Statistical analysis code for analysis of CASTxB6 F2 mouse cross

4. Preservation of CASTxB6 liver modules in male CASTxB6 and other crosses

Peter Langfelder

March 29, 2011

Contents

1 Overview and R session basics

In this document we detail the calculation of network preservation statistics between the CASTxB6 liver female data on the one hand and the corresponding male data and data from other crosses on the other hand. We use the WGCNA package [1] to calculate the network preservation statistics. Since data from other crosses are measured on different microarray platforms, we first pre-process all data by collapsing probe-level data into gene-level data using the WGCNA function `collapseRows`. We then match genes using the Entrez codes that we find to be a better matching mechanism than gene symbols. We then perform rudimentary outlier sample identification using hierarchical clustering of microarray samples, and lastly we attempt to remove possible batch effects by clustering the retained samples again, running automatic cluster identification, and finally using the ComBat (Combining Batches) function [2] to adjust for differences between the identified clusters.
We then calculate the module preservation statistics and analyze the results. In the main article we report the results obtained from the unadjusted data but we include the batch analysis in this report for reference and for the benefit of interested readers.
Because some of the calculations described here can take a substantial amount of time, we regularly save their results. Once a result has been saved, the corresponding part of the code need not be re-run; instead, the result can be loaded from disk. In particular, in Section 3 we re-load or re-create all results of the previous sections.

1.a Setting up the R session

The first step of the analysis is to set the working directory. We recommend to use a subdirectory (folder) within the directory in which the network calculations (part 2) were performed.

```
# Set working directory. This step is necessary if your data is saved in a directory other than the current
# directory. Replace the path name below with the directory where the data is stored on your drive.
# setwd("Z:/home/plangfelder/Work/Mouse-ReciprocalCXB/CxBOnly/NetworkAnalysis/Preservation-LiverInAdipose");
# Load the WGCNA library
library(WGCNA)
# Source custom functions that facilitate some of the analysis steps. Adjust the path/file name if
# necessary.
source("../customFunctions-CastxB6.R");
# This setting is important, do not leave out
options(stringsAsFactors = FALSE);
options(width = 109)
set.seed(1); #needed for .Random.seed to be defined
```

2 Preprocessing data from other crosses

2.a Preliminaries

We now set up a few basic variables and load the preprocessed expression data as well as module labels. Liver and adipose will be indexed 1 and 2, respectively. The files necessary for this step have been generated in parts 1 and 2 of the analysis. Adjust file paths below to point correctly to the necessary files.

```
# Set up and load the reference CASTxB6 data.
nTissues = 2;
setLabels = c("Female Liver", "Female Adipose");
shortLabels = c("Liver", "Adipose");
shortshortLabels = c("L", "A");
tissueLabels = c("Liver", "Adipose");
setLabelsX = c("CxB_Female_Liver", "CxB_Female_Adipose");
# Input files. Change the paths if necessary.
files = c("../../CxBOnly-Liver-outliersRemoved-exprFemaOR-pValFemaOR.RData",
          "../../CxBOnly-Adipose-outliersRemoved-exprFemaOR-pValFemaOR.RData");
express = list();
# Load and store the CASTxB6 expression data
for (tis in 1:nTissues)
{
  x = load(file = files[tis]);
  express[[tis]] = list(data = exprFemaOR);
}
expr = express;
rm(express);
exprSize = checkSets(expr);
nSamples = exprSize$nSamples;
collectGarbage();
# Load the module labels
load(file = "../Female-LA-labels-MEs-ordMEs-afterCleaning.RData")
colors = lapply(labels, labels2colors)
collectGarbage();
```

We now load probe annotation files for all data sets. Again, please adjust the path below to point to the correct directory that contains the expression data files.

```
# Prepare file names
dir = "../../../../Data-AllMouse/";
annotationFiles = c("ApoE_GeneAnnotation.csv.bz2", "BXHwt_GeneAnnotation.csv.bz2",
                    "affyMouse430aAnnotation.csv.bz2",
                    "CXB_GeneAnnotation.csv.bz2");
# This index refers to the data sets defined below
annotationIndex = c(4,2,2,2,2,2,2,3);
cxbAnnot = 4;
allAnnotIndex =c(cxbAnnot, annotationIndex);
# Load annotation data
annot = list();
nAnnot = length(annotationFiles);
for (a in 1:nAnnot)
  annot[[a]] = read.csv(bzfile(spaste(dir, annotationFiles[a])));
collectGarbage();
# How many CASTxB6 probes are present in the BxH data and how many are not?
table(annot[[1]]$substanceBXH %in% annot[[2]]$sequence)
```

