Tutorial for the WGCNA package in R: Consensus network, GO analysis, cell type enrichment, module network plots

Elisabeth Rebboah

December 2018

Part 1: Data Input, cleaning, and pre-processing

Set working directory and load libraries necessary for WGCNA and other steps. enableWGCNAThreads allows parallel execution with multiple cores.

```
setwd("~/WGCNA")
getwd()
library(WGCNA)
library(flashClust)
library(gplots)
library(cluster)
library(igraph); #for part 8
library(RColorBrewer); #for part 8
options(stringsAsFactors=FALSE)
#Only when not on server, comment out:
enableWGCNAThreads()
```

Expression matrix datExpr needs to have genes in columns and samples in rows, i.e. final datExpr might be something like 20,000 genes by 200 patients. Also must have Ensembl annotation for list of genes. Expression values are from microarray or RNA-seq, already pre-processed and normalized. Target matrices contain samples in rows and traits in columns, such as diagnosis, age, sex, etc.

For a consensus network, the genes between datasets must overlap. For example, for three datasets, datExpr.Ref.DS, datExpr.Ref.AD_1, and datExpr.Ref.AD_2 :

```
gnS=intersect(colnames(datExpr.Ref.DS),intersect(colnames(datExpr.Ref.AD_1),colnames(datExpr
    .Ref.AD_2)))
datExpr.Ref.DS=datExpr.Ref.DS[,match(gnS,colnames(datExpr.Ref.DS))]
datExpr.Ref.AD_1 =datExpr.Ref.AD_1 [,match(gnS,colnames(datExpr.Ref.AD_1))]
datExpr.Ref.AD_2 =datExpr.Ref.AD_2 [,match(gnS,colnames(datExpr.Ref.AD_2))]
```

From two or more expression matrices, combine into a multiExpr dataframe, a vector of lists, in each list there must be a component named "data" whose content is the individual expression matrices.

```
#Make multiExpr list of 3 data frames
nSets=3;
setLabels=c("Downs syndrome samples", "Alzheimer's samples 1", "Alzheimer's samples 2")
#Form multi-set expression data
multiExpr = vector(mode="list", length=nSets)

#Remove rownames and column names from datExpr dfs, re-assign in multiExpr
genes = colnames(datExpr.Ref.DS);
samples = rownames(datExpr.Ref.DS);
rownames(datExpr.Ref.DS)=NULL;
colnames(datExpr.Ref.DS)=NULL;
multiExpr[[1]] = list(data = datExpr.Ref.DS);
colnames(multiExpr[[1]]$data) = genes;
rownames(multiExpr[[1]]$data) = samples;

genes = colnames(datExpr.Ref.AD_1);
```

```
samples = rownames(datExpr.Ref.AD_1);
rownames(datExpr.Ref.AD_1)=NULL;
colnames(datExpr.Ref.AD_1)=NULL;
multiExpr[[2]] = list(data = datExpr.Ref.AD_1);
names(multiExpr[[2]]$data) = genes;
rownames(multiExpr[[2]]$data) = samples;

genes = colnames(datExpr.Ref.AD_2);
samples = rownames(datExpr.Ref.AD_2);
rownames(datExpr.Ref.AD_2)=NULL;
colnames(datExpr.Ref.AD_2)=NULL;
multiExpr[[3]] = list(data = datExpr.Ref.AD_2);
colnames(multiExpr[[3]]$data) = genes;
rownames(multiExpr[[3]]$data) = samples;

exprSize = checkSets(multiExpr)
exprSize #Check data is in correct format; checkSets function checks whether given sets have
     the correct format and retrieves dimensions.

#$nGenes
#[1] 11810
#$nSamples
#[1] 110 104 465
```

Also make list of lists for all traits in multiMeta object, and check that all genes and samples don't have too many missing values (gsg$allOK should return TRUE if all genes and samples are okay).

```
#Make multiMeta of traits for all sets - list of lists
multiMeta=list(aging = list(data=targets.Ref.DS), AD1 = list(data=targets.Ref.AD_1), AD2 =
    list(data=targets.Ref.AD_2))

#Check that all genes and samples have sufficiently low numbers of missing values
gsg = goodSamplesGenesMS(multiExpr, verbose = 3);
gsg$allOK
```

