Simulation studies of module preservation: Simulation A: half of the reference modules are preserved in test data

Peter Langfelder and Steve Horvath

May 31, 2011

Contents

1 Overview

This tutorial presents a simulation study of module preservation in which we simulate a reference set with 10 modules of sizes between 50 and 1000 profiles ("genes"), and a test set in which 5 of the 10 reference modules (labeled by numbers 1–5) are preserved, and genes in the other 5 modules (labeled by numbers 6–10) are simulated with independent random profiles (in the language of WGCNA these genes are simulated "grey"). We refer the reader to Supplementary Text S5 of the module preservation paper [1] for details.
We encourage readers unfamiliar with any of the functions used in this tutorial to type, in the active R session,

```
help(functionName)
```

(replace `functionName` with the actual name of the function) to get a detailed description of what the functions does, what the input arguments mean, and what is the output.

1.a Setting up the R session

After starting R we execute a few commands to set the working directory and load the requisite packages:

```
# 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 package
library(WGCNA);
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
```

2 Data simulation

We simulate two data sets, each with 100 samples. First we set up simulation parameters such as module sizes etc. and simulate seed module eigenenes. The eigengenes are random vectors with i.i.d. components drawn from $N(0, 1)$.

```
nSamples = c(100, 100);
nMods = 10;
nSets = length(nSamples);
nAllSamples = sum(nSamples);
# Set random seed for reproducibility
set.seed(1)
# Simulate seed eigengenes
allEigengenes = matrix(rnorm(nAllSamples * nMods), nAllSamples, nMods);
# Module sizes
modSizes = c(1000, 500, 250, 100, 50, 1000, 500, 250, 100, 50, 300);
nGenes = as.integer(sum(modSizes)*1.2);
modProps = modSizes/nGenes;
# Eigengenes for each set:
eigengenes = list();
eigengenes[[1]] = list(data = allEigengenes[1:nSamples[1], ]);
eigengenes[[2]] = list(data = allEigengenes[(nSamples[1]+1):nAllSamples, ]);
```

Next we simulate the data using the WGCNA function `simulateMultiExpr`. We define the matrix `leaveOut` which tells the simulation function which modules should be left out in each of the data sets. In this case, we leave out half of the modules in the second data set.

```
leaveOut = matrix(FALSE, nMods, nSets);
leaveOut[c(6,7,8,9,10), 2] = TRUE;
data = simulateMultiExpr(eigengenes, nGenes, modProps, leaveOut = leaveOut,
                         minCor = 0.3, maxCor = 0.9, backgroundNoise = 0.2);
# Generate the multiExpr variable from the simulated data
multiExpr = data$multiExpr;
colnames(multiExpr[[1]]$data) = spaste("Gene", c(1:nGenes));
colnames(multiExpr[[2]]$data) = spaste("Gene", c(1:nGenes));
```

3 Module identification

We now identify modules in the each of the simulated data sets using the WGCNA function `blockwiseModules`.

```
set.seed(3)
mods = list();
for (set in 1:nSets)
  mods[[set]] = blockwiseModules(multiExpr[[set]]$data, numericLabels = TRUE, verbose = 4)
# Correspondence of identified and simulated colors:
table(mods[[1]]$colors, data$allLabels[, 1])
table(mods[[2]]$colors, data$allLabels[, 2])
# We will use the identified module colors for module preservation calculations.
colorList = list();
colorList[[1]] = mods[[1]]$colors;
colorList[[2]] = mods[[2]]$colors;
```

4 Calculation of module preservation

Here we run the main module preservation function `modulePreservation`. After the calculation we save the results; if a re-analysis of previously calculated results is performed, one can simply read the results from disk, thus saving a lot of time.

```
system.time( {
mp = modulePreservation(multiExpr, colorList, referenceNetworks=1,
                        nPermutations = 100,
                        networkType = "unsigned",
                        randomSeed = 2345,
                        permutedStatisticsFile = "halfPreserved-noReg-permStats.RData",
                        verbose = 4, indent = 0)
          } );
# Save the results.
save(mp, file= "simulation-halfPreserved.RData");
```

