.

Step 4: Qualitatively and quantitatively measure network preservation at the module level.

Now that we have all of our WGCNA variables as well as our module definitions, we can start assessing how well our modules in network A1 are preserved in network A2. As a qualitative assessment, we impose the modules from A1 on the network for data set A2, and then plot the resulting networks.

```
pdf("Final_modules.pdf",height=8,width=12)
plotDendroAndColors(geneTreeA1, modulesA1, "Modules", dendroLabels=FALSE, hang=0.03, addGuide=TRUE,
  guideHang=0.05, main="Gene dendrogram and module colors (A1)")
plotDendroAndColors(geneTreeA2, modulesA1, "Modules", dendroLabels=FALSE, hang=0.03, addGuide=TRUE,
  guideHang=0.05, main="Gene dendrogram and module colors (A2)")
dev.off()
```

If you now open up this pdf file, you will see the following plots (A1 is on the left, A2 on the right):



We can see from the fact that these module labels still group together in A2 (right) that there is very good preservation. It is important to note that it is often not possible to see an obvious grouping of model labels in the second data set, even if there is significant module preservation.

To quantify this result, we take advantage of another function built into the WGCNA library: `modulePreservation`. This function assesses how well a module in one study is preserved in another study using a number of strategies and (for the purposes of this tutorial) outputs a single Z-score summary.

```
# (This step will take ~10-30 minutes)
multiExpr = list(A1=list(data=t(datExprA1g)),A2=list(data=t(datExprA2g)))
multiColor = list(A1 = modulesA1)
mp=modulePreservation(multiExpr,multiColor,referenceNetworks=1,verbose=3,networkType="signed",
  nPermutations=30,maxGoldModuleSize=100,maxModuleSize=400)
stats = mp$preservation$Z$ref.A1$inColumnsAlsoPresentIn.A2
stats[order(-stats[,2]),c(1:2)]
```

The output on the screen will look like this:

	moduleSize	Zsummary.pres
brown	400	60.88719
pink	181	43.33866
turquoise	400	29.86704
blue	400	29.46645
green	400	22.72069
red	260	19.22092
black	183	17.42699
yellow	400	17.26732
magenta	75	17.00574
grey	400	13.55119
gold	100	11.10579

In general, the higher the value of "Zsummary.pres" the more preserved the module is between data sets: $5 < Z < 10$ indicates moderate preservation, while $Z > 10$ indicates high preservation. (See the WGCNA website for a more complete tutorial on this function.) The "grey" module contains uncharacterized gene while the gold module contains random genes. In general, these modules should have lower Z-scores than most of the other modules. In this case, we find that all modules are very well preserved.

Step 5: Module membership (kME) and its use in comparing networks

kME is a useful value, in that it can be used to measure correlations between each gene and each ME, and thus even genes which were not initially assigned to a module can be included in between-network comparisons.

We first will get the kME values, along with their associated p-values for A1 and will then output the resulting table to a file ("kMEtable1.csv").

```
geneModuleMembership1 = signedKME(t(datExprA1g), ME_1A)
colnames(geneModuleMembership1)=paste("PC",colorsA1,".cor",sep="");

MMPvalue1=corPvalueStudent(as.matrix(geneModuleMembership1),dim(datExprA1g)[[2]]);
colnames(MMPvalue1)=paste("PC",colorsA1,".pval",sep="");

Gene          = rownames(datExprA1g)
kMEtable1     = cbind(Gene,Gene,modulesA1)
for (i in 1:length(colorsA1))
  kMEtable1 = cbind(kMEtable1, geneModuleMembership1[,i], MMPvalue1[,i])
colnames(kMEtable1)=c("PSID", "Gene", "Module", sort(c(colnames(geneModuleMembership1),
colnames(MMPvalue1))))

write.csv(kMEtable1,"kMEtable1.csv",row.names=FALSE)
```

Now repeat for A2, using the module assignments from A1 to determine kME values.

```
# First calculate MEs for A2, since we haven't done that yet
PCs2A = moduleEigengenes(t(datExprA2g), colors=modulesA1)
ME_2A = PCs2A$eigengenes

geneModuleMembership2 = signedKME(t(datExprA2g), ME_2A)
colnames(geneModuleMembership1)=paste("PC",colorsA1,".cor",sep="");

MMPvalue2=corPvalueStudent(as.matrix(geneModuleMembership2),dim(datExprA2g)[[2]]);
colnames(MMPvalue2)=paste("PC",colorsA1,".pval",sep="");

kMEtable2 = cbind(Gene,Gene,modulesA1)
for (i in 1:length(colorsA1))
  kMEtable2 = cbind(kMEtable2, geneModuleMembership2[,i], MMPvalue2[,i])
colnames(kMEtable2)=colnames(kMEtable1)

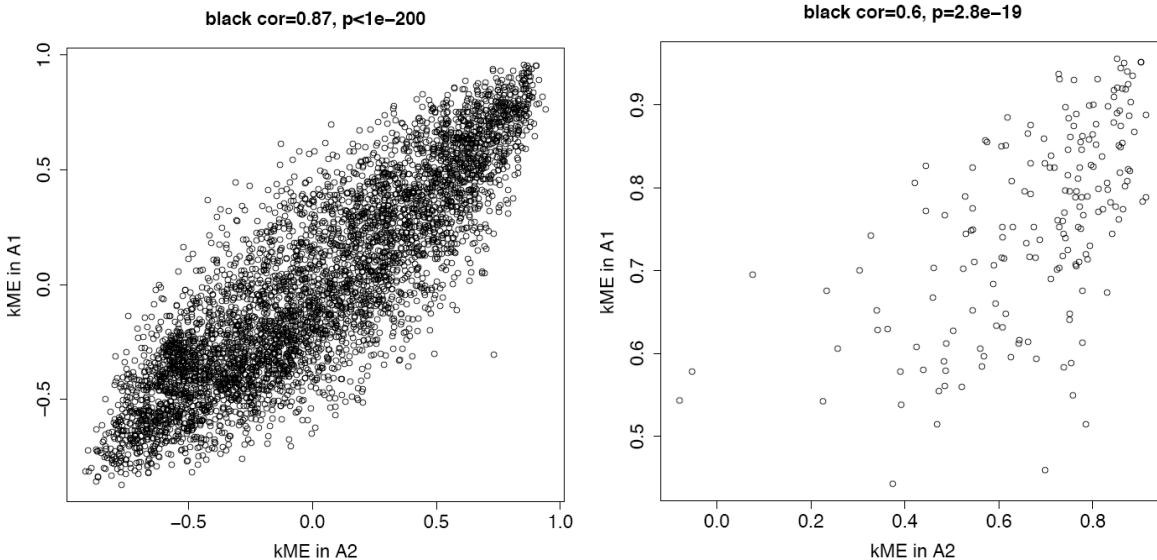
write.csv(kMEtable2,"kMEtable2.csv",row.names=FALSE)
```