Next we set up expression data of all other crosses and genders. As before, please check that the paths (folder or directory names) are set correctly and change them if necessary.

```
files = list( c("../../CxBOnly-Liver-outliersRemoved-exprMaleOR-pValMaleOR.RData",
             spaste(dir, c( "ApoE_Liver_F2_Female_mlratio.csv.bz2",
                            "ApoE_Liver_F2_Male_mlratio.csv.bz2",
                            "BXHwt_Liver_F2_BXH_V2_Female_mlratio.csv.bz2",
                            "BXHwt_Liver_F2_BXH_V2_Male_mlratio.csv.bz2",
                            "HXBwt_Liver_F2_HXB_V1_Female_mlratio.csv.bz2",
                            "HXBwt_Liver_F2_HXB_V1_Male_mlratio.csv.bz2",
                            "MDPexpression.csv.bz2"))),
              c("../../CxBOnly-Adipose-outliersRemoved-exprMaleOR-pValMaleOR.RData",
               spaste(dir, c("ApoE_Adipose_F2_Female_mlratio.csv.bz2",
                             "ApoE_Adipose_F2_Male_mlratio.csv.bz2",
                             "BXHwt_Adipose_F2_BXH_V2_Female_mlratio.csv.bz2",
                             "BXHwt_Adipose_F2_BXH_V2_Male_mlratio.csv.bz2",
                             "HXBwt_Adipose_F2_HXB_V1_Female_mlratio.csv.bz2",
                             "HXBwt_Adipose_F2_HXB_V1_Male_mlratio.csv.bz2"))));
testSetNames = list( c("CxB_Liver_Male",
                "ApoE_Liver_Female",
                "ApoE_Liver_Male",
                "BXHwt_Liver_F2_BXH_V2_Female",
                "BXHwt_Liver_F2_BXH_V2_Male",
                "HXBwt_Liver_F2_HXB_V1_Female",
                "HXBwt_Liver_F2_HXB_V1_Male",
                "MDP"),
               c("CxB_Adipose_Male",
                 "ApoE_Adipose_F2_Female",
                 "ApoE_Adipose_F2_Male",
                 "BXHwt_Adipose_F2_BXH_V2_Female",
                 "BXHwt_Adipose_F2_BXH_V2_Male",
                 "HXBwt_Adipose_F2_HXB_V1_Female",
                 "HXBwt_Adipose_F2_HXB_V1_Male"));
preprocessed = c(TRUE, rep(FALSE, 10));
nTestSets = c(8, 7);
```