If the module preservation results have been calculated previously, load the results from the disk:

```
load(file= "simulation-halfPreserved.RData");
```

5 Analysis and visualization of module preservation results

Here analyze the results of the `modulePreservation` calculation. We first collect the calculated Z statistics and set up useful variables:

```
stats = cbind(mp$quality$Z[[1]][[2]][, -1],
              mp$referenceSeparability$Z[[1]][[2]][, -1, drop = FALSE],
              mp$preservation$Z[[1]][[2]][, -1],
              mp$accuracy$Z[[1]][[2]][, -1],
              mp$testSeparability$Z[[1]][[2]][, -1, drop = FALSE]);
# Module labels and colors
order = order(as.numeric(rownames(stats)))
stats = stats[order, ]
labelsX = as.numeric(rownames(stats))
labelsX[labelsX==0.1] = 25
colors = labels2colors(labelsX);
moduleSizes = as.numeric(table(colorList[[1]]));
# Preserved modules (based on the overlap tables above)
preserved = c(1, 3, 5, 7, 10);
presInd = match(preserved, labelsX);
# Plotting variables
letter = labelsX;
presColor = rep(1, length(letter));
presColor[presInd] = 2;
useStats = list(c(1:8), c(9:ncol(stats)));
sectioning = list(c(2,4), c(4,5));
dims = list(sectioning[[1]] * 2.0, sectioning[[2]]*2.0);
figNames = c("quality", "preservation");
# Relabel modules such that 1:5 are preserved, 6:10 are non-preserved
numLabels2 = letter;
numLabels2[presInd] = seq(from = 5, to=1, by = -1);
numLabels2[-presInd] = c(0, 0, seq(from = 10, to=6, by = -1));
numLabels2 = as.numeric(numLabels2)
```

We now plot the results. This code automatically generates two pdf files in a subdirectory Plots (please create the subdirectory in the current directory if it does not exist), one for quality and one for preservation statistics.

```
for (f in 1:2)
{
  pdf(file=spaste("Plots/simulation-halfPreserved-",figNames[f],".pdf"), wi=dims[[f]][2], he=dims[[f]][1])
  par(mfrow = sectioning[[f]])
  par(mar = c(3.2, 3.2, 2, 0.5))
  par(mgp = c(2.0, 0.6, 0))
```

```
  for (s in useStats[[f]])
  {
    if (s < 9 || s==27)
    {
       # Shift module sizes on modules 6--10 to make them distinct from 1-5 in the plot.
       ms = moduleSizes[-1];
       ms[numLabels2[-c(1:2)] > 5] = ms[numLabels2[-c(1:2)] > 5] + 20;
    } else
       ms = moduleSizes[-1];

    min = min(stats[-c(1:1), s], na.rm = TRUE);
    max = max(stats[-c(1:1), s], na.rm = TRUE);
    if (min > -max/5) min = -max/10;
    plot(ms, stats[-c(1:2), s], col = colors, pch = 20,
         main = colnames(stats)[s],
         cex = 2,
         ylab = colnames(stats)[s], type = "n", xlab = "Module size",
         cex.main = 1, ylim = c(min, max))
    text(ms, stats[-c(1:2), s], labels = numLabels2[-c(1:2)], col = presColor[-c(1:2)]);
    box = par("usr");
    if (s==useStats[[f]][1]) legend(x = box[2], y = max/3, xjust = 1, yjust = 0.5,
          legend = c("Preserved", "Non-preserved"), fill = c(2,1), cex = 0.8)
    abline(h=0)
    abline(h=2, col = "blue", lty = 2);
    abline(h=10, col = "darkgreen", lty = 2);
  }
  dev.off();
}
```

The two plots generated by this code are shown in Figures 1 and 2. Since all modules were simulated to be well-defined, most quality statistics are high. The only exception is separability. On the other hand, the preservation statistics distinguish the preserved and non-preserved modules very well, again with the exception of separability. This is why we do not use separability in our summary statistics.