Now that we have kME values for both networks, there are a few additional ways in which we can compare the resulting networks.

The first thing we can do is plot the kME values of each gene in A1 against the corresponding kME values of each gene in A2. Modules with points showing a high correlation are highly preserved. Example images for creating these plots for both all genes (left) and the subset of genes originally assigned to a given module (right) are presented below. Here is the code:


```
pdf("all_kMetable2_vs_kMetable1.pdf",height=8,width=8)
for (c in 1:length(colorsA1)){
  verboseScatterplot(geneModuleMembership2[,c],geneModuleMembership1[,c],main=colorsA1[c],
    xlab="kME in A2",ylab="kME in A1")
}; dev.off()

pdf("inModule_kMetable2_vs_kMetable1.pdf",height=8,width=8)
for (c in 1:length(colorsA1)){
  inMod = modulesA1== colorsA1[c]
  verboseScatterplot(geneModuleMembership2[inMod,c],geneModuleMembership1[inMod,c],main=colorsA1[c],
    xlab="kME in A2",ylab="kME in A1")
}; dev.off()
save.image("tutorial.RData") # (optional line of code)
```



(Similar plots would be outputted for all modules)

These two types of plots convey similar, but not identical, information. Using all genes (left) allows one to include all positively and negatively correlated genes, but often also includes a lot of noise (although not in this case). Using only in-module genes (right) is a visual way of assessing hub gene conservation: if these genes show between-set correlation, then the genes in the upper right of the plot are likely to be common hub genes between data sets. (Hub genes are genes that show significant correlation with MEs and high within-module connectivity and will be discussed below.)

The second thing we can do is determine which genes are hubs in both networks by determine which genes have extremely high kME values in both networks.

```
topGenesKME = NULL
for (c in 1:length(colorsA1)){
  kMERank1 = rank(-geneModuleMembership1[,c])
  kMERank2 = rank(-geneModuleMembership2[,c])
  maxKMERank = rank(apply(cbind(kMERank1,kMERank2+.00001),1,max))
  topGenesKME = cbind(topGenesKME, Gene[maxKMERank<=10])
}; colnames(topGenesKME) = colorsA1
topGenesKME
```

	black	blue	brown	green	grey	magenta	pink	red	turquoise	yellow
[1,]	"15-Sep"	"ACTR10"	"CADPS"	"CYBRD1"	"CAMKV"	"AHS1"	"BSN"	"ABHD8"	"ANKRD30B"	"C9orf19"
[2,]	"Clorf19"	"ATP50"	"GABBR2"	"DPYSL3"	"CYFIP2"	"DNAJA1"	"C19orf4"	"CCDC106"	"C9orf130"	"CLIC4"
[3,]	"DAZAP2"	"COX6B1"	"MAP7D2"	"FERMT2"	"DLG2"	"DNAJB1"	"CACNA1E"	"CCM2"	"CEP27"	"CNTN2"
[4,]	"FN1A"	"KIAA1279"	"NAPB"	"GRAMD3"	"DNM1"	"DNAJB6"	"DNM1"	"CELSR3"	"CSF2RA"	"CREB5"
[5,]	"LITAF"	"LOC646630"	"NECAP1"	"HEPACAM"	"HSPA12A"	"FKBP4"	"EEF1A2"	"EGLN2"	"IL10"	"CTNNA1"
[6,]	"LOC148915"	"PRDX3"	"NSF"	"IL17D"	"MAP1A"	"HSPA1A"	"ELOVL4"	"FAM53C"	"LOC644250"	"KIAA0494"
[7,]	"LOC440926"	"TBCA"	"PGM2L1"	"LOC219854"	"NCDN"	"HSPA1B"	"FLJ33996"	"FLJ25404"	"SDHALP1"	"LASS2"
[8,]	"LOC730534"	"TMEM85"	"PREPL"	"MAPRE1"	"PNCK"	"HSPB1"	"LPHN1"	"PARD6A"	"SLC4A5"	"PLEKHG3"
[9,]	"SFT2D1"	"TXN"	"SLC9A6"	"NTRK2"	"SCAMP5"	"HSPD1"	"SNAP91"	"PLXNA1"	"ZNF652"	"PLEKHH1"
[10,]	"SGCE"	"WRB"	"WDR7"	"RHOTB3"	"UNC5A"	"STIP1"	"THY1"	"ZER1"	"ZNF786"	"VAMP3"

These genes represent the top 10 genes per module based on kME in both networks.