2.b Loading test set data and creating gene-level expression data

We next load the expression data for all data sets and use the WGCNA function `collapseRows` to turn probe-level data into gene-level data. We also save information about which probe set was used the representative for a given gene in each data set. This step can take several hours to complete. At the end we save the results so this block need not be re-run when re-running the later parts.

```
representatives = list();
# First collapse CastxB6 data
system.time({
  geneExpr = list();
  for (tis in 1:nTissues)
  {
    printFlush(tissueLabels[tis]);
    representatives[[tis]] = list();
    probes = colnames(expr[[tis]]$data);
    p2g = match(probes, annot[[cxbAnnot]]$sequence);
    fin = is.finite(p2g);
    genes = annot[[cxbAnnot]]$gene_symbol[p2g[fin]];
    LLID = annot[[cxbAnnot]]$LocusLinkID[p2g[fin]];
    cr = collapseRows(t(expr[[tis]]$data[, fin]), genes, probes[fin],
                      method = "MaxMean")
    representatives[[tis]] [[1]] = cbind( cr$group2row, LLID = LLID[ match(cr$group2row[, 1], genes)]);
    geneExpr[[tis]] = list(data = as.matrix(t(cr$datETcollapsed)))
    collectGarbage();
  }
});
# Load other data sets
probeCol = c(rep(6,7), 1);
annotProbeCol = c(4,3,1,4);
rawExpr = list();
for (tis in 1:nTissues)
{
  print(paste("Working on tissue", shortLabels[tis]))
  rawExpr[[tis]] = list();
  for (ts in 1:nTestSets[tis])
  {
    print(paste("Working on test set", ts));
    if (preprocessed[ts])
    {
      x = load(files[[tis]][ts]);
      e = exprMaleOR;
      probes = colnames(e);
    } else {
      tmp = read.csv(bzfile(files[[tis]] [ts] ))
      probes = tmp[, probeCol[ts]];
      if (ts <=7)
      {
        e = as.matrix(t(tmp[, substring(colnames(tmp), 1, 3)=="F2_"]));
      } else
        e = as.matrix(t(tmp[, -1]));
    }
    a = annotationIndex[ts];
    p2g = match(probes, annot[[a]][, annotProbeCol[a]]);
    fin = is.finite(p2g);
    print(table(fin));
    genes = annot[[a]]$gene_symbol[p2g[fin]];
    LLID = annot[[a]]$LocusLinkID[p2g[fin]];
    cr = collapseRows(t(e[, fin]), genes, probes[fin], method = "MaxMean");
```

```
    representatives[[tis]] [[ts+1]] = cbind( cr$group2row,
                                      LLID = LLID[ match(cr$group2row[, 1], genes)]);
    rawExpr[[tis]][[ts]] = list(data = as.matrix(t(cr$datETcollapsed)));
    collectGarbage();
  }
}
save(geneExpr, rawExpr, file = "preservationInOtherXs-geneExpr-rawExpr.RData");
save(representatives, file = "preservationInOtherXs-representativeProbes.RData");
```

2.c Outlier removal

Next we proceed with rudimentary outlier removal. We cluster samples in each data set using average-linkage hierarchical clustering and cut the trees at a constant height.

```
# Use hierarchical clustering to construct the trees
dists = list();
trees = list();
for (tis in 1:nTissues)
{
  print(paste("Working on tissue", shortLabels[tis]))
  dists[[tis]] = list();
  trees[[tis]] = list();
  for (ts in 1:nTestSets[tis])
  {
    printFlush(paste(" ..Working on test set", ts));
    dists[[tis]][[ts]] = dist(rawExpr[[tis]][[ts]]$data);
    trees[[tis]][[ts]] = flashClust(dists[[tis]][[ts]], method = "a");
  }
}
# sizeGrWindow(12,9)
# No outier removal for CxB Males anymore, so very high cut height
cutHeights = list(c(100000, 19, 19, 19, 19, 19, 19, 48), c(100000, 25, 25, 25, 25, 25, 25))
# Plot the trees into a single pdf file for examination
pdf(file = "Plots/preservationInOtherXs-sampleClusterings.pdf", w=20, h=12);
for (tis in 1:nTissues)
{
  for (ts in 1:nTestSets[tis])
  {
    plot(trees[[tis]][[ts]], main = files[[tis]][ts],
         xlab ="", sub="");
    abline(h = cutHeights[[tis]][ts], col = "red");
  }
}
dev.off();
# Cut the trees and form variable multiExpr to hold the outlier-removed data
multiExpr = list();
for (tis in 1:nTissues)
{
  multiExpr[[tis]] = list();
  multiExpr[[tis]][[1]] = geneExpr[[tis]];
  for (ts in 1:nTestSets[tis])
  {
    labs = cutreeStatic(trees[[tis]][[ts]], cutHeight = cutHeights[[tis]][ts]);
    keep = labs==1;
    print(table(keep))
    multiExpr[[tis]][[ts+1]] = list(data = rawExpr[[tis]][[ts]]$data[keep, ]);
  }
  names(multiExpr[[tis]]) = c(setLabelsX[tis], testSetNames[[tis]])
```

```
}
# Save the results for future use
save(multiExpr, file = "preservationInOtherXs-multiExpr.RData");
```