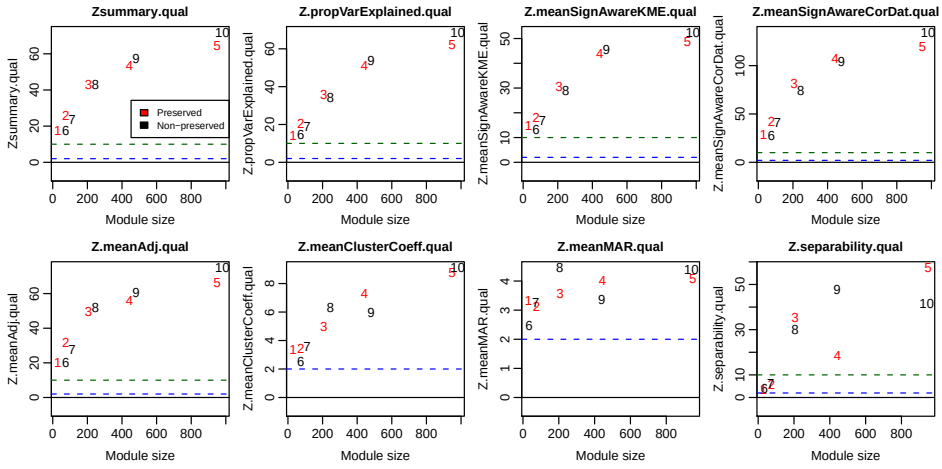

Figure 1: Module quality statistics of simulated modules in this study. Each plot shows one of the quality statistics (indicated in the title) as a function of the module size. Modules are labeled by their numeric labels. The blue and green dashed lines denote the thresholds $Z = 2$ and $Z = 10$. Since all modules were simulated to be well-defined, most quality statistics are high.