Save input data in rda file.

```
#Save everything
nGenes=exprSize$nGenes;
nSamples=exprSize$nSamples;
save(multiExpr, multiMeta, nGenes, nSamples, setLabels,
     targets.Ref.AD_1, targets.Ref.AD_2, targets.Ref.DS,
     datExpr.Ref.AD_1, datExpr.Ref.AD_2, datExpr.Ref.DS, file = "Consensus_dataInput_112418.
         rda");
```

Part 2: Choose soft thresholding power

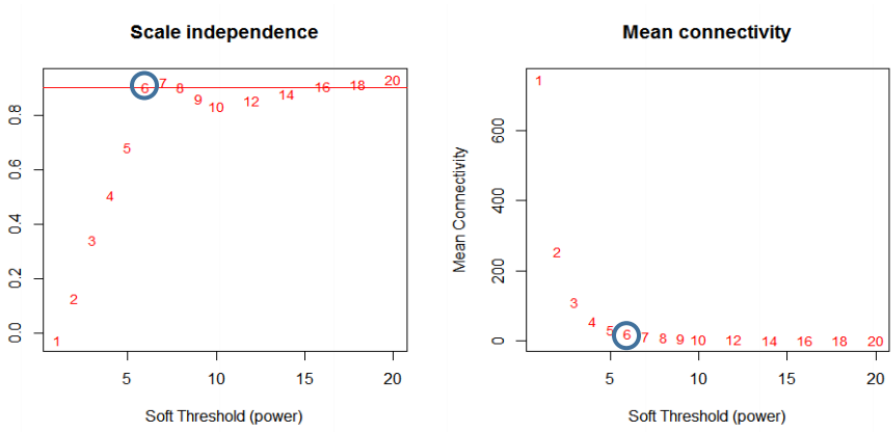
Next step is to set a power (β) for a scale-free topology network, which has to be done by the user based on plots of scale-free topology fit and network connectivity. It is best to choose a power that crosses some threshold in the scale-free topology fit plot (such as > 0.8) but does not lose too much connectivity.

Input a set of soft thresholding powers to test and call pickSoftThreshold on data for the type of network you want to build with the type of correlation function to be used in adjacency calculation; in this case it is signed with a bicor correlation function.

```
# Choose a set of soft-thresholding powers
```

```
powers = c(seq(1,10,by=1), seq(12,30, by=2));
# Initialize a list to hold the results of scale-free analysis
powerTables = vector(mode = "list", length = nSets);
# Call the network topology analysis function for each set in turn
for (set in 1:nSets)
  powerTables[[set]] = list(data = pickSoftThreshold(multiExpr[[set]]$data, powerVector=
      powers,
                                                     verbose = 5, networkType="signed",
                                                         corFnc="bicor")[[2]]);
```

Example plots, where 6 is chosen since it crosses an arbitrary threshold (red line) but doesn't lose all the connectivity.



Part 3: Network Construction

From plots generated in part 2, set the soft thresholding power. Next, run blockwiseConsensusModules to generate network across datasets. This function has many parameters than can be adjusted. Again, in this case networkType is signed. Consult function documentation for more information on parameters.

```
#Set soft thresholding power to number based on plots
softpower=16;
# Auto network
load(file = "Consensus_dataInput_112418.rda");
net=blockwiseConsensusModules(multiExpr, blocks = NULL,
                              maxBlockSize = 30000, ## This should be set to a smaller size
                                  if the user has limited RAM
                              randomSeed = 12345,
                              corType = "pearson", ## no use for bicor
                              power = softpower,
                              consensusQuantile = 0.2,
                              networkType = "signed",
                              TOMType = "unsigned",
                              TOMDenom = "min",
                              scaleTOMs = TRUE, scaleQuantile = 0,
                              sampleForScaling = TRUE, sampleForScalingFactor = 1000,
                              useDiskCache = TRUE, chunkSize = NULL,
                              deepSplit = 2,
                              detectCutHeight = 0.99999999, minModuleSize = 20,
                              mergeCutHeight = 0.2,
                              saveConsensusTOMs = TRUE,
                              consensusTOMFilePattern = "ConsensusTOM-block.%b.rda")
save(list=ls(),file="consensus_112418.rda")
```