2.d Module assignment for genes

Next we create module labels for genes. If a gene is represented by a single probe, the module label of the probe is taken as the module label of the gene. If a gene is represented by multiple probes and only one of them is assigned to a module, the gene is deemed to also belong to that module. If a gene is represented by multiple probes and several of them are assigned to a module, the gene will be called unassigned if the probes are assigned to different modules. If the probes are assigned all to the same module, so will be the gene.

```
# This function is needed for older versions of R
spread = function(x)
{
  if (sum(is.finite(x)) < 2)
  {
    0;
  } else {
    max(x, na.rm = TRUE) - min(x, na.rm = TRUE);
  }
}
# Create gene labels
geneLabels = list();
for (t in 1:nTissues)
{
  geneLabels[[t]] = list();
  probes = colnames(expr[[t]]$data);
  p2g = match(probes, annot[[cxbAnnot]]$sequence);
  fin = is.finite(p2g);
  genes = annot[[cxbAnnot]]$gene_symbol[p2g[fin]];
  labsX = labels[[t]];
  labsX[labsX==0] = NA;
  mean = tapply(labsX, genes, mean, na.rm = TRUE);
  mean[is.na(mean)] = 0;
  len = tapply(!is.na(labsX), genes, sum);
  #var = tapply(labsX, genes, var, na.rm = TRUE);
  var = tapply(labsX, genes, spread);
  var[len < 2] = 0;
  var[is.na(var)] = 0;
  mean[var!=0] = 0;
  geneLabels[[t]] = as.numeric(mean);
}
names(geneLabels) = setLabelsX;
save(geneLabels, file = "preservationInOtherXs-geneLabels.RData");
```

We now create yet another expression variable in which the columns are labeled by the Entrez ID (Locus Link ID).

```
nSets = nTestSets + 1;
commonGenes = colnames(multiExpr[[1]][[1]]$data);
ai = allAnnotIndex[1];
commonLLIDs = annot[[ ai ]]$LocusLinkID[ match(commonGenes, annot[[ ai ]]$gene_symbol)]
# Note: not all genes have a LLID, but those that do not are mostly Riken genes and just probe names.
multiExprCG = list(); # Common genes
for (t in 1:nTissues)
{
  multiExprCG[[t]] = list();
  for (set in 1:nSets[t])
```

```
  {
    ai = allAnnotIndex[set];
    setGenes = colnames(multiExpr[[t]][[set]]$data);
    setLLIDs = annot[[ ai ]]$LocusLinkID[ match(setGenes, annot[[ ai ]]$gene_symbol)]
    nNA = sum(is.na(setLLIDs));
    commonGenes = intersect(commonGenes, setGenes);
    commonLLIDs = intersect(commonLLIDs, setLLIDs);
    setLLIDs[is.na(setLLIDs)] = spaste("NA.", c(1:nNA));
    multiExprCG[[t]][[set]] = list(data = multiExpr[[t]][[set]]$data);
    colnames(multiExprCG[[t]][[set]]$data) = setLLIDs;
  }
}
collectGarbage();
save(multiExprCG, commonLLIDs, file = "preservationInOtherXs-multiExprCG.RData");
```

2.e Adjusting for possible batch effects

Some of the sample clustering dendrograms generated in the previous section show various degrees of branch structure (distinct branches grouping multiple samples) that is sometimes seen when the microarray data set contains samples processed at different times and/or by different laboratories. Although this should not be the case with our test data sets, this possibility cannot be completely ruled out. Since we do not have any batch information for the samples in the test data sets, we use a simple unsupervised technique to identify possible batches. We identify sample modules using the Dynamic Tree Cut algorithm and equate batches with the sample modules. Then we adjust for the batch effects using the ComBat R function [2].

```
# Cluster expression data in each tissue and cross/sex combination
dists = list();
trees = list();
clusters = list();
collectGarbage();
for (tis in 1:nTissues)
{
   print(paste("Working on tissue", shortLabels[tis]))
   dists[[tis]] = list();
   trees[[tis]] = list();
   clusters[[tis]] = list();
   for (ts in 1:length(multiExprCG[[tis]]))
   {
     printFlush(paste(" ..Working on test set", ts));
     dists[[tis]][[ts]] = dist(multiExprCG[[tis]][[ts]]$data);
     trees[[tis]][[ts]] = flashClust(dists[[tis]][[ts]], method = "a");
   }
 }
collectGarbage();
# Cut the trees using dynamicTreeCut and deepSplit settings of 0 and 1
deepSplits = c(0,1);
nCutParameters = length(deepSplits);
clusters = list();
for (cut in 1:nCutParameters)
{
  clusters[[cut]] = list();
  for (tis in 1:nTissues)
  {
    print(paste("Working on tissue", shortLabels[set]))
    clusters[[cut]][[tis]] = list();
    for (ts in 1:length(multiExpr[[tis]]))
    {
      printFlush(paste(" ..Working on test set", ts));
```

```
    cutHeight = 1.01 * max(trees[[tis]][[ts]]$height);
    clusters[[cut]][[tis]][[ts]] = cutreeDynamic(trees[[tis]][[ts]],
                                distM = as.matrix(dists[[tis]][[ts]]),
                                cutHeight = cutHeight,
                                maxDistToLabel = 4*cutHeight,
                                #minGap = 0.4, maxCoreScatter = 0.8,
                                deepSplit = deepSplits[cut], minClusterSize = 10);
  }
 }
}
collectGarbage();
# This code prints out numbers of samples in each cluster of all clusterings
lapply(clusters, lapply, lapply, table)
# Save results for future use
save(clusters, file = "preservationInOtherXs-sampleClusters.RData");
```

