Tutorial for the WGCNA package for R:
I. Network analysis of liver expression data in female mice

1. Data input and cleaning

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

1 Data input, cleaning and pre-processing

This is the first step of any network analysis. We show here how to load typical expression data, pre-process them into a format suitable for network analysis, and clean the data by removing obvious outlier samples as well as genes and samples with excessive numbers of missing entries.

1.a Loading expression data

The expression data is contained in the file `LiverFemale3600.csv` that comes with this tutorial. After starting an R session, we load the requisite packages and the data, after appropriately setting the working directory:

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the WGCNA package
library(WGCNA);
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
#Read in the female liver data set
femData = read.csv("LiverFemale3600.csv");
# Take a quick look at what is in the data set:
dim(femData);
names(femData);
```

In addition to expression data, the data files contain extra information about the surveyed probes we do not need. One can inspect larger data frames such as `femData` by invoking R data editor via `fix(femData)`. The expression data set contains 135 samples. Note that each row corresponds to a gene and column to a sample or auxiliary information. We now remove the auxiliary data and transpose the expression data for further analysis.

```
datExpr0 = as.data.frame(t(femData[, -c(1:8)]));
names(datExpr0) = femData$substanceBXH;
rownames(datExpr0) = names(femData)[-c(1:8)];
```

1.b Checking data for excessive missing values and identification of outlier microarray samples

We first check for genes and samples with too many missing values:

```
gsg = goodSamplesGenes(datExpr0, verbose = 3);
gsg$allOK
```

If the last statement returns `TRUE`, all genes have passed the cuts. If not, we remove the offending genes and samples from the data:

```
if (!gsg$allOK)
{
  # Optionally, print the gene and sample names that were removed:
  if (sum(!gsg$goodGenes)>0)
     printFlush(paste("Removing genes:", paste(names(datExpr0)[!gsg$goodGenes], collapse = ", ")));
  if (sum(!gsg$goodSamples)>0)
     printFlush(paste("Removing samples:", paste(rownames(datExpr0)[!gsg$goodSamples], collapse = ", ")));
  # Remove the offending genes and samples from the data:
  datExpr0 = datExpr0[gsg$goodSamples, gsg$goodGenes]
}
```

Next we cluster the samples (in contrast to clustering genes that will come later) to see if there are any obvious outliers.

```
sampleTree = hclust(dist(datExpr0), method = "average");
# Plot the sample tree: Open a graphic output window of size 12 by 9 inches
# The user should change the dimensions if the window is too large or too small.
sizeGrWindow(12,9)
#pdf(file = "Plots/sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree, main = "Sample clustering to detect outliers", sub="", xlab="", cex.lab = 1.5,
     cex.axis = 1.5, cex.main = 2)
```

It appears there is one outlier (sample `F2_221`, see Fig. 1). One can remove it by hand, or use an automatic approach. Choose a height cut that will remove the offending sample, say 15 (the red line in the plot), and use a branch cut at that height.

```
# Plot a line to show the cut
abline(h = 15, col = "red");
# Determine cluster under the line
clust = cutreeStatic(sampleTree, cutHeight = 15, minSize = 10)
table(clust)
# clust 1 contains the samples we want to keep.
keepSamples = (clust==1)
datExpr = datExpr0[keepSamples, ]
nGenes = ncol(datExpr)
nSamples = nrow(datExpr)
```

The variable `datExpr` now contains the expression data ready for network analysis.

1.c Loading clinical trait data

We now read in the trait data and match the samples for which they were measured to the expression samples.

```
traitData = read.csv("ClinicalTraits.csv");
dim(traitData)
names(traitData)

# remove columns that hold information we do not need.
allTraits = traitData[, -c(31, 16)];
allTraits = allTraits[, c(2, 11:36) ];
dim(allTraits)
names(allTraits)

# Form a data frame analogous to expression data that will hold the clinical traits.

femaleSamples = rownames(datExpr);
traitRows = match(femaleSamples, allTraits$Mice);
datTraits = allTraits[traitRows, -1];
rownames(datTraits) = allTraits[traitRows, 1];

collectGarbage();
```

We now have the expression data in the variable `datExpr`, and the corresponding clinical traits in the variable `datTraits`. Before we continue with network construction and module detection, we visualize how the clinical traits relate to the sample dendrogram.

```
# Re-cluster samples
sampleTree2 = hclust(dist(datExpr), method = "average")
# Convert traits to a color representation: white means low, red means high, grey means missing entry
traitColors = numbers2colors(datTraits, signed = FALSE);
# Plot the sample dendrogram and the colors underneath.
plotDendroAndColors(sampleTree2, traitColors,
                    groupLabels = names(datTraits),
                    main = "Sample dendrogram and trait heatmap")
```

In the plot, shown in Fig. 2, white means a low value, red a high value, and grey a missing entry.
The last step is to save the relevant expression and trait data for use in the next steps of the tutorial.