Step 6: Comparing networks and annotating modules using programs outside of R

The remainder of this document include methods that involving using R in combination with other methods to compare networks. This tutorial may be insufficient to complete the parts of this analysis that are done outside of R; similarly, some of these functions (indicated with a comment) should be considered "beta" and are sometimes unreliable. That said, this step can be taken as a guideline for things to try when comparing multiple networks (or even when analyzing data from a single network).

In order to **visualize our resulting network modules**, we can output data from our network for import into VisANT, which is a standalone visualization program that is available at <http://visant.bu.edu/>. VisANT will allow you to visually see the hub genes in a module, and is the program that most people in the lab use to make the module pictures that end up in publications. Tutorials for how to use VisANT are available on the website, or you can ask someone. Although VisANT is available both for Mac and PC, I have found that the PC version works better. Here is the code:

```
source("tutorialFunctions.R")
for (co in colorsA1[colorsA1!="grey"])
  visantPrepOverall(modulesA1, co, t(datExprA1g), rownames(datExprA1g), 500, softPower, TRUE)
```

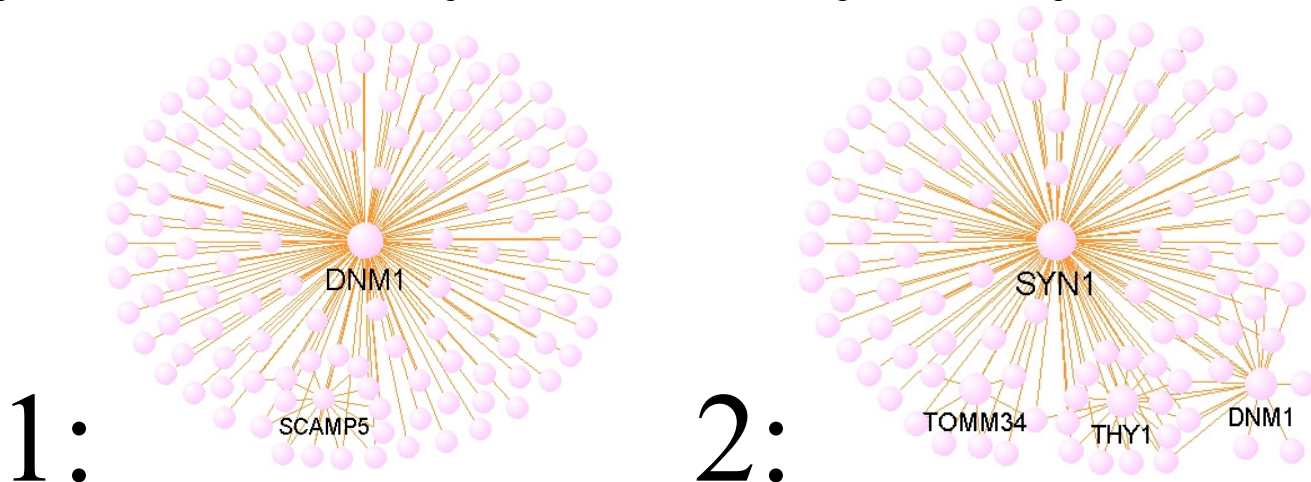
This function will output a lot of files into your current directory, which you should immediately put into a subfolder called "visantFilesA1". For each module you will have two files:

- 1) <module>_connectivityOverall.csv: this file contains a list of all of the genes in a given module sorted from the highest to the lowest intramodule connectivity (k_{in}), along with their mean expression. So the genes at the top of the list would be hub genes.
- 2) <module>_visantOverall.csv: this file contains all of the input you will need for VisANT. Note that only the first five columns should be copied into VisANT. The first two columns represent genes that have the highest topological overlap (column 5) in the module. Column 3 must be "0" and column 4 is the color of the lines to draw ("M1002" is orange).

After you have copied these files to "visantFilesA1", run the same thing for the second data set:

```
for (co in colorsA1[colorsA1!="grey"])
  visantPrepOverall(modulesA1, co, t(datExprA2g), rownames(datExprA2g), 500, softPower, TRUE)
```

This files are equivalent to the files above and should be copied to a folder called "visantFilesA2". In addition to using VisANT for module annotation, in modules with high preservation it is sometimes (albeit rarely) possible to look at these module depictions and see common hub genes. For example:



Notice that DNM1 is a hub gene in the pink module in both A1 and A2.

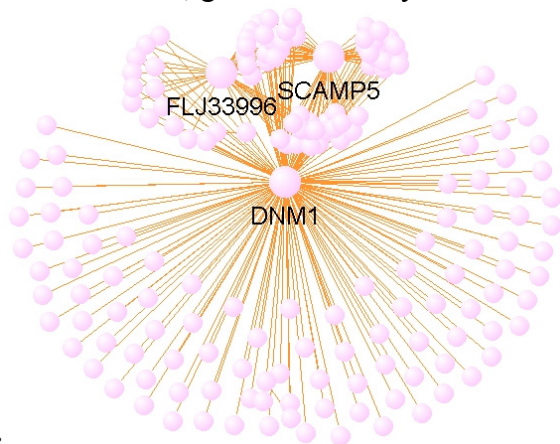
Hub genes specific to one network can also be found using VisANT. After copying the above files to "visantFilesA2", run this code, then transfer the resulting files to "visantFilesA12":

```
datExprA12g = t(cbind(datExprA1g,datExprA2g))
i1 = 1:dim(datExprA1g)[[2]];
i2 = (1:dim(datExprA2g)[[2]])+length(i1)
for (co in colorsA1[colorsA1!="grey"])
  visantPrep(modulesA1, co, i1, i2, datExprA12g, rownames(datExprA1g), 500, softPower, TRUE)
```

The output files from this function are similar to those from the previous code, except that the connectivity files include k_{in} for both A1 and A2, and the VisANT file requires some editing before it can be input into VisANT (for example, the "0" and the color columns are missing, and columns A, C, E, F, H, and I are extraneous).