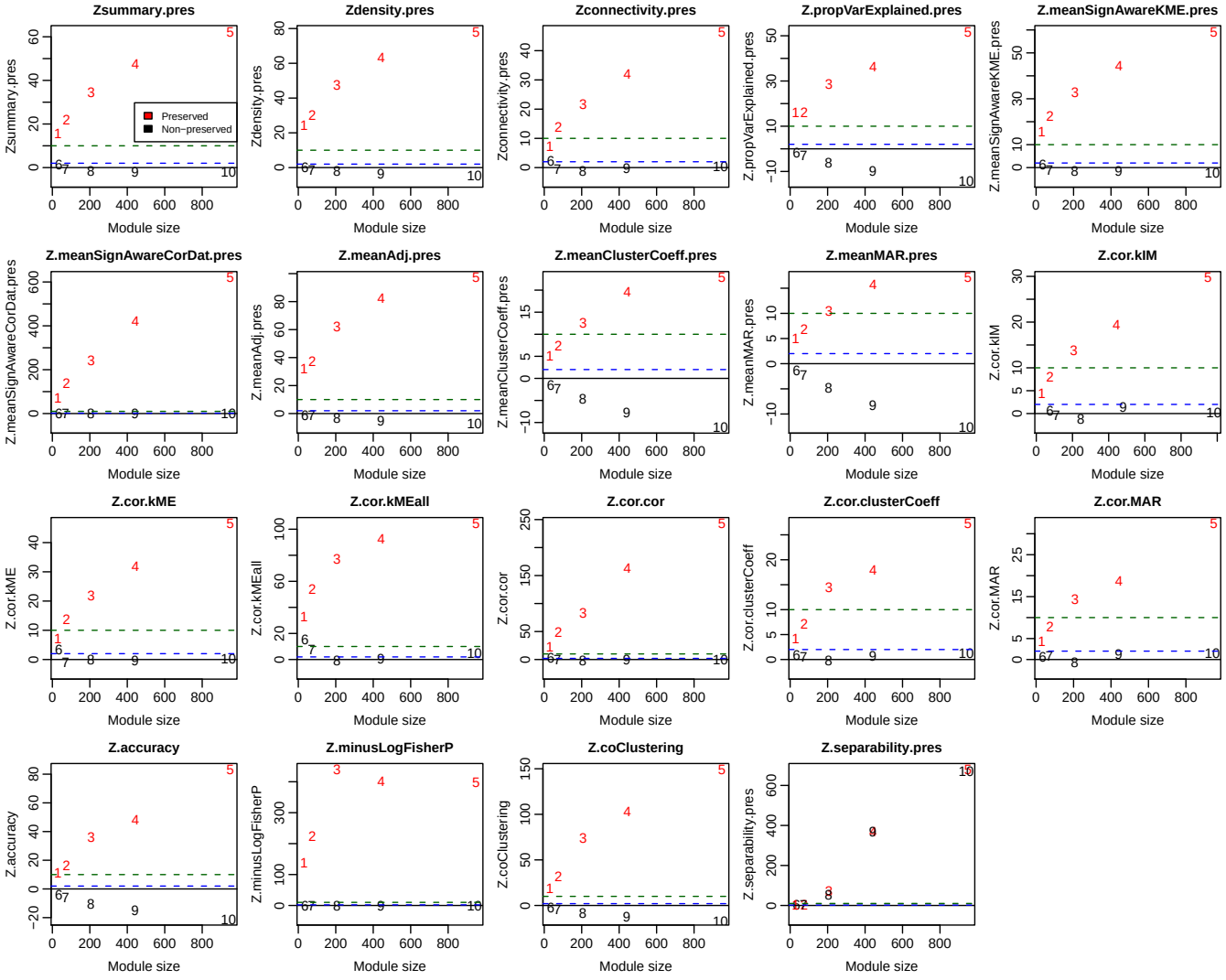

Figure 2: Module preservation statistics of simulated modules in this study. Each plot shows one of the preservation statistics (indicated in the title) as a function of the module size. Modules are labeled by their numeric labels; red color denotes preserved and black non-preserved modules. The blue and green dashed lines denote the thresholds $Z = 2$ and $Z = 10$. In this study all statistics except separability easily distinguish the preserved and non-preserved modules.

6 Alternate methods: clusterRepro

Here we apply the function clusterRepro from the R package of the same name to our simulated data. To run this code, package clusterRepro must be installed.

```
doClusterRepro = TRUE;
if (doClusterRepro)
{
  eigengenes = as.matrix(moduleEigengenes(multiExpr[[2]]$data, colorList[[1]])$eigengenes);
  library(clusterRepro);
  set.seed(10)
  rownames(eigengenes) = rownames(multiExpr[[2]]$data);
  cr = clusterRepro(Centroids = eigengenes, New.data = multiExpr[[2]]$data, Number.of.permutations = 1000);
  save(cr, file = "simulation-halfPreserved-cr.RData");
}
```

Alternatively, if the calculations have been already executed in an earlier session, one can load the results from the disk and save substantial amount of time.

```
load(file = "simulation-halfPreserved-cr.RData");
```