```
save(datExpr, datTraits, file = "FemaleLiver-01-dataInput.RData")
```

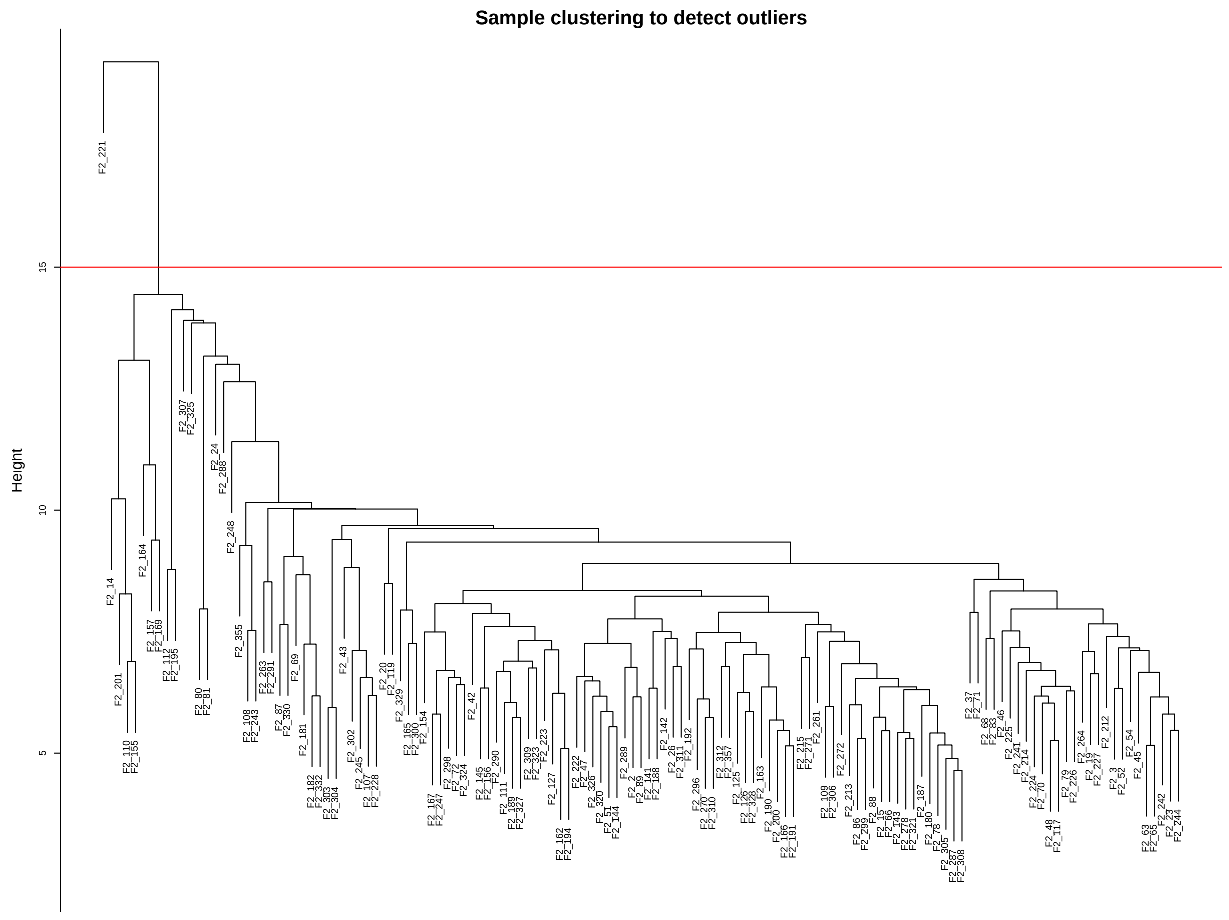

Figure 1: Clustering dendrogram of samples based on their Euclidean distance.

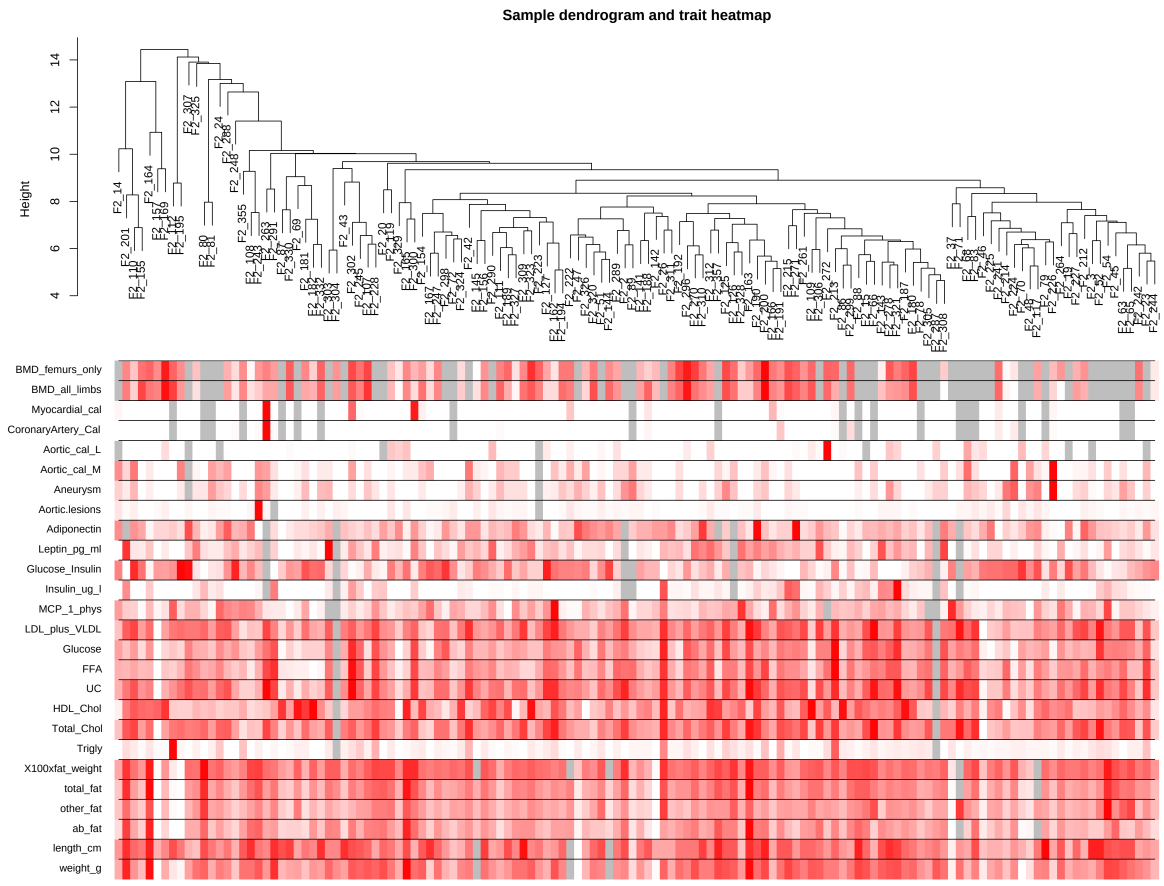

Figure 2: Clustering dendrogram of samples based on their Euclidean distance.

Tutorial for the WGCNA package for R:
I. Network analysis of liver expression data in female mice

2.a Automatic network construction and module detection

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

0 Preliminaries: setting up the R session

Here we assume that a new R session has just been started. We load the WGCNA package, set up basic parameters and load data saved in the first part of the tutorial.

Important note: The code below uses parallel computation where multiple cores are available. This works well when R is run from a terminal or from the Graphical User Interface (GUI) shipped with R itself, but at present it **does not work** with RStudio and possibly other third-party R environments. If you use RStudio or other third-party R environments, skip the `enableWGCNAThreads()` call below.

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the WGCNA package
library(WGCNA)
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Allow multi-threading within WGCNA. This helps speed up certain calculations.
# At present this call is necessary for the code to work.
# Any error here may be ignored but you may want to update WGCNA if you see one.
# Caution: skip this line if you run RStudio or other third-party R environments.
# See note above.
enableWGCNAThreads()
# Load the data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
```

We have loaded the variables `datExpr` and `datTraits` containing the expression and trait data, respectively.