Genes specific to A1 can be found two ways:

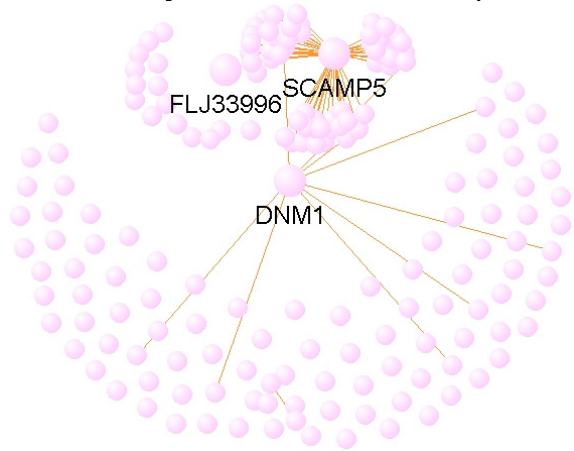
- 1) In the connectivity files, genes with high "kin_CI" and low "kin_MS" are only hubs in A1.
- 2) In the visant files, genes with many connections with "TO_Ratio" are only hubs in A1. For example:



ALL:

A1-specific:

In this case we find that SCAMP5 is a hub specific to A1, since many connections are only present in A1.



Next, we use EASE to annotate modules based on Gene Ontology (GO) enrichment. EASE (which can be found at <http://david.abcc.ncifcrf.gov/ease/ease1.htm>) is a standalone version of DAVID (which can be found at <http://david.abcc.ncifcrf.gov/>), and is only available for the Windows. Also note that there is a function in the WGCNA library called "GOenrichmentAnalysis" that also runs GO enrichment, but I have not used it before.

Create a new folder called "geneLists_A12" then run this code:

```
folder = "geneLists_A12/"
for (c in colorsA1){
  fn = paste(folder, c, ".txt", sep="");
  write.geneList(Gene[modulesA1==c], fn)
};
write(Gene, paste(folder, "all.txt", sep=""))
```

Your folder should now have files titled "<color>.txt" that contain the gene names of every gene in that module, as well as a file titled "all.txt" that has every gene in your data set. This "all.txt" is what you should use as your population file when asked in EASE (or DAVID). In EASE, I would suggest saving your output as an excel file, as that is much easier to work with than the default (which is a website). Otherwise, EASE is relatively straightforward to use.

We can also annotate modules based enrichment for user-defined lists. GO is a good system for finding genes involved in specific cellular components, biological processes, and molecular functions; however, GO is lacking in its annotation of things like cell-type-specific and disease-related genes. Furthermore, sometimes it is advantageous to compare your modules to gene lists from related publications, from previous experiments done in the lab, etc. To do that, input the following code:

```
enrichments = userListEnrichment(Gene, modulesA1, c("exampleListInput.csv", "exampleMMInput.csv"),
                                c("cellType", "humanModules"), "enrichment.csv")
```

There are two types of files that are accepted for input into the function "userListEnrichment":

- 1) Gene lists (for example: "exampleListInput.csv"): this type of file must have at least two columns with the column headers "Gene" and "Var1". The first column should contain gene lists corresponding to user-defined categories, while the second column should have descriptors of these categories. Any number of categories can be contained in a single file.
- 2) Module Membership tables (for example: "exampleMMInput.csv"): this type of file must have at least three columns labeled "PSID", "Gene", and "Module", respectively (the remaining columns are not used). From these tables, modules are treated as categories.

This function will write all of the significant enrichments to "enrichment.csv", which looks like this:

InputCategories	UserDefinedCategories	CorrectedPvalues
green	humanModules_brown	1.77E-116
yellow	humanModules_blue	3.25E-65
green	cellType_Astrocyte_Cahoy_all	1.35E-63
brown	cellType_Neuron_Cahoy_all	3.47E-59
pink	cellType_Neuron_Cahoy_all	1.08E-31
brown	humanModules_red	5.80E-27
blue	humanModules_red	1.75E-23
pink	humanModules_green	1.33E-18
yellow	cellType_Oligodendrocyte_Cahoy_all	7.12E-18
brown	humanModules_tan	1.51E-15
pink	humanModules_greenyellow	7.05E-15
brown	humanModules_green	8.27E-14
green	humanModules_pink	4.93E-13
black	humanModules_blue	6.79E-13
turquoise	humanModules_NA	3.12E-11
blue	humanModules_cyan	9.53E-10
brown	humanModules_black	1.60E-09
turquoise	humanModules_magenta	8.04E-09
black	humanModules_salmon	2.35E-06
yellow	humanModules_salmon	4.38E-06
black	cellType_Oligodendrocyte_Cahoy_all	0.000168523
red	humanModules_turquoise	0.000458179
brown	humanModules_grey	0.000752197
blue	humanModules_grey	0.001838766
grey	humanModules_NA	0.00836362

Note that all of the category enrichments (both significant and not) are saved in the variable `enrichments$pValues`. To see which genes in the green module are astrocyte genes, you would type this in R:

```
enrichments$ovGenes$'green -- cellType_Astrocyte_Cahoy_all'
# The output is...
# [1] "ACSBG1"          "ADD3"           "ADHFE1"         "AGL"            "AGT"
# [2] "AGTRL1"         "AGXT2L1"       "AHCYL1"         "AK3"            "ALDH1A1"
# (etc.)
```

