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

2.b Step-by-step 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 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 provide a detailed illustration of multiple set network construction and detection of consensus modules. We note that the one-step approach is preferable unless the user wishes to examine the calculations in deep details and experiment with alternative approaches that are not available using the one-step function. For completeness, we include the preliminary step of choosing a suitable soft-thresholding power that is copied verbatim from the tutorial illustrating the one-step network construction and module detection.

2.a Step-by-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 [3]. The authors of [3] 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)
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) ;
}
```

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

2.a.2 Calculation of network adjacencies

Network construction starts by calculating the adjacencies in the individual sets, using the soft thresholding power 6:

```
softPower = 6;
# Initialize an appropriate array to hold the adjacencies
adjacencies = array(0, dim = c(nSets, nGenes, nGenes));
# Calculate adjacencies in each individual data set
for (set in 1:nSets)
  adjacencies[set, , ] = abs(cor(multiExpr[[set]]$data, use = "p"))^softPower;
```

2.a.3 Calculation of Topological Overlap

We now turn the adjacencies into Topological Overlap Matrix (TOM) [2, 3]:

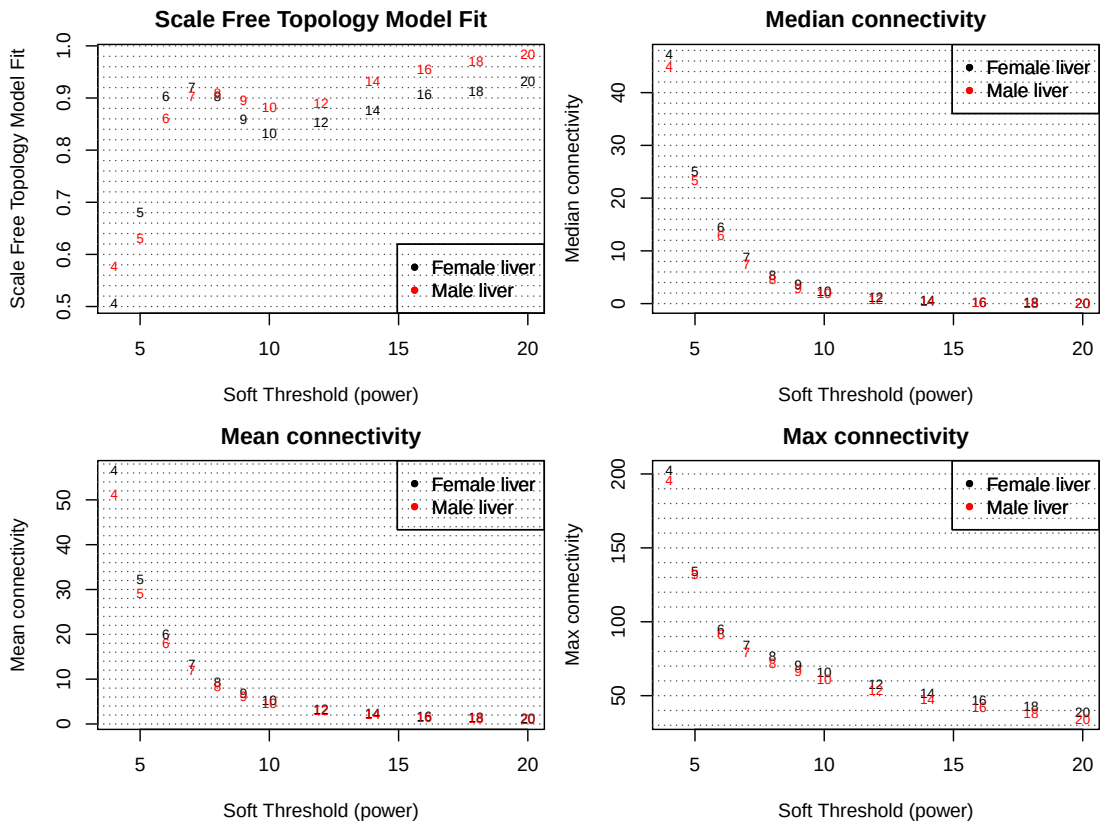

Figure 1: Summary network indices (y-axes) as functions of the soft thresholding power (x-axes). Numbers in the plots indicate the corresponding soft thresholding powers. The plots indicate that approximate scale-free topology is attained around the soft-thresholding power of 6 for both sets. Because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