2 Automatic construction of the gene network and identification of modules

This step is the bedrock of all network analyses using the WGCNA methodology. We present three different ways of constructing a network and identifying modules:

a. Using a convenient 1-step network construction and module detection function, suitable for users wishing to arrive at the result with minimum effort;

b. Step-by-step network construction and module detection for users who would like to experiment with customized/alternate methods;

c. An automatic block-wise network construction and module detection method for users who wish to analyze data sets too large to be analyzed all in one.

In this tutorial section, we illustrate the 1-step, automatic network construction and module detection.

2.a Automatic network construction and module detection

2.a.1 Choosing the soft-thresholding power: analysis of network topology

Constructing a weighted gene network entails the choice of the soft thresholding power β to which co-expression similarity is raised to calculate adjacency [1]. The authors of [1] have proposed to choose the soft thresholding power based on the criterion of approximate scale-free topology. We refer the reader to that work for more details; here we illustrate the use of the function `pickSoftThreshold` that performs the analysis of network topology and aids the user in choosing a proper soft-thresholding power. The user chooses a set of candidate powers (the function provides suitable default values), and the function returns a set of network indices that should be inspected, for example as follows:

```
# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to=20, by=2))
# Call the network topology analysis function
sft = pickSoftThreshold(datExpr, powerVector = powers, verbose = 5)
# Plot the results:
sizeGrWindow(9, 5)
par(mfrow = c(1,2));
cex1 = 0.9;
# Scale-free topology fit index as a function of the soft-thresholding power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fit,signed R^2",type="n",
     main = paste("Scale independence"));
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     labels=powers,cex=cex1,col="red");
# this line corresponds to using an R^2 cut-off of h
abline(h=0.90,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab="Soft Threshold (power)",ylab="Mean Connectivity", type="n",
     main = paste("Mean connectivity"))
text(sft$fitIndices[,1], sft$fitIndices[,5], labels=powers, cex=cex1,col="red")
```

The result is shown in Fig. 1. We choose the power 6, which is the lowest power for which the scale-free topology fit index curve flattens out upon reaching a high value (in this case, roughly 0.90).

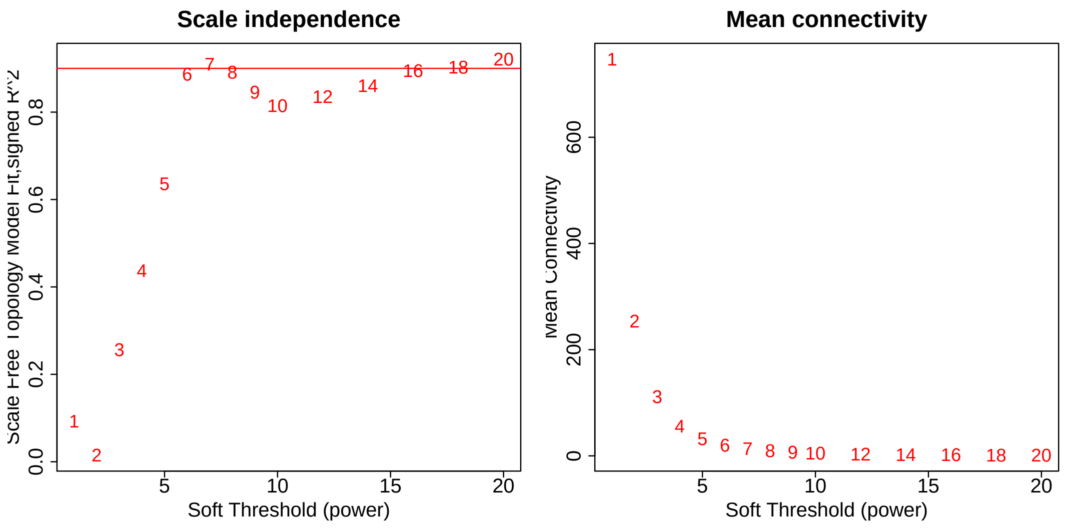

Figure 1: Analysis of network topology for various soft-thresholding powers. The left panel shows the scale-free fit index (y-axis) as a function of the soft-thresholding power (x-axis). The right panel displays the mean connectivity (degree, y-axis) as a function of the soft-thresholding power (x-axis).

2.a.2 One-step network construction and module detection

Constructing the gene network and identifying modules is now a simple function call:

```
net = blockwiseModules(datExpr, power = 6,
                       TOMType = "unsigned", minModuleSize = 30,
                       reassignThreshold = 0, mergeCutHeight = 0.25,
                       numericLabels = TRUE, pamRespectsDendro = FALSE,
                       saveTOMs = TRUE,
                       saveTOMFileBase = "femaleMouseTOM",
                       verbose = 3)
```

We have chosen the soft thresholding power 6, a relatively large minimum module size of 30, and a medium sensitivity (`deepSplit=2`) to cluster splitting. The parameter `mergeCutHeight` is the threshold for merging of modules. We have also instructed the function to return numeric, rather than color, labels for modules, and to save the Topological Overlap Matrix. The output of the function may seem somewhat cryptic, but it is easy to use. For example, `net$colors` contains the module assignment, and `net$MEs` contains the module eigengenes of the modules.

A word of caution for the readers who would like to adapt this code for their own data. The function `blockwiseModules` has many parameters, and in this example most of them are left at their default value. We have attempted to provide reasonable default values, but they may not be appropriate for the particular data set the reader wishes to analyze. We encourage the user to read the help file provided within the package in the R environment and experiment with tweaking the network construction and module detection parameters. The potential reward is, of course, better (biologically more relevant) results of the analysis.