Get module eigengenes (MEs) from "net" dataframe. To cluster the network into a dendrogram, use hclust on the dissimilarity matrix, 1-TOM.

```
consMEs = net$multiMEs;
moduleColors = net$colors;
table(moduleColors)

load("ConsensusTOM-block.1.rda") # consensus TOM
consTree= hclust(1-consTomDS,method="average");
save(list=ls(),file="consensus_112418.rda")
```

To identify modules, cutreeHybrid detects clusters in the dendrogram based on a set of parameters, some of which are looped through so that the user can determine which set of modules makes the most sense. As a personal preference, I also plotted how each gene correlates to all traits to see any obvious trends between modules and traits. Trait data has to be converted to numbers, then each gene is correlated to the trait. For example, diagnosis is 0 or 1, control or diseased, and genes could be positively or negatively correlated to having the disease.

Examples of converting traits to binaries, followed by calculating biweight midcorrelation between each gene across all samples and each trait value as a number across all samples, so between two numeric vectors. Bicor is median-based rather than mean-based, therefore it is more robust to outliers than Pearson correlation.

```
Diagnosis=factor(as.character(targets.Ref.AD_2$Disease))
Diagnosis<-relevel(Diagnosis,'Control')
Diagnosis=as.numeric(Diagnosis)
Age=as.numeric(targets.Ref.AD_2$Age)
GEO=as.numeric(factor(targets.Ref.AD_2$geo_accession))
Gender=as.numeric(factor(targets.Ref.AD_2$Sex))

geneSigsAD2=matrix(NA,nrow=4,ncol=ncol(datExpr.Ref.AD_2))
for(i in 1:ncol(geneSigsAD2)) {
  exprvec=as.numeric(datExpr.Ref.AD_2[,i])
  ager=bicor(Age,exprvec,use="pairwise.complete.obs")
  sexr=bicor(exprvec, Gender,use="pairwise.complete.obs")
  conditionr=bicor(exprvec, Diagnosis,use="pairwise.complete.obs")
  geor=bicor(GEO,exprvec,use="pairwise.complete.obs")
  geneSigsAD2[,i]=c(ager, sexr,conditionr,geor)
  cat('Done for gene...',i,'\n')
}
```

Finding different sets of modules based on different values for parameters mms, dthresh, and ds using cutreeHybrid, which are then plotted:

```
# Calculate modules for each set of parameters
load("consensus_112418.rda")
mColorh <- mLabelh <- colorLabels <- NULL
for (minModSize in c(40,100,160)) {
  for (dthresh in c(0.1,0.2,0.25)) {
    for (ds in c(2,4)) {
      print("Trying parameters:")
      print(c(minModSize,dthresh,ds))
      tree = cutreeHybrid(dendro = consTree, pamStage=FALSE,
                          minClusterSize = minModSize, cutHeight = 0.99999999,
                          deepSplit = ds, distM = as.matrix(1-consTomDS))

      merged <- mergeCloseModules(exprData = multiExpr,colors = tree$labels,
                                  cutHeight = dthresh)
      mColorh <- cbind(mColorh,labels2colors(merged$colors))
      mLabelh <- c(mLabelh,paste("DS=",ds," mms=\n",minModSize," dcor=",dthresh))
    }
  }
```

```
}
```