Here are a few other websites that can be used to annotate modules given a list of genes:

- ChiliBot: <http://www.chilibot.net/>
- WebGestalt: <http://bioinfo.vanderbilt.edu/webgestalt/>
- Ingenuity: <http://www.ingenuity.com/>
- Galaxy: <http://main.g2.bx.psu.edu/>
- GSEA: <http://www.broadinstitute.org/gsea/index.jsp>
- UGET: <http://genome.ucla.edu/projects/UGET>
- STRING: <http://string.embl.de/>

Step 7: Using phenotypic information to determine differentially expressed genes and modules

In addition to using outside sources to annotate modules, phenotypic information of some kind is nearly always available for inclusion in module analysis and comparison (as well as for assessing differential expression). For example, in an experiment assessing the effects of drug treatment, samples will, at the very least, be assigned as a treatment or a control group. Likewise, in an experiment assessing changes between control and disease, phenotypic information regarding disease state will be provided. Furthermore, basic information such as age, gender, post-mortem interval, and brain region are also often provided. To demonstrate how one could perform such analysis, we first need to have phenotypic information:

```
region = rep("CA1",32);
region[c(1,4,6,11,12,15,16,17,22,24,25,26,28,29,31,32)] = "CA3"
age = c(81,72,86,90,88,90,90,74,83,73,73,70,85,85,75,90,72,70,90,84,75,85,80,
      86,85,84,81,88,80,90,83,74)
```

Now we will **find the genes that show the highest differential expression** with respect to both age and brain region in data set A1:

```
# Find the top 3 region genes
var = list(region=="CA1", region=="CA3")
datReg = t(apply(datExprAlg,1,t.test.1))
colnames(datReg)=c("MeanCA1","MeanCA3","SD_CA1","SD_CA3","PvalRegion")
datReg[order(datReg[,5])[1:3],]
#           MeanCA1      MeanCA3      SD_CA1      SD_CA3      PvalRegion
# NRIP3      10509.8842  22481.7087  2951.5061  6239.7370  6.739892e-07
# SAT1       4656.6850   2964.6736   885.3605   658.5044  1.335158e-06
# KCNH3      3814.7772   1840.0098  1049.7946   749.2281  1.493591e-06

# Find the top 3 age genes
var = age
datAge = t(apply(datExprAlg,1,cor.test.1))
colnames(datAge)=c("CorrAge","PvalAge")
datAge[order(datAge[,2])[1:3],]
#           CorrAge      PvalAge
# DDX42      0.7116889  4.947339e-06
# SCD        0.6961764  9.660017e-06
# KIAA0907  -0.6818583  1.728760e-05
```

We can use an analogous strategy to **find all of the modules that are either region or age related**, using the module eigengene as a proxy for expression levels genes in each module:

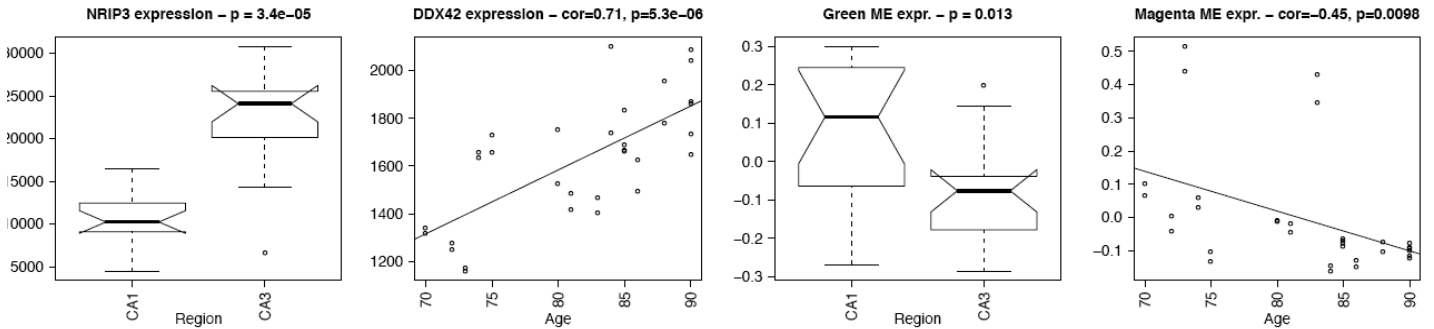
```
# Find the region-related modules
var = list(region=="CA1", region=="CA3")
datRegM = t(apply(t(ME_1A),1,t.test.1))
colnames(datRegM)=c("MeanCA1","MeanCA3","SD_CA1","SD_CA3","PvalRegion")
datRegM[datRegM[,5]<0.02,]
#           MeanCA1      MeanCA3      SD_CA1      SD_CA3      PvalRegion
# MEgreen    0.07875933 -0.07875933  0.1869713  0.1359238  0.011049496
# MEpink    -0.08035560  0.08035560  0.1241285  0.1936073  0.009706674

# Find the age-related modules
var = age
datAgeM = t(apply(t(ME_1A),1,cor.test.1))
colnames(datAgeM)=c("CorrAge","PvalAge")
datAgeM[datAgeM[,2]<0.02,]
#           CorrAge      PvalAge
# MEblack    0.4485706  0.010023898
# MEmagenta -0.4502347  0.009717422
# MEdred     -0.4303339  0.013952662
```

Overall, we find a few modules that show relationships to region and age: green has higher expression in CA1, pink has higher expression in CA3, black show increased expression with age, and magenta and red show decreased expression with age at $p < 0.02$. Note that one would normally account for multiple comparisons in these analyses—in this case after doing so, none of the modules would be significantly related to region or age.