A second word of caution concerning block size. In particular, the parameter `maxBlockSize` tells the function how large the largest block can be that the reader's computer can handle. The default value is 5000 which is appropriate for most modern desktops. Note that if this code were to be used to analyze a data set with more than 5000 probes, the function `blockwiseModules` will split the data set into several blocks. This will *break some of the plotting code below,* that is executing the code will lead to errors. Readers wishing to analyze larger data sets need to do one of the following:

- If the reader has access to a large workstation with more than 4 GB of memory, the parameter `maxBlockSize` can be increased. A 16GB workstation should handle up to 20000 probes; a 32GB workstation should handle perhaps 30000. A 4GB standard desktop or a laptop may handle up to 8000-10000 probes, depending on operating system and ihow much memory is in use by other running programs.
- If a computer with large-enough memory is not available, the reader should follow Section 2.c, *Dealing with large datasets*, and adapt the code presented there for their needs. In general it is preferable to analyze a data set in one block if possible, although in Section 2.c we present a comparison of block-wise and single-block analysis that indicates that the results are very similar.

We now return to the network analysis. To see how many modules were identified and what the module sizes are, one can use `table(net$colors)`. Its output is

```
> table(net$colors)

 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 99 609 460 409 316 312 221 211 157 123 106 100  94  91  77  76  58  47  34
```

and indicates that there are 18 modules, labeled 1 through 18 in order of descending size, with sizes ranging from 609 to 34 genes. The label 0 is reserved for genes outside of all modules.
The hierarchical clustering dendrogram (tree) used for the module identification is returned in `net$dendrograms[[1]]; #$`.
The dendrogram can be displayed together with the color assignment using the following code:

```
# open a graphics window
sizeGrWindow(12, 9)
# Convert labels to colors for plotting
mergedColors = labels2colors(net$colors)
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net$dendrograms[[1]], mergedColors[net$blockGenes[[1]]],
                    "Module colors",
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05)
```

The resulting plot is shown in Fig. 2. We note that if the user would like to change some of the tree cut, module membership, and module merging criteria, the package provides the function `recutBlockwiseTrees` that can apply modified criteria without having to recompute the network and the clustering dendrogram. This may save a substantial amount of time.
We now save the module assignment and module eigengene information necessary for subsequent analysis.

```
moduleLabels = net$colors
moduleColors = labels2colors(net$colors)
MEs = net$MEs;
geneTree = net$dendrograms[[1]];
save(MEs, moduleLabels, moduleColors, geneTree,
     file = "FemaleLiver-02-networkConstruction-auto.RData")
```

References

[1] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1):Article 17, 2005.

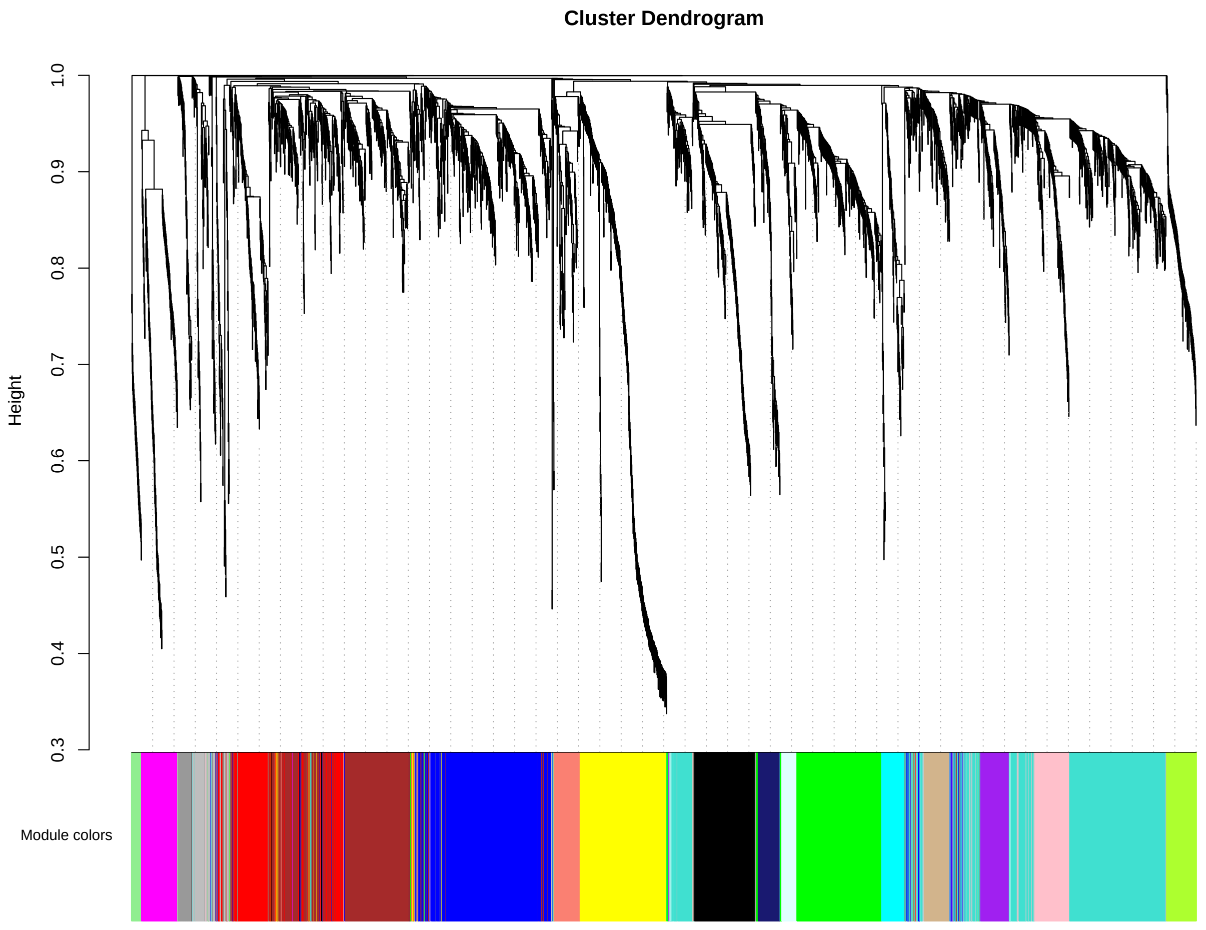

Figure 2: Clustering dendrogram of genes, with dissimilarity based on topological overlap, together with assigned module colors.

Tutorial for the WGCNA package for R: I. Network analysis of liver expression data in female mice

3. Relating modules to external information and identifying important genes

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

0 Preliminaries: setting up the R session and loading results of previous parts

Here we assume that a new R session has just been started. We load the WGCNA package, set up basic parameters and load data saved in previous parts of the tutorial.