We now apply the `ComBat` function using the identified clusters as batches.

```
nCuts = nCutParameters + 1 # Add one entry for non-combat'ed data
multiExprCb = list();
for (cut in 1:nCuts)
{
 multiExprCb[[cut]] = list();
 for (t in 1:nTissues)
 {
  multiExprCb[[cut]][[t]] = list();
  for (set in 1:nSets[t])
  {
   printFlush(spaste("Working on cut ", cut, ", tissue ", tissueLabels[t], ", set ",
                     names(multiExpr[[t]])[set]));
   datExpr = multiExprCG[[t]][[set]]$data
   if (cut > 1) { nBatches = length(unique(clusters[[cut-1]][[t]][[set]])); } else nBatches = 1;
   if (cut>1 && nBatches > 1)
   {
    set.seed(10);
    datExpr = ComBat(datExpr,
                     sampleInfo = data.frame(SampleID = rownames(datExpr),
                                             Batch = clusters[[cut-1]][[t]][[set]]));
   }
   multiExprCb[[cut]][[t]][[set]] = list(data = datExpr);
  }
 }
}
collectGarbage();
# Fix gene names across cuts
for (t in 1:nTissues)
 for (set in 1:nSets[t])
 {
  geneNames = colnames(multiExprCb[[1]][[t]][[set]]$data);
  for (cut in 2:nCuts)
  {
   colnames(multiExprCb[[cut]][[t]][[set]]$data) = geneNames;
  }
 }
save(multiExprCb, file = "preservationInOtherXs-multiExprCb.RData");
```

This concludes the pre-processing steps. We proceed with calculating network module preservation statistics.

3 Calculation of network module preservation statistics

In this section we detail the calculation of network module preservation statistics. Since the previous section can take a substantial amount of time to run, we will assume that the results have not necessarily been calculated in the same R session; we set up a new R session from scratch. However, this code will do no harm to an already existing R session, so the analysis can also flow seamlessly between the previous section and the current one.
The first step of the analysis is to set the working directory. Please make sure the working directory set here is the same as at the beginning of this document.

```
# Set working directory. This step is necessary if your data is saved in a directory other than the current
directory. Replace the path name below with the directory where the data is stored on your drive.
# setwd("Z:/home/plangfelder/Work/Mouse-ReciprocalCXB/CxBOnly/NetworkAnalysis/Preservation-LiverInAdipose");
# Load the WGCNA library
library(WGCNA)
# Source custom functions that facilitate some of the analysis steps. Adjust the path/file name if
# necessary.
source("../customFunctions-CastxB6.R");
# This setting is important, do not leave out
options(stringsAsFactors = FALSE);
options(width = 109)
set.seed(1); #needed for .Random.seed to be defined
```

Next we set up data set names and load the necessary results of the previous section.

```
load("preservationInOtherXs-multiExprCb.RData");
testSetNames = list( c("CASTxB6 Liver Male",
                "BxH ApoE Liver Female",
                "BxH ApoE Liver Male",
                "BxH Liver Female",
                "BXH Liver Male",
                "HxB Liver Female",
                "HxB Liver Male",
                "MDP"),
               c("CASTxB6 Adipose Male",
                 "BxH ApoE Adipose Female",
                 "BxH ApoE Adipose Male",
                 "BxH Adipose Female",
                 "BxH Adipose Male",
                 "HxB Adipose Female",
                 "HxB Adipose Male"));
nTestSets = c(8,7);
nSets = nTestSets + 1;
refSetNames = c("CASTxB6 Liver Female", "CASTxB6 Adipose Female");
tissueLabels = c("Liver", "Adipose");
oppositeTissueNames = refSetNames[2:1];
# Set up multi-color
load("preservationInOtherXs-geneLabels.RData");
nCuts = length(multiExprCb);
nTissues = length(geneLabels);
```