Once mms, ds, and dthresh are chosen based on best fit of clusters, run cutreeHybrid again and merge chosen modules so that genes are associated with a given module. These genes are summarized by a "module eigengene" that captures the most variance in the module expression data, can be thought of as the first principal component of the module.

```
mms=160
ds=4
dthresh=0.1

tree = cutreeHybrid(dendro = consTree, pamStage=FALSE,
                    minClusterSize = mms, cutHeight = 0.99999999,
                    deepSplit = ds, distM = as.matrix(1-consTomDS))

merged <- mergeCloseModules(exprData = multiExpr,colors = tree$labels,
                            cutHeight = dthresh)

# Eigengenes of the new merged modules:
MEs_DS = merged$newMEs[[1]]$data;
MEs_AD1 = merged$newMEs[[2]]$data;
MEs_AD2 = merged$newMEs[[3]]$data;
```

Next, connectivity of each gene across datasets to the module eigengene is determined by the consensusKME function, then consensus kMEs are determined.

```
#Calculate kMEs and p value, Z score of kMEs for each gene (connectivity of each gene to a
    module eigengene) across datasets
consKME1=consensusKME(multiExpr=multiExpr, moduleColor.cons,
                      multiEigengenes = NULL,
                      consensusQuantile = 0.2,
                      signed = TRUE)
#Consensus kMEs:
consensus.KMEs=consKME1[,regexpr('consensus.kME',names(consKME1))>0]
```

Part 4: Get annotated gene lists and their associated kMEs

Convert Ensemble gene IDs to gene symbols, output Excel file of list of genes and their associated kMEs. This can be used as an input file for further analyses.

```
load("Consensus_MEs_112418.rda")
ensembl=read.csv("/home/vivek/FTD_Seeley/Analysis_Nov2017/ENSG85_Human.csv.gz")
consensus.KMEs$Ensembl.Gene.ID=paste(rownames(consensus.KMEs))

merged=merge(consensus.KMEs,ensembl,by.x="Ensembl.Gene.ID",by.y="Ensembl.Gene.ID",all.x=T)
ind=match(consensus.KMEs$Ensembl.Gene.ID,merged$Ensembl.Gene.ID)
merged1=merged[ind,]
consensus.KMEs.annot=merged1

geneInfo.cons=as.data.frame(cbind(consensus.KMEs.annot$Ensembl.Gene.ID,consensus.KMEs.annot$
    Associated.Gene.Name,
                                  moduleColor.cons,consensus.KMEs))
geneInfo.cons=geneInfo.cons[,-ncol(geneInfo.cons)] # check if last column is Ensembl gene id

colnames(geneInfo.cons)[1]= "Ensembl.Gene.ID"
colnames(geneInfo.cons)[2]= "GeneSymbol"
colnames(geneInfo.cons)[3]= "Initially.Assigned.Module.Color"

write.csv(geneInfo.cons,'geneInfo.cons.DSAD_112418.csv')
save(list=ls(),file='geneInfo.cons.112418.rda')
```

Part 5: Module-trait relationships

Use correlations to make a heatmap of each module eigengene across traits of interest, display significant p-values and correlations in heatmap boxes. These values can be saved to Excel sheets later.

```
pdf(paste('NetworkPlot_AD1_consensus_112418.pdf'),width=16,height=30)
par( mar = c(8, 12, 3, 3) );
labeledHeatmap( Matrix = moduleTraitCor_AD1,
                xLabels = colnames(factors1_AD1),
                yLabels = rownames(moduleTraitPvalue_AD1),
                ySymbols = rownames(moduleTraitPvalue_AD1),
                colorLabels = FALSE,
                colors = blueWhiteRed(50),
                textMatrix = textMatrix1,
                setStdMargins = FALSE,
                cex.text = 1.5,
                zlim = c(-1, 1),
                cex.lab.x = 1.2,
                main = paste("Module-trait relationships")
);
```

This code plots eigengene network in the form of a dendrogram and heatmap. This shows how related the eigengenes are to each other. Each row and column corresponds to an eigengene (labeled by consensus module color). Red indicates high adjacency (positive correlation) and white no adjacency (no correlation).

```
#Plot eigengene heatmap
par(cex = 1.0)
plotEigengeneNetworks(MEs_DS, "Eigengene Network", marHeatmap = c(3,4,2,2), marDendro = c
    (0,4,1,2),cex.adjacency = 0.3,plotDendrograms = TRUE, xLabelsAngle = 90,heatmapColors=
    blueWhiteRed(100)[51:100])
```