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the WGCNA package
library(WGCNA)
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Load the expression and trait data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

We use the network file obtained by the step-by-step network construction and module detection; we encourage the reader to use the results of the other approaches as well.

3 Relating modules to external clinical traits

3.a Quantifying module–trait associations

In this analysis we would like to identify modules that are significantly associated with the measured clinical traits. Since we already have a summary profile (*eigengene*) for each module, we simply correlate eigengenes with external traits and look for the most significant associations:

```
# Define numbers of genes and samples
nGenes = ncol(datExpr);
nSamples = nrow(datExpr);
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(datExpr, moduleColors)$eigengenes
MEs = orderMEs(MEs0)
moduleTraitCor = cor(MEs, datTraits, use = "p");
moduleTraitPvalue = corPvalueStudent(moduleTraitCor, nSamples);
```

Since we have a moderately large number of modules and traits, a suitable graphical representation will help in reading the table. We color code each association by the correlation value:

```
sizeGrWindow(10,6)
# Will display correlations and their p-values
textMatrix = paste(signif(moduleTraitCor, 2), "\n(",
                        signif(moduleTraitPvalue, 1), ")", sep = "");
dim(textMatrix) = dim(moduleTraitCor)
par(mar = c(6, 8.5, 3, 3));
# Display the correlation values within a heatmap plot
labeledHeatmap(Matrix = moduleTraitCor,
             xLabels = names(datTraits),
             yLabels = names(MEs),
             ySymbols = names(MEs),
             colorLabels = FALSE,
             colors = greenWhiteRed(50),
             textMatrix = textMatrix,
             setStdMargins = FALSE,
             cex.text = 0.5,
             zlim = c(-1,1),
             main = paste("Module-trait relationships"))
```

The resulting color-coded table is shown in Fig. 1.
The analysis identifies the several significant module–trait associations. We will concentrate on weight as the trait of interest.

3.b Gene relationship to trait and important modules: Gene Significance and Module Membership

We quantify associations of individual genes with our trait of interest (weight) by defining Gene Significance GS as (the absolute value of) the correlation between the gene and the trait. For each module, we also define a quantitative measure of module membership MM as the correlation of the module eigengene and the gene expression profile. This allows us to quantify the similarity of all genes on the array to every module.

```
# Define variable weight containing the weight column of datTrait
weight = as.data.frame(datTraits$weight_g);
names(weight) = "weight"
# names (colors) of the modules
modNames = substring(names(MEs), 3)

geneModuleMembership = as.data.frame(cor(datExpr, MEs, use = "p"));
MMPvalue = as.data.frame(corPvalueStudent(as.matrix(geneModuleMembership), nSamples));
```

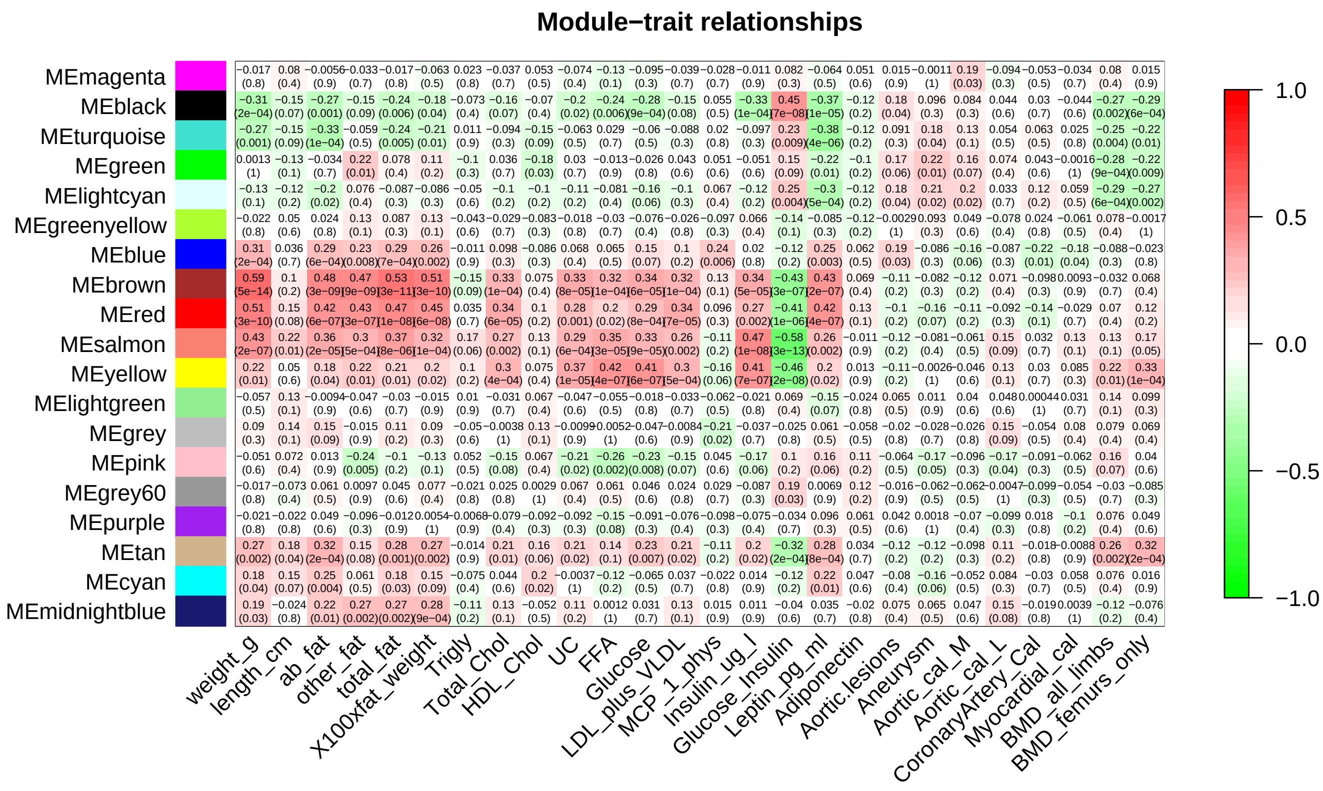

Figure 1: Module-trait associations. Each row corresponds to a module eigengene, column to a trait. Each cell contains the corresponding correlation and p-value. The table is color-coded by correlation according to the color legend.

```
names(geneModuleMembership) = paste("MM", modNames, sep="");
names(MMPvalue) = paste("p.MM", modNames, sep="");

geneTraitSignificance = as.data.frame(cor(datExpr, weight, use = "p"));
GSPvalue = as.data.frame(corPvalueStudent(as.matrix(geneTraitSignificance), nSamples));

names(geneTraitSignificance) = paste("GS.", names(weight), sep="");
names(GSPvalue) = paste("p.GS.", names(weight), sep="");
```