We now plot the results of clusterRepro.

```
# Shift some module sizes a bit to prevent overlapping and unreadable labels
msx = ms = moduleSizes[-1];
msx[numLabels2[-c(1:2)] > 5] = ms[numLabels2[-c(1:2)] > 5] + 20;
# Open a suitably sized graphics window or alternatively a pdf file to hold the result
sizeGrWindow(4,8);
#pdf(file="Plots/simulation-halfPreserved-clusterRepro.pdf", wi=4, he=8);
par(mfrow = c(2,1))
par(mar = c(3.2, 3.2, 2, 0.5))
par(mgp = c(2.0, 0.6, 0))
plot(msx, cr$Actual.IGP, col = colors, pch = 20,
     main = "A. Half-preserved\nObserved IGP",
     cex = 2,
     ylab = "Observed IGP", type = "n", xlab = "Module size",
     cex.main = 1)
text(msx, cr$Actual.IGP, labels = numLabels2[-c(1:2)], col = presColor[-c(1:2)]);
addGrid();
box = par("usr");
legend(x = box[2], y = (box[4]-box[3])/2 + box[3], xjust = 1, yjust = 0.5,
          legend = c("Preserved", "Non-preserved"), fill = c(2,1), cex = 1 )
plot(msx, cr$p.value, col = colors, pch = 20,
     main = "D. Half-preserved\nIGP permutation p-value",
     cex = 2,
     ylab = "p-value", type = "n", xlab = "Module size",
     cex.main = 1)
text(msx, cr$p.value, labels = numLabels2[-c(1:2)], col = presColor[-c(1:2)]);
addGrid();
# If plotting into a file, close it. An un-closed pdf file is not readable.
dev.off();
```

The result is shown in Figure 3. One can say that observed IGP can distinguish preserved and non-preserved modules with the exception of modules 1 (preserved, IGP = 0.31) and 10 (nonpreserved, IGP = 0.32).

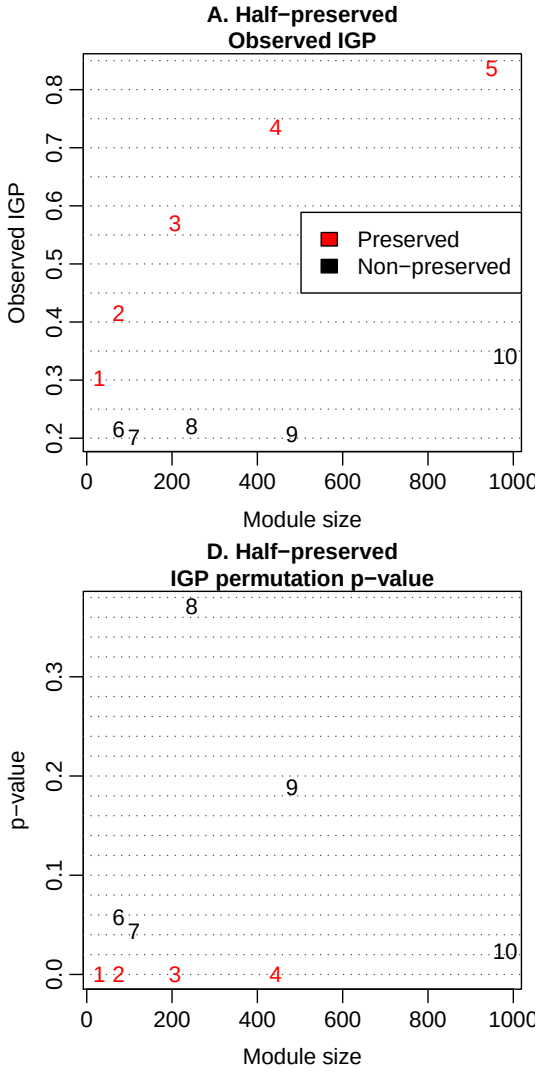

Figure 3: Observed IGP and permutation p-value of simulated modules in this study calculated by `clusterRepro`. Modules are labeled by their numeric labels; red color denotes preserved and black non-preserved modules. In this study IGP distinguishes the preserved and non-preserved modules (except modules 1 and 10) if one chooses a threshold of 0.3. Because only a small number (typically around 20) of the executed permutations are actually used for the p-value calculation, the confidence intervals of the p-values are rather large. In the case of module 5, none of the 1000 permutations was used and hence the p-value is missing.

References

[1] P Langfelder, R Luo, M.C. Oldham, and S Horvath. Is my network module preserved and reproducible? 2010.