Also make boxplots and scatterplots of trait enrichment for each module. For the binary factor traits (male/female, diseased/control) boxplots indicate enrichment. For the numeric traits such as age, a scatterplot shows if there is enrichment for a specific age and plots a trendline.

```
#Plot boxplots, scatterplots
toplot=t(MEs_AD2)
cols=substring(colnames(MEs_AD2),3,20)
par(mfrow=c(4,4))
par(mar=c(5,6,4,2))
for (i in 1:nrow(toplot)) {
  boxplot(toplot[i,]~factor(as.vector(as.factor(targets.Ref.AD_2$Disease)),c('Control','AD')
      ),col=cols[i],ylab="ME",main=rownames(toplot)[i],xlab=NULL,las=2)
  verboseScatterplot(x=as.numeric(targets.Ref.AD_2$Age),y=toplot[i,],xlab="Age",ylab="ME",
      abline=TRUE,cex.axis=1,cex.lab=1,cex=1,col=cols[i],pch=19)
  boxplot(toplot[i,]~factor(targets.Ref.AD_2$Sex),col=cols[i],ylab="ME",main=rownames(toplot
      )[i],xlab=NULL,las=2)
}

dev.off()
```

Part 6: GO Analysis

Finally, find GO enrichment terms for each module. This is the most interesting part of the analysis since you can start to make biological connections to the modules.

First, create directories for inputs and outputs.

```
dir.create("./geneInfo")
dir.create("./geneInfo/background/")
dir.create("./geneInfo/input/")
dir.create("./geneInfo/output/")
```

After adding some reference files into these directors, analysis is run as a nohupped bash shell script that uses inputs and outputs from these new directories.

```
# Run GO elite as nohupped shell script:
codedir <- "/home/vivek/bin/GO-Elite_v.1.2.5-Py"
pathname <- "/home/erebboah/WGCNA/geneInfo"
nperm=10000
system(paste("nohup python ",codedir,"/GO_Elite.py --species Hs --mod Ensembl --permutations ",
             nperm,"  --method \"z-score\" --zscore 1.96 --pval 0.01 --num 5 --input ",pathname,
             "/input --denom ",pathname,"/background --output ",pathname,"/output &",sep="")
             )
```

In this way, code can be run in the background for several hours. It should only take a few hours or so for 16 modules (depending on available CPU). Plot the results:

```
pdf("GOElite_plot_Modules_112718.pdf",height=8,width=12)
for(i in 1:length(uniquemodcolors)){
  thismod= uniquemodcolors[i]
  tmp=read.csv(file=paste(pathname,"/ORA_pruned/",thismod,"_Module-GO_z-score_elite.txt",sep=""),sep="\t")
  tmp=subset(tmp,Ontology.Type!='cellular_component')
  tmp=tmp[,c(2,9)] ## Select GO-terms and Z-score
  tmp=tmp[order(tmp$Z.Score,decreasing=T),] #
  if (nrow(tmp)<10){
    tmp1=tmp ## Take top 10 Z-score
    tmp1 = tmp1[order(tmp1$Z.Score),] ##Re-arrange by increasing Z-score
    par(mar=c(5,40,5,2))
    barplot(tmp1$Z.Score,horiz=T,col="blue",names.arg= tmp1$Ontology.Name,cex.names=1.2,las=1,main=paste("Gene Ontology Plot of",thismod,"Module"),xlab="Z-Score")
    abline(v=2,col="red")
  } else {
    tmp1=tmp[c(1:10),] ## Take top 10 Z-score
    tmp1 = tmp1[order(tmp1$Z.Score),] ##Re-arrange by increasing Z-score
    par(mar=c(5,40,5,2))
    barplot(tmp1$Z.Score,horiz=T,col="blue",names.arg= tmp1$Ontology.Name,cex.names=1.2,las=1,main=paste("Gene Ontology Plot of",thismod,"Module"),xlab="Z-Score")
    abline(v=2,col="red")
    }

  cat('Done ...',thismod,'\n')
}

dev.off()
```