3.c Intramodular analysis: identifying genes with high GS and MM

Using the GS and MM measures, we can identify genes that have a high significance for weight as well as high module membership in interesting modules. As an example, we look at the brown module that has the highest association with weight. We plot a scatterplot of Gene Significance vs. Module Membership in the brown module:

```
module = "brown"
column = match(module, modNames);
moduleGenes = moduleColors==module;

sizeGrWindow(7, 7);
par(mfrow = c(1,1));
verboseScatterplot(abs(geneModuleMembership[moduleGenes, column]),
                 abs(geneTraitSignificance[moduleGenes, 1]),
                 xlab = paste("Module Membership in", module, "module"),
                 ylab = "Gene significance for body weight",
                 main = paste("Module membership vs. gene significance\n"),
                 cex.main = 1.2, cex.lab = 1.2, cex.axis = 1.2, col = module)
```

The plot is shown in Fig. 2. Clearly, GS and MM are highly correlated, illustrating that genes highly significantly associated with a trait are often also the most important (central) elements of modules associated with the trait. The reader is encouraged to try this code with other significance trait/module correlation (for example, the magenta, midnightblue, and red modules with weight).

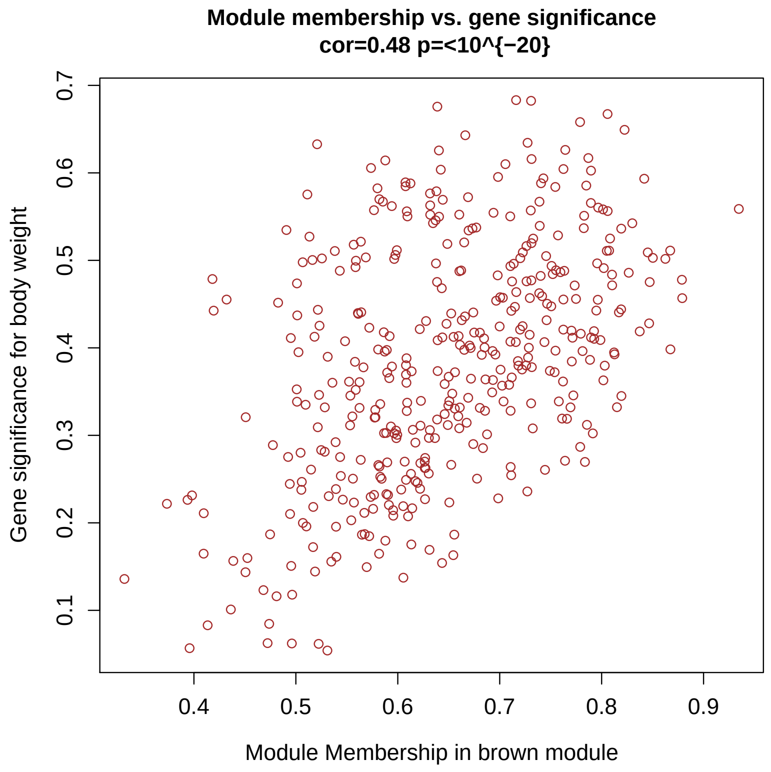

Figure 2: A scatterplot of Gene Significance (GS) for weight vs. Module Membership (MM) in the brown module. There is a highly significant correlation between GS and MM in this module.

3.d Summary output of network analysis results

We have found modules with high association with our trait of interest, and have identified their central players by the Module Membership measure. We now merge this statistical information with gene annotation and write out a file that summarizes the most important results and can be inspected in standard spreadsheet software such as MS Excel or Open Office Calc. Our expression data are only annotated by probe ID names: the command

```
names(datExpr)
```

will return all probe IDs included in the analysis. Similarly,

```
names(datExpr)[moduleColors=="brown"]
```

will return probe IDs belonging to the brown module. To facilitate interpretation of the results, we use a probe annotation file provided by the manufacturer of the expression arrays to connect probe IDs to gene names and universally recognized identification numbers (Entrez codes).

```
annot = read.csv(file = "GeneAnnotation.csv");
dim(annot)
names(annot)
probes = names(datExpr)
probes2annot = match(probes, annot$substanceBXH)
# The following is the number or probes without annotation:
sum(is.na(probes2annot))
# Should return 0.
```

We now create a data frame holding the following information for all probes: probe ID, gene symbol, Locus Link ID (Entrez code), module color, gene significance for weight, and module membership and p-values in all modules. The modules will be ordered by their significance for weight, with the most significant ones to the left.

```
# Create the starting data frame
geneInfo0 = data.frame(substanceBXH = probes,
                      geneSymbol = annot$gene_symbol[probes2annot],
                      LocusLinkID = annot$LocusLinkID[probes2annot],
                      moduleColor = moduleColors,
                      geneTraitSignificance,
                      GSPvalue)
# Order modules by their significance for weight
modOrder = order(-abs(cor(MEs, weight, use = "p")));
# Add module membership information in the chosen order
for (mod in 1:ncol(geneModuleMembership))
{
  oldNames = names(geneInfo0)
  geneInfo0 = data.frame(geneInfo0, geneModuleMembership[, modOrder[mod]],
                         MMPvalue[, modOrder[mod]]);
  names(geneInfo0) = c(oldNames, paste("MM.", modNames[modOrder[mod]], sep=""),
                       paste("p.MM.", modNames[modOrder[mod]], sep=""))
}
# Order the genes in the geneInfo variable first by module color, then by geneTraitSignificance
geneOrder = order(geneInfo0$moduleColor, -abs(geneInfo0$GS.weight));
geneInfo = geneInfo0[geneOrder, ]
```

This data frame can be written into a text-format spreadsheet, for example by

```
write.csv(geneInfo, file = "geneInfo.csv")
```

The reader is encouraged to open and view the file in a spreadsheet software, or inspect it directly within R using the command `fix(geneInfo)`.