Next we make sure all data sets are labeled as they should in the expression and module label variables.

```
# Give names to the variable containing module labels
multiColor = list();
for (t in 1:nTissues)
{
  multiColor[[t]] = list(geneLabels[[t]]);
  names(multiColor[[t]]) = refSetNames[t];
}
# Names for the expression data sets
```

```
for (cut in 1:nCuts)
  for (t in 1:nTissues)
    names(multiExprCb[[cut]][[t]]) = c(refSetNames[t], testSetNames[[t]]);
# Several other versions of set names
setNames = list();
allSetNames = list(); # This one is for the combined mpAll
nAllSets = nSets;
for (t in 1:nTissues)
{
  setNames[[t]] = c(refSetNames[t], testSetNames[[t]]);
  allSetNames[[t]] = c(setNames[[t]], oppositeTissueNames[t]);
  nAllSets[t] = nSets[t] + 1;
}
```

We now call the function `modulePreservation` that calculates network module preservation statistics. The code is written so that it will run in two threads if the package `multicore` is installed and the variable `runParallel` is set `TRUE`. The parallel run can save some time on multi-core systems. Regardless of whether this code runs in 1 or 2 threads, it may take up to a few days to complete. The calculation can be speeded up by replacing the line `nCalculatedCuts=nCuts` immediately below by `nCalculatedCuts=1`. This speeds up the calculation at the expense of restricting the analysis to original data without considering possible batch effects. However, in our main article we do not consider the batch effects either, so this omission is not detrimental. We save the result so this part needs only be run once.

```
nCalculatedCuts = nCuts
mp = list();
# Function that calculates preservation in one tissue parametrized by t
tissueRun = function(t)
{
  printFlush(paste("Working on cut", cut, ", tissue", tissueLabels[t]));
  permStatFile = spaste("preservation-permutedStats-cut-", cut, "-t-", t, ".RData");
  mp = modulePreservation(multiExprCb[[cut]][[t]],
                                  multiColor[[t]],
                                  randomSeed = t * 384 + 37,
                                  referenceNetworks = 1,
                                  nPermutations = 100,
                                  networkType = "signed hybrid",
                                  corFnc = "bicor",
                                  quickCor = 0,
                                  greyName = 0,
                                  permutedStatisticsFile = permStatFile,
                                   verbose = 3);
  save(mp, file = spaste("modulePreservation-mp-cut-", cut, "-t-", t, ".RData"));
  mp;
}
# Edit this seting to either run the two tissues
runParallel = TRUE;
if (!require(multicore)) runParallel = FALSE;
# Main code
for (cut in 1:nCalculatedCuts)
{
  mp[[cut]] = list();
  proc = list();
  if (runParallel)
  {
    for (t in 1:nTissues)
      proc[[t]] = parallel(tissueRun(t));
    res = collect(proc);
  } else {
    res = list();
```

```
    for (t in 1:nTissues)
      res[[t]] = tissueRun(t);
  }
  mp[[cut]] = res;
  save(mp, file = spaste("modulePreservation-mp-cut-", cut, ".RData"));
}
# Save the results
save(mp, file = "modulePreservation-mp.RData");
```

If the results have been calculated previously, they can be loaded using

```
load(file = "modulePreservation-mp.RData");
```

4 Analysis of summary preservation statistics

We now load the results of liver – adipose preservation and combine them the results of the current analysis for easier analysis and plotting. In our main article we only discuss the expression data without considering possible batch effects, so we restrict the analysis to `cut=1`.

```
cut = 1;
mp1 = mp;
load("../Preservation-LiverInAdipose/modulePreservation-mp.RData");
# Combine the results into one
mpAll = list();
for (t in 1:nTissues)
{
  mpAll[[t]] = list(observed = mp1 [[cut]] [[t]] $ preservation$observed,
                    Z = mp1 [[cut]] [[t]] $ preservation$Z);
  mpAll[[t]]$observed[[1]] [[nSets[t] + 1]] = mp$preservation$observed[[t]] [[3-t]];
  mpAll[[t]]$Z[[1]] [[nSets[t] + 1]] = mp$preservation$Z[[t]] [[3-t]];
}
excludeMods = c(0, 0.1);
```

