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

2.c Dealing with large data sets: block-wise network construction and consensus 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 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.
# 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
```

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 block-wise automatic multiple set network construction and detection of consensus modules. The block-wise approach is suitable for large data sets that cannot be handled in a single block. 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. This step is the same in all three methods for network construction and module detection.

2.a Automatic block-wise network construction and consensus 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 Block-wise network construction and consensus module detection

Throughout this tutorial we work with a relatively small data set of 3600 measured probes. However, modern microarrays measure up to 50,000 probe expression levels at once. Constructing and analyzing networks (in particular, consensus networks) with such large numbers of nodes is computationally challenging even on a large server. We now illustrate a method, implemented in the WGCNA package, that allows the user to perform a consensus network analysis with such a large number of genes. Instead of actually using a very large data set, we will for simplicity pretend that hardware limitations restrict the number of genes that can be analyzed at once to 2000. The basic idea is to use a two-level clustering. First, we use a fast, computationally inexpensive and relatively crude clustering method to pre-cluster genes into consensus blocks of size close to and not exceeding the maximum of 2000 genes. We then perform a full consensus network analysis and module detection in each block separately. At the end, modules whose eigengenes are highly correlated across all sets are merged. The advantage of the block-wise approach is a much smaller memory footprint (which is the main problem with large data sets on standard desktop computers), and a significant speed-up of the calculations. The trade-off is that

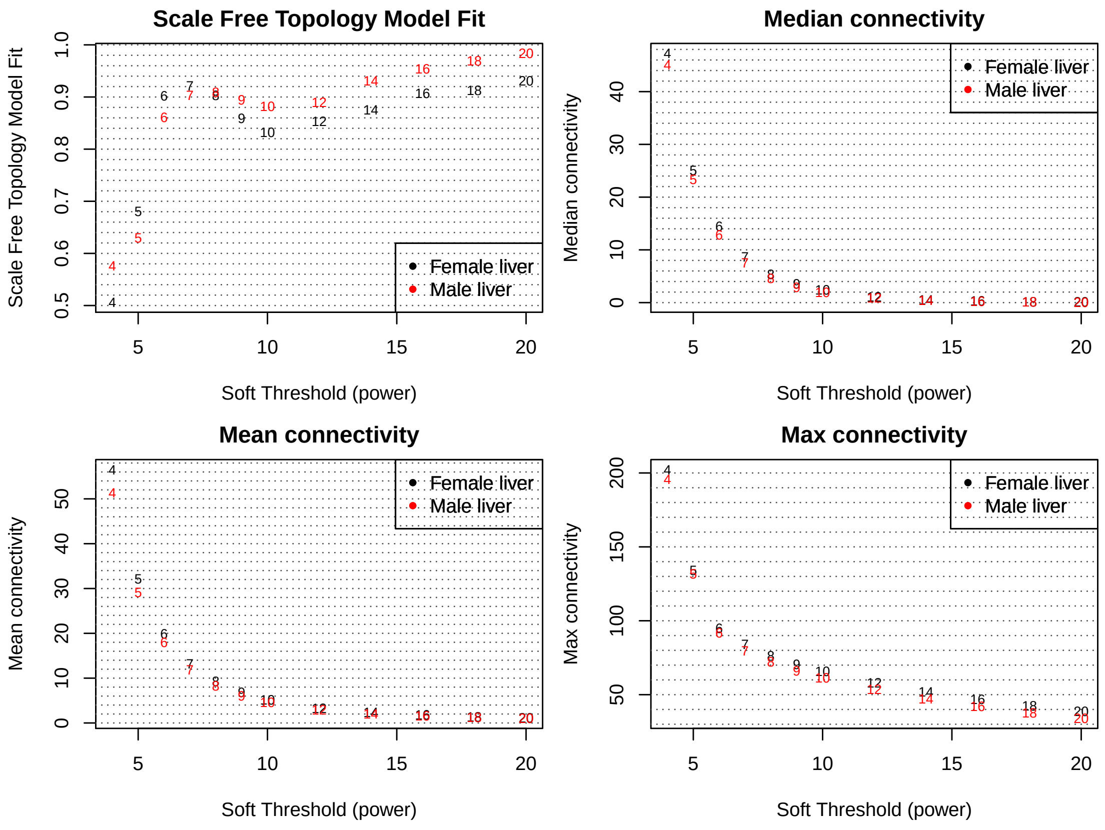

Figure 1: Summary network indices (y-axes) as functions of the soft thresholding power (x-axes). Numbers in the plots indicate the corresponding soft thresholding powers. The plots indicate that approximate scale-free topology is attained around the soft-thresholding power of 6 for both sets. Because 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.

due to using a simpler clustering to obtain blocks, the blocks may not be optimal, causing some outlying genes to be assigned to a different module than they would be in a full network analysis.
We will now pretend that even the relatively small number of genes, 3600, that we have been using here is too large, and the computer we run the analysis on is not capable of handling more than 2000 genes in one block. The automatic network construction and module detection function `blockwiseModules` can handle the splitting into blocks automatically; the user just needs to specify the largest number of genes that can fit in a block:

```
bnet = blockwiseConsensusModules(
    multiExpr, maxBlockSize = 2000, power = 6, minModuleSize = 30,
    deepSplit = 2,
    pamRespectsDendro = FALSE,
    mergeCutHeight = 0.25, numericLabels = TRUE,
    minKMEtoStay = 0,
    saveTOMs = TRUE, verbose = 5)
```

We have chosen the same network construction and module detection parameters 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 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. In this example we have set the maximum block size to 2000 to illustrate the block-wise analysis and its results, but this value is needlessly small for most modern computers; the default is 5000 which is appropriate for most modern desktops. 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. In general it is preferable to analyze a data set in as few blocks as possible.
Below we will compare the results of the block-wise analysis above with the single-block analysis illustrated in Section 2.a. To make the comparison easier, we relabel the block-wise module labels so that modules with a significant overlap with single-block modules have the same label:

```
load(file = "Consensus-NetworkConstruction-auto.RData")
bwLabels = matchLabels(bnet$colors, moduleLabels, pThreshold = 1e-7);
bwColors = labels2colors(bwLabels)
```

To see the results, we first look at the summary of the module labels:

```
table(bwLabels)
bwLabels
  0   1   2   3   4   6   7   8   9  11  12  13  14  15  17  18  19  20
327 688 299 308 439 349 240 212 112 103 132  90  77  64  42  49  39  30
```

Next we plot the gene dendrograms and module color assignment in each block separately:

```
# Here we show a more flexible way of plotting several trees and colors on one page
sizeGrWindow(12,6)
```

```
#pdf(file = "Plots/BlockwiseGeneDendrosAndColors.pdf", wi = 12, he = 6);
# Use the layout function for more involved screen sectioning
layout(matrix(c(1:4), 2, 2), heights = c(0.8, 0.2), widths = c(1,1))
#layout.show(4);
nBlocks = length(bnet$dendrograms)
# Plot the dendrogram and the module colors underneath for each block
for (block in 1:nBlocks)
  plotDendroAndColors(bnet$dendrograms[[block]], moduleColors[bnet$blockGenes[[block]]],
                      "Module colors",
                      main = paste("Gene dendrogram and module colors in block", block),
                      dendroLabels = FALSE, hang = 0.03,
                      addGuide = TRUE, guideHang = 0.05,
                      setLayout = FALSE)
dev.off();
```

The result is shown in Fig. 2.

2.a.3 Comparing the block-wise and standard modules

How do the block-wise modules compare to the modules obtained by a standard, single-block analysis? Recall that the single-block analysis found 19 modules; here we are presented with only 15 modules. To see the relationship between the standard single-block analysis and the block-wise analysis, we load the results of the single-block analysis and plot the single-block gene dendrogram together with the single-block and block-wise module colors:

```
sizeGrWindow(12,9);
#pdf(file="Plots/SingleDendro-BWColors.pdf", wi = 12, he = 9);
plotDendroAndColors(consTree,
                    cbind(moduleColors, bwColors),
                    c("Single block", "Blockwise"),
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05,
                    main = "Single block consensus gene dendrogram and module colors")
dev.off();
```

The plot is shown in Fig. 3. The plot shows that there is actually very good correspondence between the single-block and block-wise modules; however, some of the single-block modules appear merged in the block-wise analysis. As we show in Section 4 of these tutorials, the merged modules formed meta-modules, that is their eigengenes were correlated. Such modules are often merged, but in the single-block analysis their eigengenes fell just outside of the merging threshold. An alternative, more quantitative but perhaps less intuitive, way of comparing two different module assignments is illustrated in part 3 of this tutorial; we leave it as an exercise for the interested reader to adapt the code of part 3 to compare the single-block and block-wise module colors.

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.

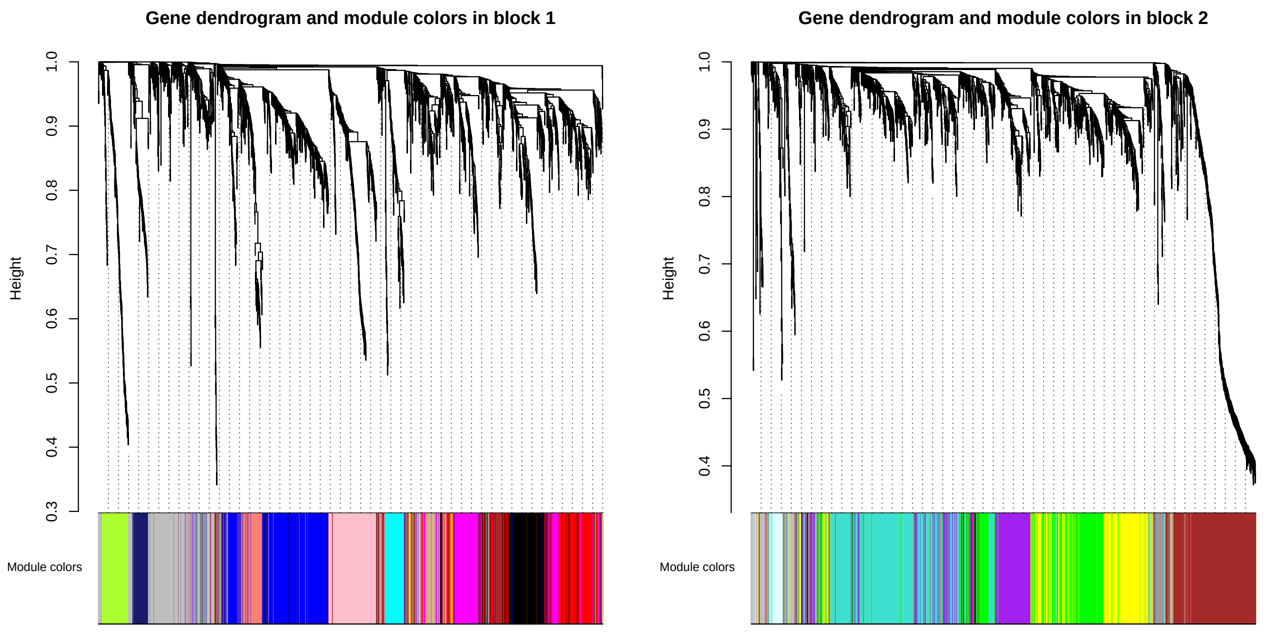

Figure 2: Gene dendrograms obtained by block-wise clustering the dissimilarity based on consensus Topological Overlap with the corresponding module colors indicated by the color row.

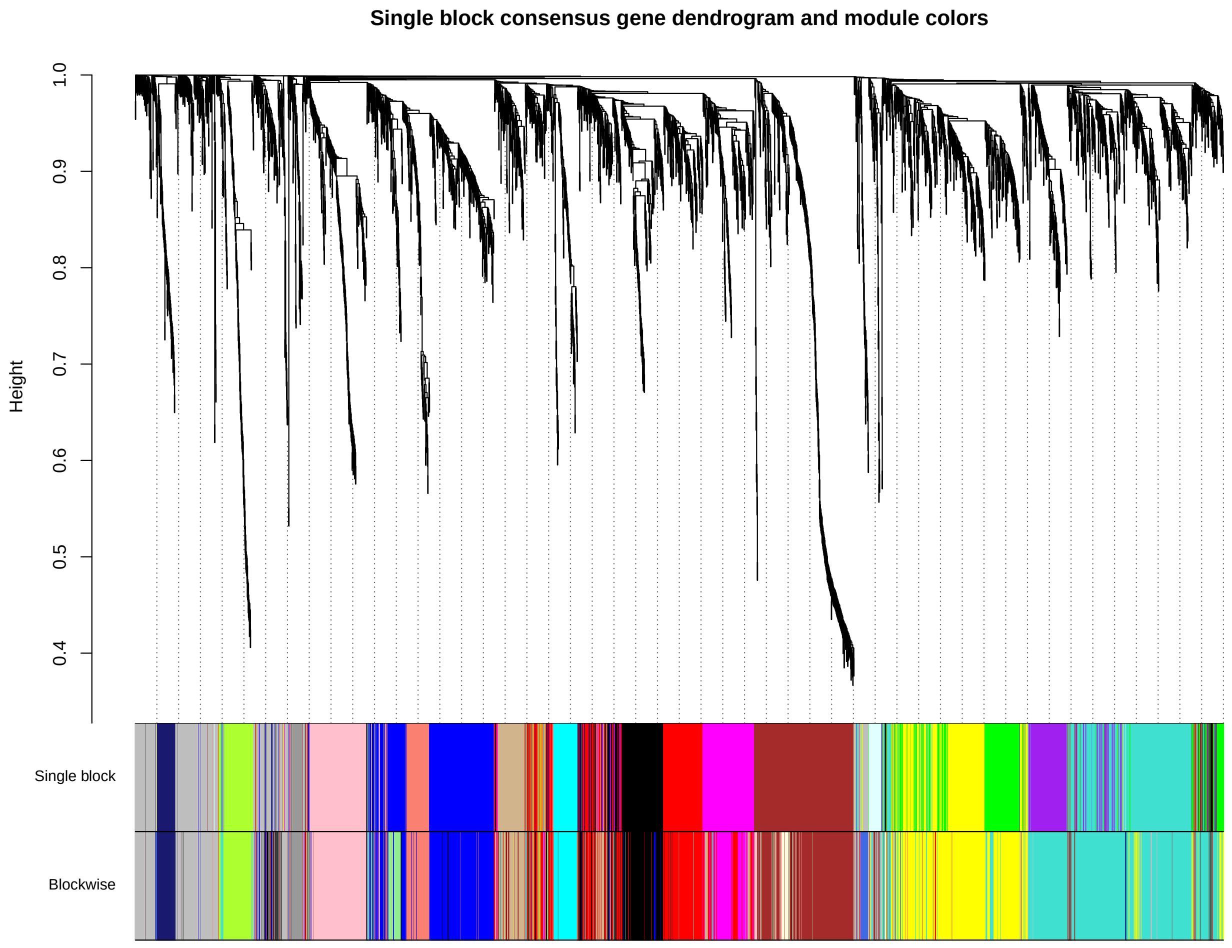

Figure 3: Gene dendrogram obtained in the single-block analysis together with the corresponding single-block module colors and the module colors obtained by block-wise clustering. The plot indicates a strong correspondence between the standard and the block-wise modules.