Tutorial for the WGCNA package for R: I. Network analysis of liver expression data in female mice

Interfacing network analysis with other data such as functional annotation and gene ontology

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

0 Preliminaries: setting up the R session

Here we assume that a new R session has just been started. We load the WGCNA package, set up basic parameters and load data saved in previous parts of the tutorial.

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the WGCNA package
library(WGCNA)
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Load the expression and trait data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

We use the network file obtained by the step-by-step network construction and module detection; we encourage the reader to use the results of the other approaches as well.

4 Interfacing network analysis with other data such as functional annotation and gene ontology

Our previous analysis has identified several modules (labeled brown, red, and salmon) that are highly associated with weight. To facilitate a biological interpretation, we would like to know the gene ontologies of the genes in the modules, whether they are significantly enriched in certain functional categories etc.

4.a Output gene lists for use with online software and services

One option is to simply export a list of gene identifiers that can be used as input for several popular gene ontology and functional enrichment analysis suites such as David or AmiGO. For example, we write out the LocusLinkID (entrez) codes for the brown module into a file:

```
# Read in the probe annotation
annot = read.csv(file = "GeneAnnotation.csv");
# Match probes in the data set to the probe IDs in the annotation file
probes = names(datExpr)
probes2annot = match(probes, annot$substanceBXH)
# Get the corresponding Locuis Link IDs
allLLIDs = annot$LocusLinkID[probes2annot];
# $ Choose interesting modules
intModules = c("brown", "red", "salmon")
for (module in intModules)
{
  # Select module probes
  modGenes = (moduleColors==module)
  # Get their entrez ID codes
  modLLIDs = allLLIDs[modGenes];
  # Write them into a file
  fileName = paste("LocusLinkIDs-", module, ".txt", sep="");
  write.table(as.data.frame(modLLIDs), file = fileName,
              row.names = FALSE, col.names = FALSE)
}
# As background in the enrichment analysis, we will use all probes in the analysis.
fileName = paste("LocusLinkIDs-all.txt", sep="");
write.table(as.data.frame(allLLIDs), file = fileName,
            row.names = FALSE, col.names = FALSE)
```

4.b Enrichment analysis directly within R

The WGCNA package now contains a function to perform GO enrichment analysis using a simple, single step. To run the function, Biconductor packages GO.db, AnnotationDBI, and the appropriate organism-specific annotation package(s) need to be installed before running this code. The organism-specific packages have names of the form org.Xx.eg.db, where Xx stands for organism code, for example, Mm for mouse, Hs for human, etc. The only exception is yeast, for which no org.Xx.eg.db package is available; instead, the package carries the name org.Sc.sgd.db. Please visit the Bioconductor main page at http://www.bioconductor.org to download and install the required packages. In our case we are studying gene expressions from mice, so this code needs the package org.Mm.eg.db. Calling the GO enrichment analysis function `GOenrichmentAnalysis` is very simple. The function takes a vector of module labels, and the Entrez (a.k.a. Locus Link) codes for the genes whose labels are given.

```
GOenr = GOenrichmentAnalysis(moduleColors, allLLIDs, organism = "mouse", nBestP = 10);
```

The function runs for awhile and returns a long list, the most interesting component of which is

```
tab = GOenr$bestPTerms[[4]]$enrichment
```

This is an enrichment table containing the 10 best terms for each module present in `moduleColors`. Names of the columns within the table can be accessed by

```
names(tab)
```

We refer the reader to the help page of the function within R (available using `?GOenrichmentAnalysis` at the R prompt) for details of what each column means. Because the term definitions can be quite long, the table is a bit difficult to display on the screen. For readers who prefer to look at tables in Excel or similar spreadsheet software, it is best to save the table into a file and open it using their favorite tool:

```
write.table(tab, file = "GOEnrichmentTable.csv", sep = ",", quote = TRUE, row.names = FALSE)
```

On the other hand, to quickly take a look at the results, one can also abridge the table a bit and display it directly on screen:

```
keepCols = c(1, 2, 5, 6, 7, 12, 13);
screenTab = tab[, keepCols];
# Round the numeric columns to 2 decimal places:
numCols = c(3, 4);
screenTab[, numCols] = signif(apply(screenTab[, numCols], 2, as.numeric), 2)
# Truncate the the term name to at most 40 characters
screenTab[, 7] = substring(screenTab[, 7], 1, 40)
# Shorten the column names:
colnames(screenTab) = c("module", "size", "p-val", "Bonf", "nInTerm", "ont", "term name");
rownames(screenTab) = NULL;
# Set the width of R's output. The reader should play with this number to obtain satisfactory output.
options(width=95)
# Finally, display the enrichment table:
screenTab
```

The table is quite long and the reader will have to scroll through the output to see it all. Several of the modules have very significant enrichment, for example the blue, brown, and salmon modules.

Tutorial for the WGCNA package for R:
I. Network analysis of liver expression data in female mice

6. Exporting a gene network to external visualization software

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

1 Preliminaries: setting up the R session and loading results of previous parts

Here we assume that a new R session has just been started. We load the WGCNA package, set up basic parameters and load data saved in previous parts of the tutorial.

```r
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the WGCNA package
library(WGCNA)
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Load the expression and trait data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

We use the network file obtained by the step-by-step network construction and module detection; we encourage the reader to use the results of the other approaches as well.

6 Exporting network data to network visualization software

6.a Exporting to VisANT

The package provides a convenient function for exporting the network to VisANT [1]. We illustrate a simple export of the full weighted network of a single module.

```
# Recalculate topological overlap
TOM = TOMsimilarityFromExpr(datExpr, power = 6);
# Read in the annotation file
annot = read.csv(file = "GeneAnnotation.csv");
# Select module
module = "brown";
# Select module probes
probes = names(datExpr)
inModule = (moduleColors==module);
modProbes = probes[inModule];
# Select the corresponding Topological Overlap
modTOM = TOM[inModule, inModule];
dimnames(modTOM) = list(modProbes, modProbes)
# Export the network into an edge list file VisANT can read
vis = exportNetworkToVisANT(modTOM,
  file = paste("VisANTInput-", module, ".txt", sep=""),
  weighted = TRUE,
  threshold = 0,
  probeToGene = data.frame(annot$substanceBXH, annot$gene_symbol) )
```

Because the brown module is rather large, we can restrict the genes in the output to say the 30 top hub genes in the module:

```
nTop = 30;
IMConn = softConnectivity(datExpr[, modProbes]);
top = (rank(-IMConn) <= nTop)
vis = exportNetworkToVisANT(modTOM[top, top],
  file = paste("VisANTInput-", module, "-top30.txt", sep=""),
  weighted = TRUE,
  threshold = 0,
  probeToGene = data.frame(annot$substanceBXH, annot$gene_symbol) )
```

To provide an example of a VisANT visualization, we loaded the file produced by the above code in VisANT. For better readability we varied the threshold for displaying a link between two nodes. The results are shown in Fig. 1.

6.b Exporting to Cytoscape

Cytoscape [2] allows the user to input an edge file and a node file, allowing the user to specify for example the link weights and the node colors. Here we demonstrate the output of two modules, the red and brown ones, to Cytoscape.