Part 7: Cell type enrichment

Make heatmap plus p-values and correlation scores across modules for these cell types: neurons, microglia, myelinating oligodendrocytes, astrocytes, and endothelial cells. Also useful for making biological insights into modules.

```
pdf("CellTypeEnrich_WGCNAMods_110618.pdf", width=6,height=10)
labeledHeatmap(Matrix=dispMat,
               yLabels=rownames(dispMat),
               yColorLabels=TRUE,
               xLabels= colnames(dispMat),
               colors=blueWhiteRed(40),
               textMatrix = textMatrix1,
               cex.lab.x=1.0,
               zlim=c(-0.1,3),
               main="Cell-type enrichment Heatmap")
dev.off()
```

Part 8: TOM network plot

Will make node and edge plots with hub genes in the center surrounded by all other genes in each module. Also somewhat useful for biological insights, hub genes can serve as sanity checks that the WGCNA actually finds relevant genes to the diseases being studied.

```
pdf("ModuleNetworks.pdf",height=9,width=10);
for (mod in uniquemodcolors)  {
  numgenesingraph = 100;
  numconnections2keep = 1500;
  cat('module:',mod,'\n');
  geneInfo.cons=geneInfo.cons[geneInfo.cons$GeneSymbol!="NA",]
  colind = which(colnames(geneInfo.cons)==paste("consensus.kME",mod, sep=""));
  rowind = which(geneInfo.cons[,3]==mod);
  cat(' ',length(rowind),'probes in module\n');
  submatrix = geneInfo.cons[rowind,];
  orderind = order(submatrix[,colind],decreasing=TRUE);
  if (length(rowind) < numgenesingraph) {
    numgenesingraph = length(rowind);
    numconnections2keep = numgenesingraph * (numgenesingraph - 1);
  }
  cat('Making network graphs, using top',numgenesingraph,'probes and',numconnections2keep,'
      connections of TOM\n');
  submatrix = submatrix[orderind[1:numgenesingraph],];
  #Identify the columns in the TOM that correspond to these hub probes
  matchind = match(submatrix$Ensembl.Gene.ID,colnames(multiExpr));
  reducedTOM = TOM.matrix[matchind,matchind];

  orderind = order(reducedTOM,decreasing=TRUE);
  connections2keep = orderind[1:numconnections2keep];
  reducedTOM = matrix(0,nrow(reducedTOM),ncol(reducedTOM));
  reducedTOM[connections2keep] = 1;

  g0 <- graph.adjacency(as.matrix(reducedTOM[1:10,1:10]),mode="undirected",weighted=TRUE,
      diag=FALSE)
  layoutMata <- layout.circle(g0)

  g0 <- graph.adjacency(as.matrix(reducedTOM[11:50,11:50]),mode="undirected",weighted=TRUE,
      diag=FALSE)
  layoutMatb <- layout.circle(g0)

  g0 <- graph.adjacency(as.matrix(reducedTOM[51:ncol(reducedTOM),51:ncol(reducedTOM)]),mode=
      "undirected",weighted=TRUE,diag=FALSE)
  layoutMatc <- layout.circle(g0)
  g1 <- graph.adjacency(as.matrix(reducedTOM),mode="undirected",weighted=TRUE,diag=FALSE)
  layoutMat <- rbind(layoutMata*0.25,layoutMatb*0.8, layoutMatc)

  plot(g1,edge.color="grey",vertex.color=mod,vertex.label=as.character(submatrix$GeneSymbol)
      ,vertex.label.cex=0.7,vertex.label.dist=0.45,vertex.label.degree=-pi/4,vertex.label.
```

```
        color="black",layout= layoutMat,vertex.size=submatrix[,colind]^2*8,main=paste(mod,"
        module"))
39
40 }
41 dev.off();
```