```
# Initialize an appropriate array to hold the TOMs
TOM = array(0, dim = c(nSets, nGenes, nGenes));
# Calculate TOMs in each individual data set
for (set in 1:nSets)
  TOM[set, , ] = TOMsimilarity(adjacencies[set, , ]);
```

2.a.4 Scaling of Topological Overlap Matrices to make them comparable across sets

Topological Overlap Matrices of different data sets may have different statistical properties. For example, the TOM in the male data may be systematically lower than the TOM in female data. Since consensus is defined as the component-wise minimum of the two TOMs, a bias may result. Here we illustrate a simple scaling that mitigates the effect of different statistical properties to some degree. We scale the male TOM such that the 95th percentile equals the 95th percentile of the female TOM:

```
# Define the reference percentile
scaleP = 0.95
# Set RNG seed for reproducibility of sampling
set.seed(12345)
# Sample sufficiently large number of TOM entries
nSamples = as.integer(1/(1-scaleP) * 1000);
# Choose the sampled TOM entries
scaleSample = sample(nGenes*(nGenes-1)/2, size = nSamples)
TOMScalingSamples = list();
# These are TOM values at reference percentile
scaleQuant = rep(1, nSets)
# Scaling powers to equalize reference TOM values
scalePowers = rep(1, nSets)
# Loop over sets
for (set in 1:nSets)
{
  # Select the sampled TOM entries
  TOMScalingSamples[[set]] = as.dist(TOM[set, , ])[scaleSample]
  # Calculate the 95th percentile
  scaleQuant[set] = quantile(TOMScalingSamples[[set]],
                             probs = scaleP, type = 8);
  # Scale the male TOM
  if (set>1)
  {
    scalePowers[set] = log(scaleQuant[1])/log(scaleQuant[set]);
    TOM[set, ,] = TOM[set, ,]^scalePowers[set];
  }
}
```

The array `TOM` now contains the scaled TOMs. To see what the scaling achieved, we form a quantile-quantile plot of the male and female topological overlaps before and after scaling:

```
# For plotting, also scale the sampled TOM entries
scaledTOMSamples = list();
for (set in 1:nSets)
  scaledTOMSamples[[set]] = TOMScalingSamples[[set]]^scalePowers[set]
# Open a suitably sized graphics window
sizeGrWindow(6,6)
#pdf(file = "Plots/TOMScaling-QQPlot.pdf", wi = 6, he = 6);
# qq plot of the unscaled samples
qqUnscaled = qqplot(TOMScalingSamples[[1]], TOMScalingSamples[[2]], plot.it = TRUE, cex = 0.6,
                    xlab = paste("TOM in", setLabels[1]), ylab = paste("TOM in", setLabels[2]),
```

