Tutorial for the WGCNA package for R
II. Consensus network analysis of liver expression data, female and male mice

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

Peter Langfelder and Steve Horvath

February 13, 2016

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 in the past it **has not worked** with RStudio and possibly other third-party R environments. If you use RStudio or other third-party R environments and you encounter problems, try skipping 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.
# 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 = "Consensus-dataInput.RData");
```

```
# The variable lnames contains the names of loaded variables.
lnames
# Get the number of sets in the multiExpr structure.
nSets = checkSets(multiExpr)$nSets
```

Among other variables we have loaded the variables `multiExpr` and `Traits` containing the expression and trait data, respectively. Further, expression data dimensions are stored in `nGenes` and `nSamples`.

2 Network construction and module detection

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 function for network construction and detection of consensus modules, 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 multiple set network construction and detection of consensus modules. We note that while the actual network construction and module detection is executed in a single function call, a preliminary step of choosing a suitable soft-thresholding power must be performed first.

2.a One-step 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.

```
# Choose a set of soft-thresholding powers
powers = c(seq(4,10,by=1), seq(12,20, 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]]$data, powerVector=powers,
                                                     verbose = 2)[[2]]);
collectGarbage();
# Plot the results:
colors = c("black", "red")
# Will plot these columns of the returned scale free analysis tables
plotCols = c(2,5,6,7)
colNames = c("Scale Free Topology Model Fit", "Mean connectivity", "Median connectivity",
"Max connectivity");
# Get the minima and maxima of the plotted points
ylim = matrix(NA, nrow = 2, ncol = 4);
for (set in 1:nSets)
{
  for (col in 1:length(plotCols))
  {
    ylim[1, col] = min(ylim[1, col], powerTables[[set]]$data[, plotCols[col]], na.rm = TRUE);
    ylim[2, col] = max(ylim[2, col], powerTables[[set]]$data[, plotCols[col]], na.rm = TRUE);
  }
}
# Plot the quantities in the chosen columns vs. the soft thresholding power
sizeGrWindow(8, 6)
#pdf(file = "Plots/scaleFreeAnalysis.pdf", wi = 8, he = 6)
par(mfcol = c(2,2));
par(mar = c(4.2, 4.2 , 2.2, 0.5))
cex1 = 0.7;
for (col in 1:length(plotCols)) for (set in 1:nSets)
{
  if (set==1)
  {
    plot(powerTables[[set]]$data[,1], -sign(powerTables[[set]]$data[,3])*powerTables[[set]]$data[,2],
         xlab="Soft Threshold (power)",ylab=colNames[col],type="n", ylim = ylim[, col],
         main = colNames[col]);
    addGrid();
  }
  if (col==1)
  {
    text(powerTables[[set]]$data[,1], -sign(powerTables[[set]]$data[,3])*powerTables[[set]]$data[,2],
         labels=powers,cex=cex1,col=colors[set]);
  } else
    text(powerTables[[set]]$data[,1], powerTables[[set]]$data[,plotCols[col]],
         labels=powers,cex=cex1,col=colors[set]);
  if (col==1)
  {
    legend("bottomright", legend = setLabels, col = colors, pch = 20) ;
  } else
    legend("topright", legend = setLabels, col = colors, pch = 20) ;
}
dev.off();
```

The result is shown in Fig. 1. We choose the power 6 for both sets.

2.a.2 Network construction and consensus module detection

We call the function `blockwiseConsensusModules`:

```
net = blockwiseConsensusModules(
        multiExpr, power = 6, minModuleSize = 30, deepSplit = 2,
        pamRespectsDendro = FALSE,
        mergeCutHeight = 0.25, numericLabels = TRUE,
        minKMEtoStay = 0,
        saveTOMs = TRUE, verbose = 5)
```

We have chosen the soft thresholding power 6, minimum module size 30, the module detection sensitivity `deepSplit` 2, cut height for merging of modules 0.20 (implying that modules whose eigengenes are correlated above $1 - 0.2 = 0.8$ will be merged), we requested that the function return numeric module labels rather than color labels, we have effectively turned off reassigning genes based on their module eigengene-based connectivity K_{ME}, and we have instructed the code to save the calculated consensus topological overlap.

A word of caution for the readers who would like to adapt this code for their own data. The function `blockwiseConsensusModules` 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

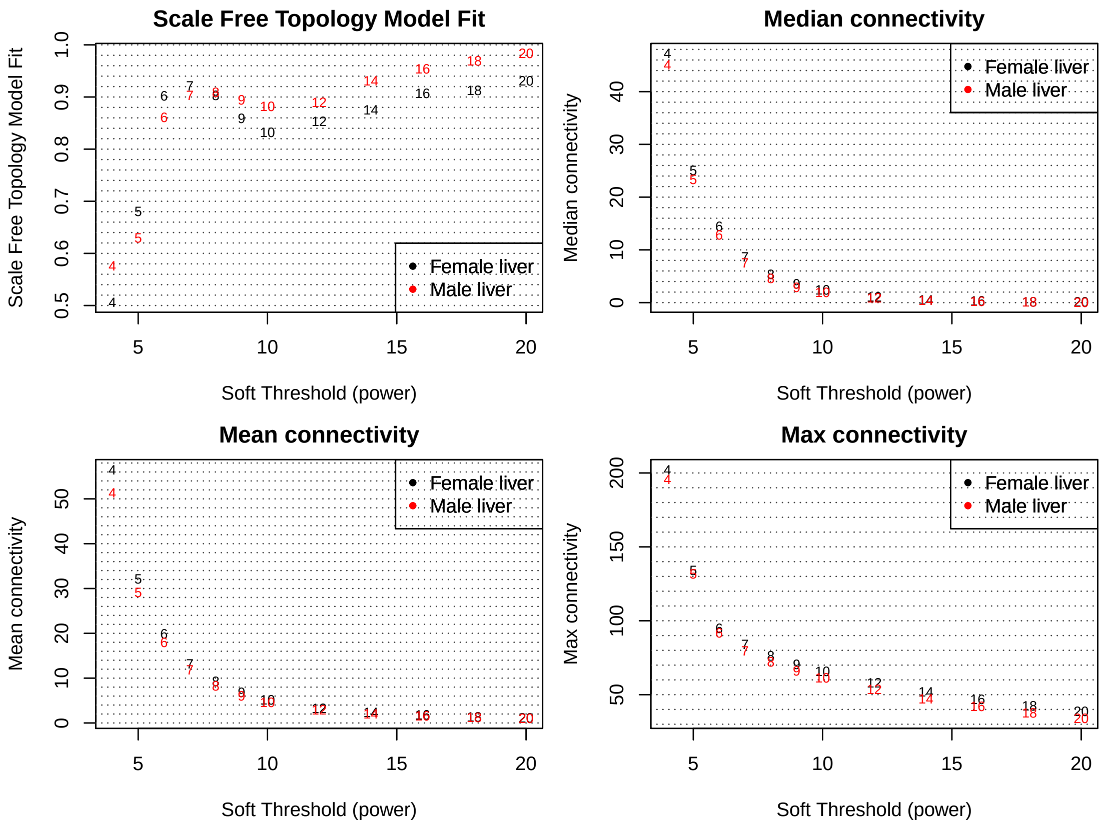

Figure 1: Summary network indices (y-axes) as functions of the soft thresholding power (x-axes). Numbers in the plots indicate the corresponding soft thresholding powers. The plots indicate that approximate scale-free topology is attained around the soft-thresholding power of 6 for both sets. Because the summary connectivity measures decline steeply with increasing soft-thresholding power, it is advantageous to choose the lowest power that satisfies the approximate scale-free topology criterion.

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 `blockwiseConsensusModules` 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 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 similar.

We now return to the analysis. The result `net` has several components:

```
> names(net)
 [1] "colors"                     "unmergedColors"
 [3] "multiMEs"                   "goodSamples"
 [5] "goodGenes"                  "dendrograms"
 [7] "TOMFiles"                   "blockGenes"
 [9] "blocks"                     "originCount"
[11] "networkCalibrationSamples" "individualTOMInfo"
[13] "consensusTOMInfo"           "consensusQuantile"
```

For now we only need the module labels contained in the component `colors`, the module eigengenes for each data set contained in `multiMEs`, and the gene dendrogram (clustering tree) in `dendrograms[[1]]`:

```
consMEs = net$multiMEs;
moduleLabels = net$colors;
# Convert the numeric labels to color labels
moduleColors = labels2colors(moduleLabels)
consTree = net$dendrograms[[1]];
```

A quick way to take a look at the results is to plot the gene dendrogram and the corresponding module colors:

```
sizeGrWindow(8,6);
#pdf(file = "Plots/ConsensusDendrogram-auto.pdf", wi = 8, he = 6)
plotDendroAndColors(consTree, moduleColors,
                    "Module colors",
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05,
                    main = "Consensus gene dendrogram and module colors")

dev.off()
```

The resulting plot is shown in Fig. 2. Before we end, we save the information necessary for the subsequent parts of the tutorial:

```
save(consMEs, moduleLabels, moduleColors, consTree, file = "Consensus-NetworkConstruction-auto.RData")
```

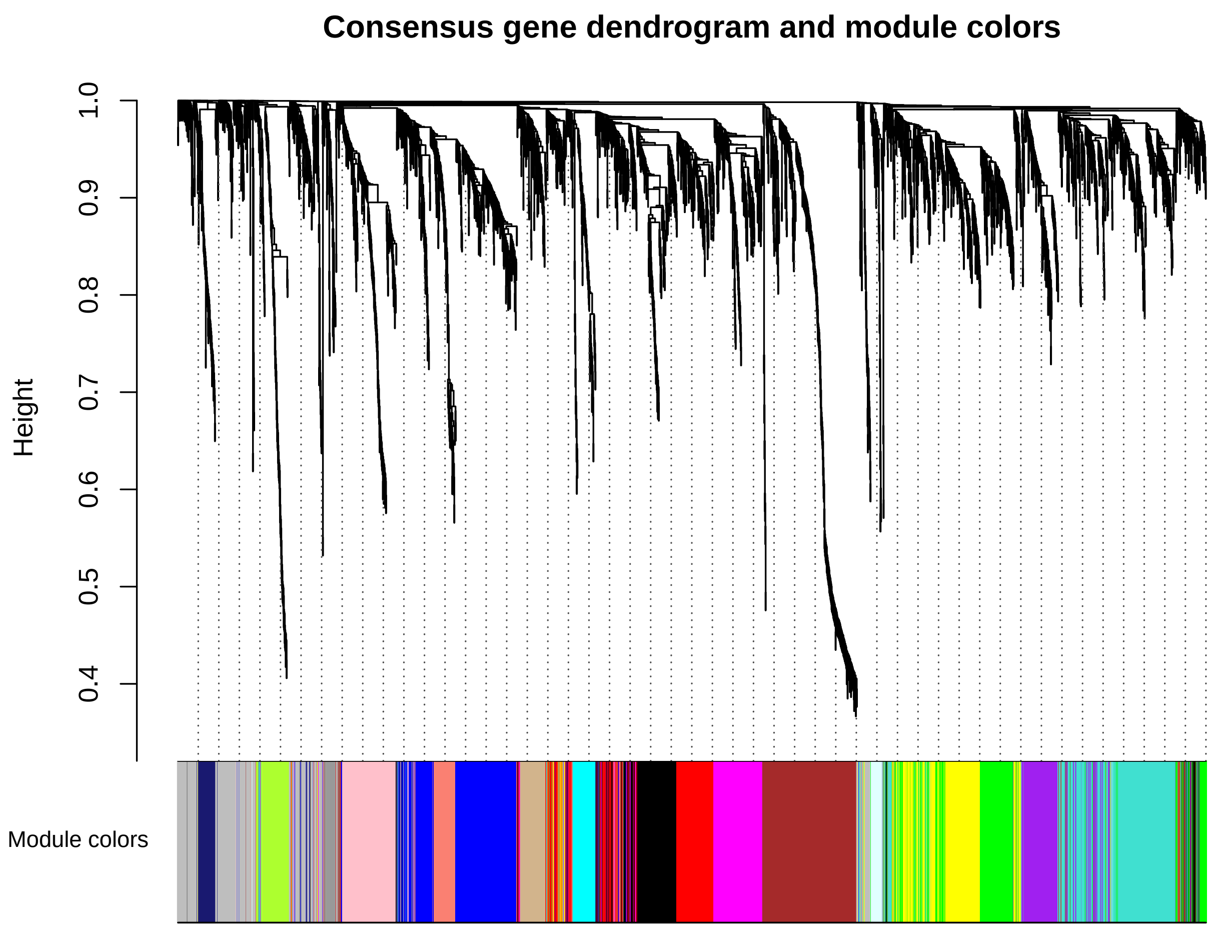

Figure 2: Gene dendrogram obtained by clustering the dissimilarity based on consensus Topological Overlap with the corresponding module colors indicated by the color row.

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.