We can now visualize these results using functions in the WGCNA library that make.

```
pdf("RegionAgePlots.pdf",width=16,height=4)
par(mfrow=c(1,4))
verboseBoxplot(as.numeric(datExprAlg["NRIP3",]), region,
  main="NRIP3 expression -", las=2, xlab="Region", ylab="")
verboseScatterplot(age, as.numeric(datExprAlg["DDX42",]),
  main="DDX42 expression -", las=2, abline=TRUE, xlab="Age", ylab="")
verboseBoxplot(as.numeric(ME_1A["MEgreen"]), region,
  main="Green ME expr. -", las=2, xlab="Region", ylab="")
verboseScatterplot(age, as.numeric(ME_1A["MEMagenta"]),
  main="Magenta ME expr. -", las=2, abline=TRUE, xlab="Age", ylab="")
dev.off()
```

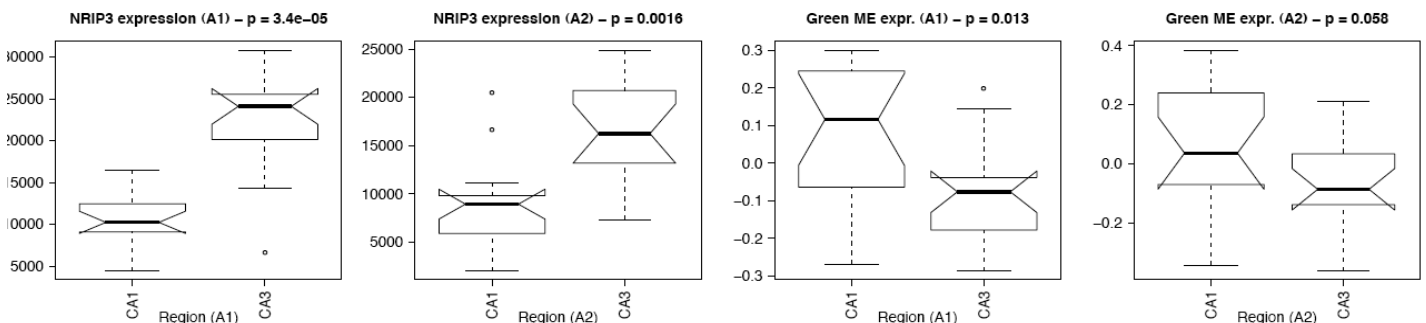


These functions (as well as verboseBarplot) are fairly dynamic, and can lead to publication-quality images if you spend some time adjusting the input parameters.

We can visually compare how a gene or module relates to phenotype across data sets using these plots as well. To do that, we need info for the same trait in both data sets and then we just draw the comparable graphs and compare by eye.

```
region2 = rep("CA1", 31);
region2[c(1,4,7,8,9,11,15,18,20,21,22,26,28,29,30)] = "CA3"
var = list(region2=="CA1", region2=="CA3")
datReg2 = t(apply(datExprA2g,1,t.test.1))
colnames(datReg2)=c("MeanCA1", "MeanCA3", "SD_CA1", "SD_CA3", "PvalRegion")
datRegM2 = t(apply(t(ME_2A),1,t.test.1))
colnames(datRegM2)=c("MeanCA1", "MeanCA3", "SD_CA1", "SD_CA3", "PvalRegion")
```

```
pdf("RegionAgePlots12.pdf",width=16,height=4)
par(mfrow=c(1,4))
verboseBoxplot(as.numeric(datExprAlg["NRIP3",]), region,
  main="NRIP3 expression (A1) -", las=2, xlab="Region (A1)", ylab="")
verboseBoxplot(as.numeric(datExprA2g["NRIP3",]), region2,
  main="NRIP3 expression (A2) -", las=2, xlab="Region (A2)", ylab="")
verboseBoxplot(as.numeric(ME_1A["MEgreen"]), region,
  main="Green ME expr. (A1) -", las=2, xlab="Region (A1)", ylab="")
verboseBoxplot(as.numeric(ME_2A["MEgreen"]), region2,
  main="Green ME expr. (A2) -", las=2, xlab="Region (A2)", ylab="")
dev.off()
```



In this case we find that both NRIP3 and the green module share similar trends in A1 and A2 with respect to region, but that the results are more significant in data set A1.

Step 8: Comparing networks with different module definitions

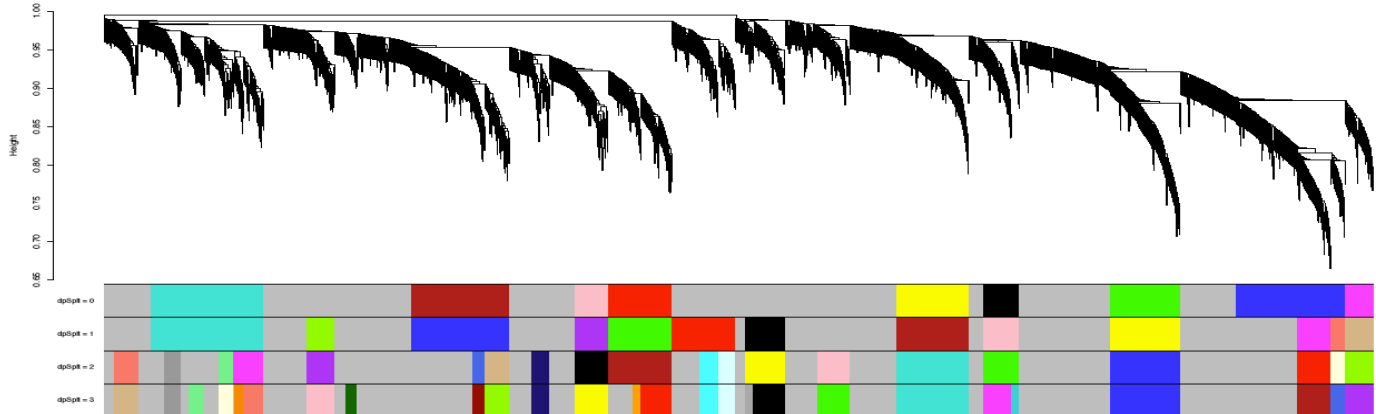
Thus far in the meta-analysis we have compared networks in which the module identifiers for genes from one network are assigned to be module identifiers for the same genes in a second network. Another common occurrence is to have two networks with different sets of module identifiers. In that situation, how would one go about comparing modules?