```
                  main = "Q-Q plot of TOM", pch = 20)
# qq plot of the scaled samples
qqScaled = qqplot(scaledTOMSamples[[1]], scaledTOMSamples[[2]], plot.it = FALSE)
points(qqScaled$x, qqScaled$y, col = "red", cex = 0.6, pch = 20);
abline(a=0, b=1, col = "blue")
legend("topleft", legend = c("Unscaled TOM", "Scaled TOM"), pch = 20, col = c("black", "red"))
dev.off();
```

The result is shown in Fig. 2. In this case the scaling changed the male TOM only very slightly, and brought it closer to the reference line shown in blue.

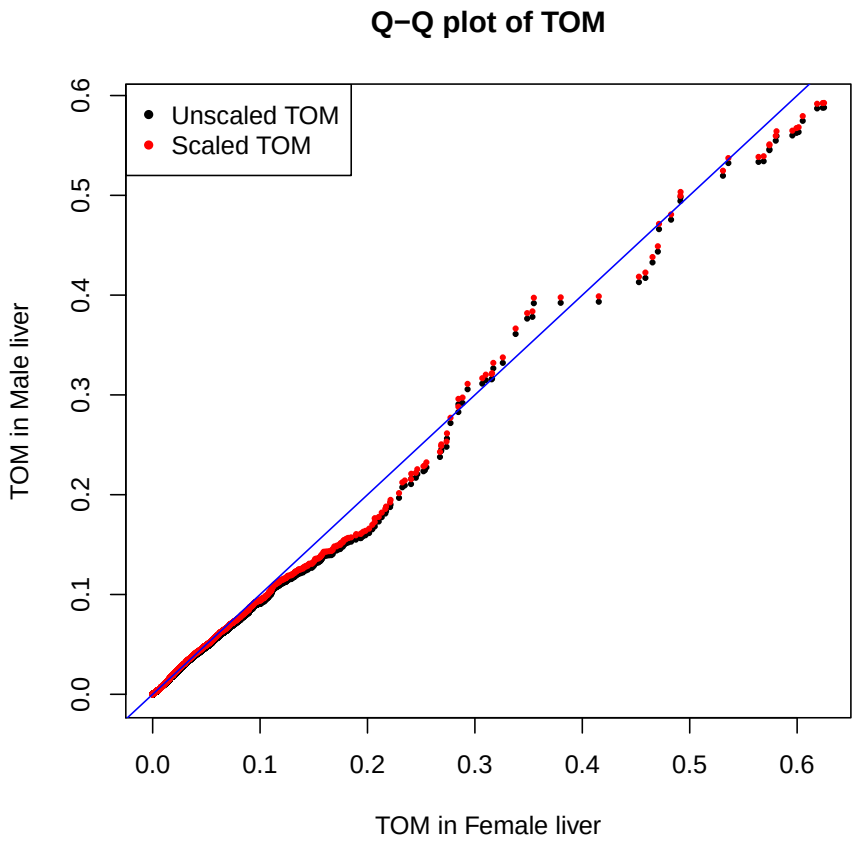

Figure 2: Quantile-quantile plot of the TOMs in male and female data sets. The black points are TOMs before scaling, the red points are TOMs after scaling. The closer the points lie to the reference line shown in blues, the closer is the distribution of the TOM values in the two data sets.

2.a.5 Calculation of consensus Topological Overlap

We now calculate the consensus Topological Overlap by taking the component-wise ("parallel") minimum of the TOMs in individual sets:

```
consensusTOM = pmin(TOM[1, , ], TOM[2, , ]);
```

Thus, the consensus topological overlap of two genes is only large if the corresponding entries in the two sets are also large.

2.a.6 Clustering and module identification

We use the consensus TOM as input to hierarchical clustering, and identify modules in the resulting dendrogram using the Dynamic Tree Cut algorithm [1]

```
# Clustering
consTree = hclust(as.dist(1-consensusTOM), method = "average");
# We like large modules, so we set the minimum module size relatively high:
minModuleSize = 30;
# Module identification using dynamic tree cut:
unmergedLabels = cutreeDynamic(dendro = consTree, distM = 1-consensusTOM,
              deepSplit = 2, cutHeight = 0.995,
              minClusterSize = minModuleSize,
              pamRespectsDendro = FALSE );
unmergedColors = labels2colors(unmergedLabels)
```

To see a quick summary of the module detection, we use `table(unmergedLabels)`:

```
> table(unmergedLabels)
unmergedLabels
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
343 343 285 274 273 264 261 239 206 184 134 108 105 105  99  93  80  62  61  44
 20
 37
```

The Dynamic Tree Cut returned 20 proper modules with sizes ranging from 343 to 37 genes. The label 0 is reserved for genes not assigned to any of the modules. The following code plots the consensus gene dendrogram together with the preliminary module colors:

```
sizeGrWindow(8,6)
plotDendroAndColors(consTree, unmergedColors, "Dynamic Tree Cut",
                  dendroLabels = FALSE, hang = 0.03,
                  addGuide = TRUE, guideHang = 0.05)
```

The resulting plot is shown in Fig. 3.

2.a.7 Merging of modules whose expression profiles are very similar

The Dynamic Tree Cut may identify modules whose expression profiles are very similar. It may be prudent to merge such modules since their genes are highly co-expressed. To quantify co-expression similarity of entire modules, we calculate their eigengenes (MEs) and cluster them on their consensus correlation, that is the minimum correlation across the two sets:

```
# Calculate module eigengenes
unmergedMEs = multiSetMEs(multiExpr, colors = NULL, universalColors = unmergedColors)
# Calculate consensus dissimilarity of consensus module eigengenes
consMEDiss = consensusMEDissimilarity(unmergedMEs);
# Cluster consensus modules
consMETree = hclust(as.dist(consMEDiss), method = "average");
# Plot the result
sizeGrWindow(7,6)
par(mfrow = c(1,1))
plot(consMETree, main = "Consensus clustering of consensus module eigengenes",
     xlab = "", sub = "")
abline(h=0.25, col = "red")
```

The resulting tree is shown in Fig. 4. One pair of modules falls below the merging threshold. The merging can be performed automatically:

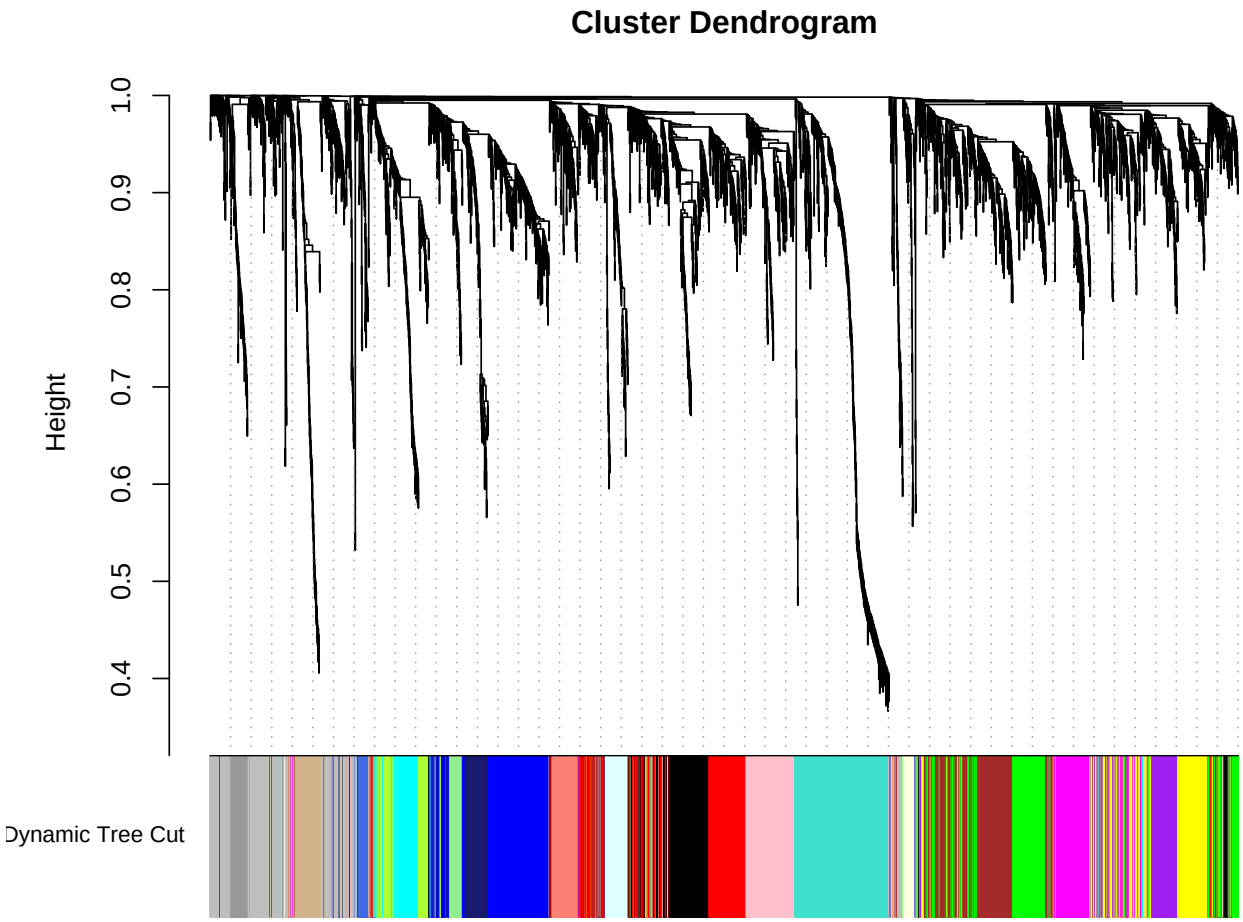

Figure 3: Gene dendrogram obtained by clustering based on the consensus topological overlap across male and female mice. The identified preliminary modules are indicated in the color row beneath the dendrogram.