```
# Recalculate topological overlap if needed
TOM = TOMsimilarityFromExpr(datExpr, power = 6);
# Read in the annotation file
annot = read.csv(file = "GeneAnnotation.csv");
# Select modules
modules = c("brown", "red");
# Select module probes
probes = names(datExpr)
inModule = is.finite(match(moduleColors, modules));
modProbes = probes[inModule];
modGenes = annot$gene_symbol[match(modProbes, annot$substanceBXH)];
# Select the corresponding Topological Overlap
modTOM = TOM[inModule, inModule];
```

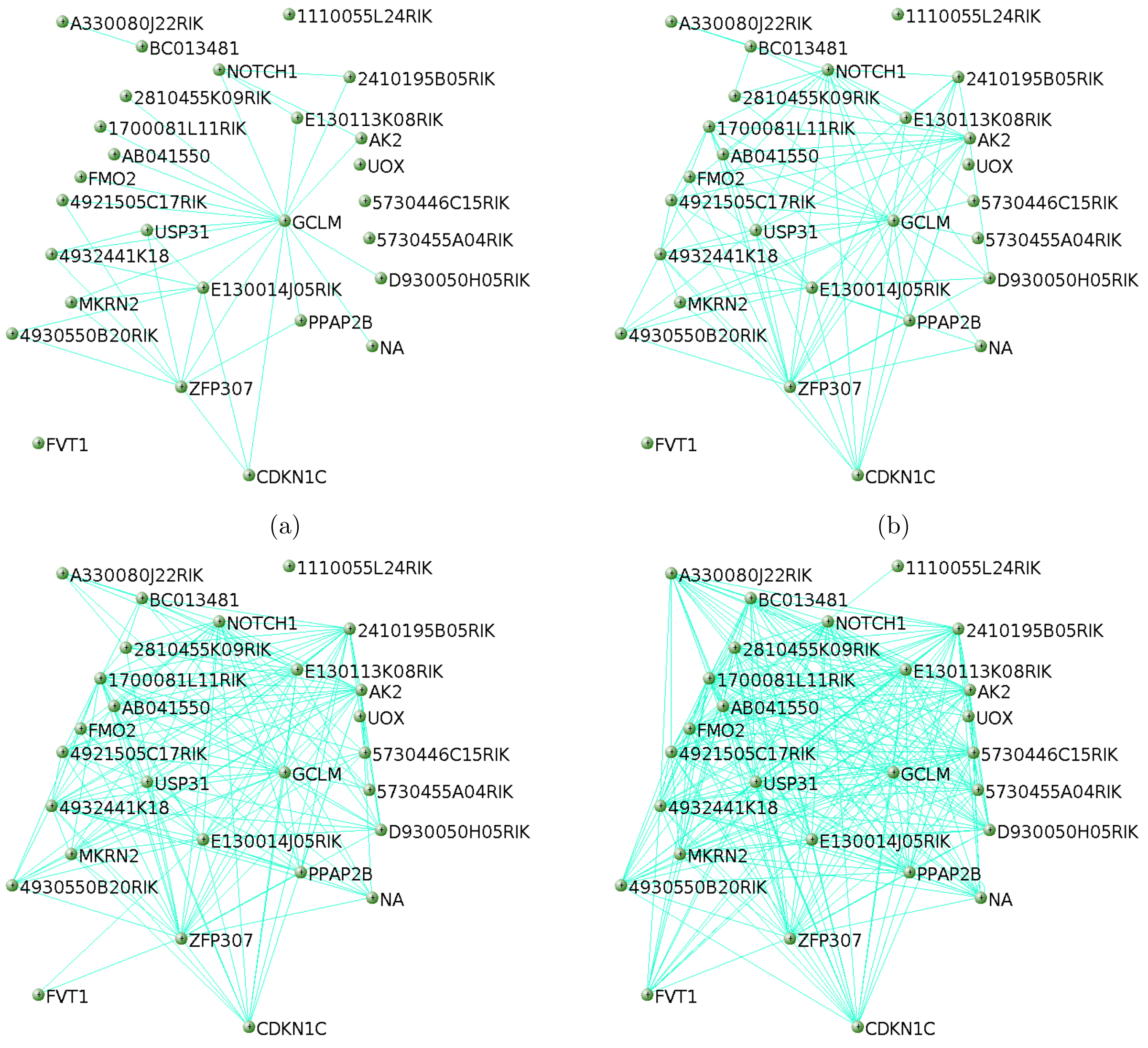

Figure 1: Visualization of the network connections among the most connected genes in the brown module, generated by the VisANT software. The four plots (a), (b), (c), and (d) show network connections whose topological overlap is above the thresholds of 0.10, 0.08, 0.06, and 0.03.

```
dimnames(modTOM) = list(modProbes, modProbes)
# Export the network into edge and node list files Cytoscape can read
cyt = exportNetworkToCytoscape(modTOM,
  edgeFile = paste("CytoscapeInput-edges-", paste(modules, collapse="-"), ".txt", sep=""),
  nodeFile = paste("CytoscapeInput-nodes-", paste(modules, collapse="-"), ".txt", sep=""),
  weighted = TRUE,
  threshold = 0.02,
  nodeNames = modProbes,
  altNodeNames = modGenes,
  nodeAttr = moduleColors[inModule]);
```

Note that network input to Cytoscape is a bit more involved and the user should take care to select all necessary options for the edge and node files to be interpreted correctly. We refer the reader to Cytoscape documentation for all the necessary details.

References

[1] Zhenjun Hu, Evan S. Snitkin, and Charles DeLisi. VisANT: an integrative framework for networks in systems biology. *Brief Bioinform*, 9(4):317–325, 2008.

[2] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13(11):2498–2504, 2003.