To address this issue, we will first **make a network using data set B1** (from step 1), taking only the subset of genes that were used in the network for A1 above. To do this we use basically the same code as above.

```
# (This section will take 5-15 minutes to run)
datExprB2g = (collapseRows(datExprB2,genesA,probesA))[[1]]
GeneAB = sort(intersect(rownames(datExprB2g),Gene)) # There are 3396 genes in this network

datExprB2g = datExprB2g[GeneAB,]
adjacencyB2 = adjacency(t(datExprB2g),power=softPower,type="signed");
diag(adjacencyB2)=0
dissTOMB2 = 1-TOMsimilarity(adjacencyB2, TOMtype="signed")
geneTreeB2 = flashClust(as.dist(dissTOMB2), method="average")

mColorh=NULL
for (ds in 0:3){
  tree = cutreeHybrid(dendro = geneTreeB2, pamStage=FALSE, minClusterSize = (30-3*ds),
    cutHeight = 0.99, deepSplit = ds, distM = dissTOMB2)
  mColorh=cbind(mColorh,labels2colors(tree$labels));
}
pdf("Module_choices_B2.pdf", height=10,width=25);
plotDendroAndColors(geneTreeB2, mColorh, paste("dpSplt =",0:3), main = "",dendroLabels=FALSE);
dev.off()
modulesB2 = mColorh[,1] # (Deepsplit=0 Chosen based on plot below)
colorsB2 = names(table(modulesB2))
```