```
merge = mergeCloseModules(multiExpr, unmergedLabels, cutHeight = 0.25, verbose = 3)
```

The variable `merge` contains various information; we will need the following:

```
# Numeric module labels
moduleLabels = merge$colors;
# Convert labels to colors
moduleColors = labels2colors(moduleLabels)
# Eigengenes of the new merged modules:
consMEs = merge$newMEs;
```

Lastly, we plot the gene dendrogram again, this time with both the unmerged and the merged module colors:

```
sizeGrWindow(9,6)
plotDendroAndColors(consTree, cbind(unmergedColors, moduleColors),
                    c("Unmerged", "Merged"),
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05)
```

The plot is shown in Fig. 5. We now save the information necessary in the subsequent parts of the tutorial:

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

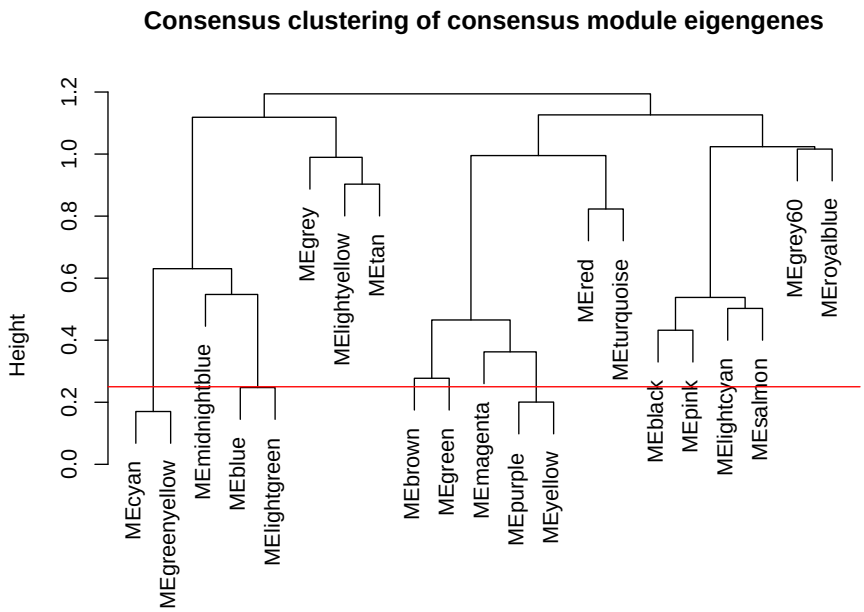