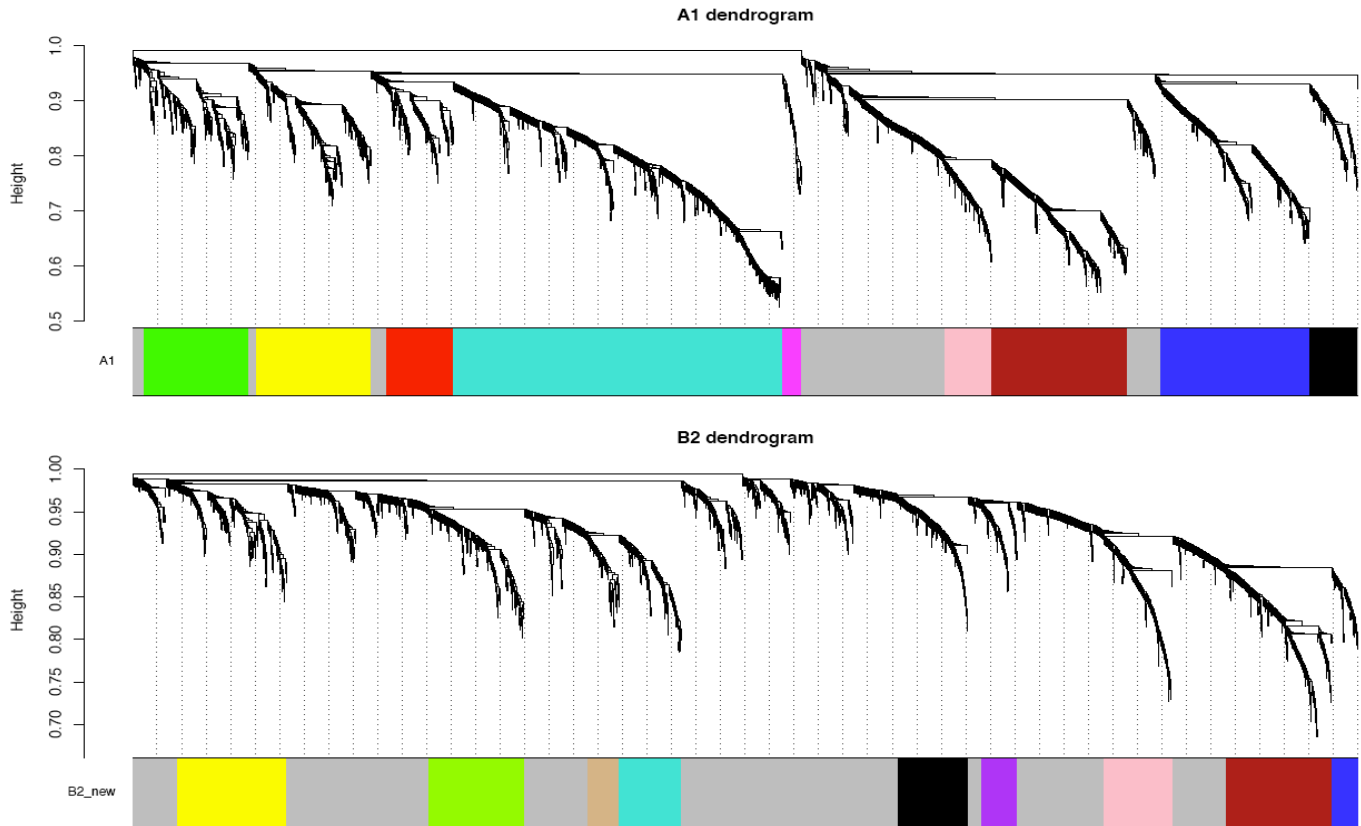


We can now check module overlap: once we have our module definition, we can check to see which modules in network A1 contain a significant number of overlapping genes with modules in B2. These would then be modules in the two networks that correspond to one another, and the labels from one network could be remapped such that corresponding modules are assigned the same color.

```
modulesB2_new = matchModules(Gene, modulesA1, GeneAB, modulesB2)
# Old - New labels: black - purple # Your screen output will look like this
# Old - New labels: blue - brown
# Old - New labels: brown - greenyellow # etc.
enrichmentsB2A1 = userListEnrichment(GeneAB,modulesB2,"kMETable1.csv","A1","enrichmentB2_A1.csv")
enrichmentsB2A1$sigOverlaps # To show the significant overlaps on the screen
# InputCategories UserDefinedCategories CorrectedPvalues
# pOut "blue" "A1_brown" "2.41636641189253e-59"
# pOut "turquoise" "A1_yellow" "1.51940862185272e-43"
# pOut "green" "A1_brown" "3.67833377321179e-41" # etc.
```

Overall, there are a lot of significantly overlapping modules between the two networks. This can be visualized by plotting the two dendrograms at once, using the following code:

```
pdf("A1_B2_dendrogram_plots.pdf",height=5,width=15)
plotDendroAndColors(geneTreeA1, modulesA1, "A1", dendroLabels=FALSE, hang=0.03,
  addGuide=TRUE, guideHang=0.05, main="A1 dendrogram")
plotDendroAndColors(geneTreeB2, modulesB2_new, "B2_new", dendroLabels=FALSE, hang=0.03,
  addGuide=TRUE, guideHang=0.05, main="B2 dendrogram")
dev.off()
```



Most of the modules in B2 match up to at least one module in A1 (ie., yellow, black, blue, etc.), but there are a few network-specific modules (ie, red in A1 and tan in B2). Depending on the purpose of the analysis, both modules in common, and modules not in common between networks could be interesting.

Comparing module annotations is another way to determine module agreement between networks. This can be done by performing nearly any of the annotation strategies in steps 6 and 7 for both A1 and B2, and then looking to see which modules have shared annotation. For example, if the red module in A1 and the tan module in B2 both have an over-representation for mitochondrial genes, one may want to consider these modules as overlapping, even if they don't share a significant number of genes in common. Likewise, if the yellow modules in A1 and B2 both are enriched for astrocyte genes, that provides further evidence that these genes have common function.

Concluding remarks

This is only a subset (but a very large subset!) of the things that you can do to compare data from multiple sources, regardless of the array platform of the original data. While in most ways microarray analysis is a science, in many other ways it is an art, requiring patience, approximation, and biological know-how. In the mean time, good luck, and if anything obvious is missing from this meta-analysis, please let me know.

Further reading

The collapseRows function:

Miller JA, Langfelder P, Cai C, Horvath S (2011) Strategies for optimally aggregating gene expression data: The collapseRows R function. Technical Report. (Submitted)

WGCNA:

- a) Langfelder P, Horvath S (2008) WGCNA: an R package for Weighted Correlation Network Analysis. BMC Bioinformatics. 2008 Dec 29;9(1):559.
- b) Horvath S, Zhang B, Carlson M, Lu KV, Zhu S, Felciano RM, Laurance MF, Zhao W, Shu, Q, Lee Y, Scheck AC, Liao LM, Wu H, Geschwind DH, Febbo PG, Kornblum HI, Cloughesy TF, Nelson SF, Mischel PS (2006) Analysis of Oncogenic Signaling Networks in Glioblastoma Identifies ASPM as a Novel Molecular Target. Natl Acad Sci U S A. 2006 Nov 14;103(46):17402-17407
- c) Zhang B, Horvath S (2005) A General Framework for Weighted Gene Co-Expression Network Analysis. Statistical Applications in Genetics and Molecular Biology. Vol. 4: No. 1, Article 17.

Brain data applications of WGCNA the following and more:

- a) Miller JA, Horvath S, Geschwind DH. (2010) Divergence of human and mouse brain transcriptome highlights Alzheimer disease pathways. Proc Natl Acad Sci U S A. 2010 Jul 13;107(28):12698-703.
- b) Winden KD, Oldham MC, Mirnics K, Ebert PJ, Swan CH, Levitt P, Rubenstein JL, Horvath S, Geschwind DH. (2009) The organization of the transcriptional network in specific neuronal classes. Mol Syst Biol. 2009;5:291.
- c) Oldham M, Konopka G, Iwamoto K, Langfelder P, Kato T, Horvath S, Geschwind D (2008) Functional organization of the transcriptome in human brain. Nature Neuroscience 11:1271-1282.
- d) Oldham M, Horvath S, Geschwind D (2006) Conservation and Evolution of Gene Co-expression Networks in Human and Chimpanzee Brains. PNAS. 2006 Nov 21;103(47):17973-8

Preservation statistics:

Langfelder P, Luo R, Oldham MC, Horvath S (2011) Is my network module preserved and reproducible? PloS Comp Biol. 7(1): e1001057 PMID: 21283776

Dynamic tree cutting:

Langfelder P, Zhang B, Horvath S (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. Bioinformatics. Bioinformatics. 2008 Mar 1;24(5):719-20.

Eigengene:

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-expression modules. BMC Systems Biology. BMC Syst Biol. 2007 Nov 21;1(1):54

kME and module membership:

Horvath S, Dong J (2008) Geometric Interpretation of Gene Co-Expression Network Analysis. PloS Computational Biology. 4(8): e1000117