Figure 4: Dendrogram of consensus module eigengenes obtained by clustering the eigengenes on their consensus correlation across male and female mice. The red line is the merging threshold; groups of eigengenes below the threshold represent modules whose expression profiles are too similar and should be merged.

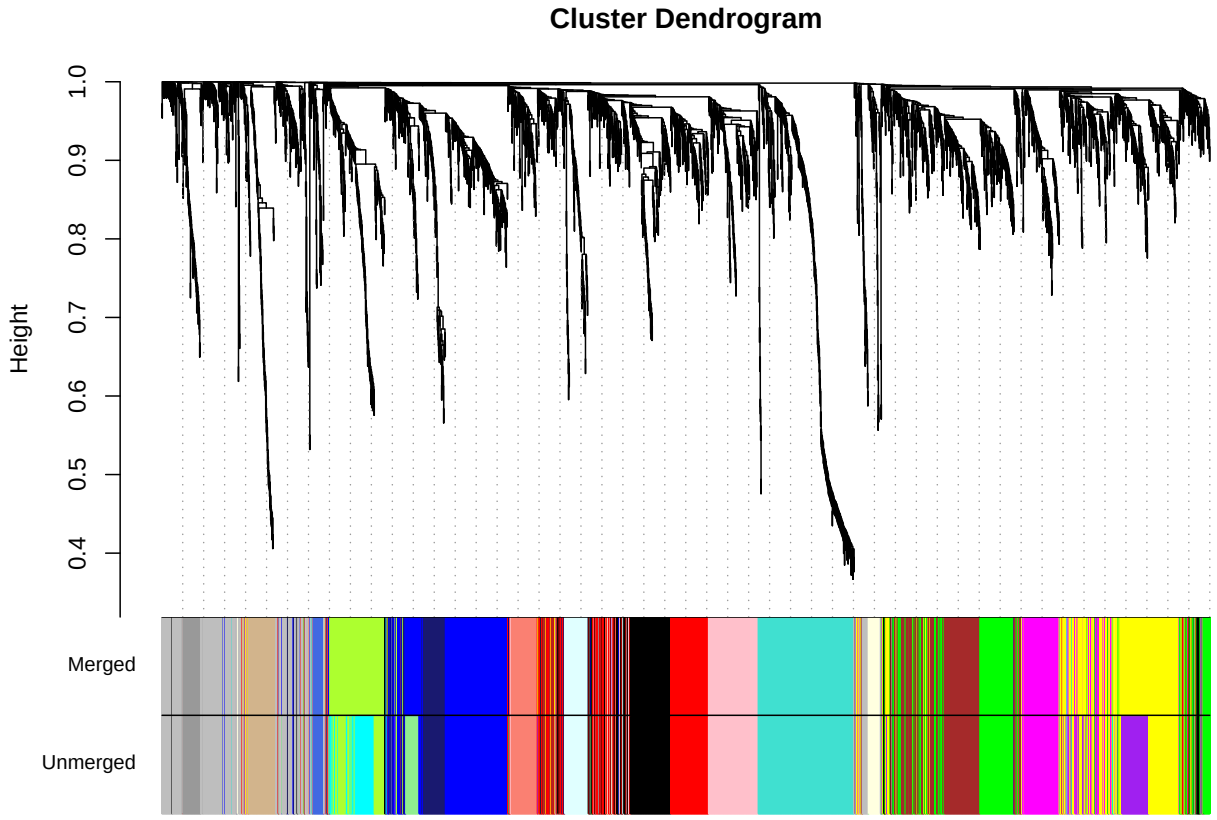

Figure 5: Gene dendrogram obtained by clustering the dissimilarity based on consensus Topological Overlap. The two color rows show the preliminary (unmerged) and the final, merged module assignments.

References

[1] P. Langfelder and S. Horvath. Eigengene networks for studying the relationships between co-expression modules. *BMC Systems Biology*, 1:54, 2007.

[2] E. Ravasz, A. Somera, D. Mongru, Z. Oltvai, and A. Barabási. Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555, 2002.

[3] 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.