We next create module labels by the highest enriched GO term. Modify the path/file names to point to the enrichment tables generated in Part 2 (Network Analysis).

```
goAnnotFiles = c("../CxBOnly-Female-Liver-GOenrichment.txt", "../CxBOnly-Female-Adipose-GOenrichment.txt");
goLabels = list();
goModules = list();
goPvalue = list();
for (t in 1:nTissues)
{
  goAnn = read.delim(goAnnotFiles[t], header = TRUE);
  nModules = length(unique(goAnn$module));
  best = tapply(c(1:nrow(goAnn)), goAnn$module, min);
  goModules[[t]] = goAnn$module[best][-1];
  goLabels[[t]] = spaste(goModules[[t]], ": ", goAnn$termName[best][-1]);
  goPvalue[[t]] = goAnn$BonferoniP[best][-1];
  goLabels[[t]] [goPvalue[[t]] > 1e-4] = goModules[[t]] [goPvalue[[t]] > 1e-4];
}
collectGarbage();
formGoLabels = lapply(goLabels, fixLabels, maxCharPerLine = 14);
```

We also prepare a list of HDL-related modules to be labeled by a special color (red).

```
# Prepare a list of HDL-related modules
hdlModules = list();
for (t in 1:nTissues)
{
```

```
  tab = read.csv(file = spaste("../CxBOnly-", tissueLabels[t], "-HDLBestModules.csv"), header = TRUE);
  hdlModules[[t]] = as.numeric(substring(tab$ME, 3));
}
hdlColor = "red"
```

We now plot the main result which is also included in the main article as Figure xx. We plot the preservation $Z_{summary}$ as an "array of barplots" in which each barplot corresponds to one test data set. We use a custom function `labeledBarplotArray` which is analogous to the `labeledHeatmap` function in the WGCNA package. We plot the results directly into a pdf file.

```
# Open a suitable pdf file for plotting
pdf(file="Plots/modulePreservation-barplotArrays-%02d.pdf", onefile = FALSE, wi=10, he=8);
# Loop over tissues
for (t in 1:nTissues)
{
  #Prepare data and labels for plotting
  colOrder = c(nAllSets[t]-1, 1:(nAllSets[t]-2));
  textMat = signif(allSetStats [[t]] $ Z, 2);
  dim(textMat) = dim(allSetStats [[t]] $ mr);
  numMat = allSetStats [[t]] $ Z;
  # Restrict the bars to be between 0 and 20 units long (actual value will be displayed in text)
  numMat[numMat < 0] = 0;
  numMat[numMat > 20] = 20;
  mods = rownames(allSetStats [[t]] $ Z);
  order = order(as.numeric(mods));
  colors = spaste("ME", labels2colors(as.numeric(mods)))[order];
  par(mar = c(7, 14, 3, 2));
  labeledBarplotArray(mat = numMat[order, colOrder],
                ablines = c(2, 10),
                abColors = c("blue", "darkgreen"),
                ablines.lty = c(2,2),
                alignRight = FALSE,
                barGap = 0.20,
                yLabels = colors,
                ySymbols = altLabels[[t]][order],
                xLabels = allSetNames[[t]][-1][colOrder],
                colors = greenWhiteRed(100)[50:100],
                main = spaste("Preservation of CASTxB6 ", tissueLabels[t], " modules in ",
                            tissueLabels[3-t], " tissue and other crosses" ),
                cex.lab = 0.8,
                textMat = textMat[order, colOrder], cex.text = 0.7,
                yColorWidth = 0.02,
                textMat.minXPosition = 10,
                colors.lab.y = labelColors[[t]][order],
                commonScale = TRUE);
}
# Close the plottong files
dev.off();
```

The results are shown in Figures 1 and 2. We observe that several modules (2, 5, 7) show strong preservation across all test data sets. Several other modules, including the HDL-related module 6, show moderate to strong preservation across all data sets.

References

[1] Peter Langfelder and Steve Horvath. WGCNA: an R package for weighted correlation network analysis. *BMC Bioinformatics*, 9(1):559, 2008.

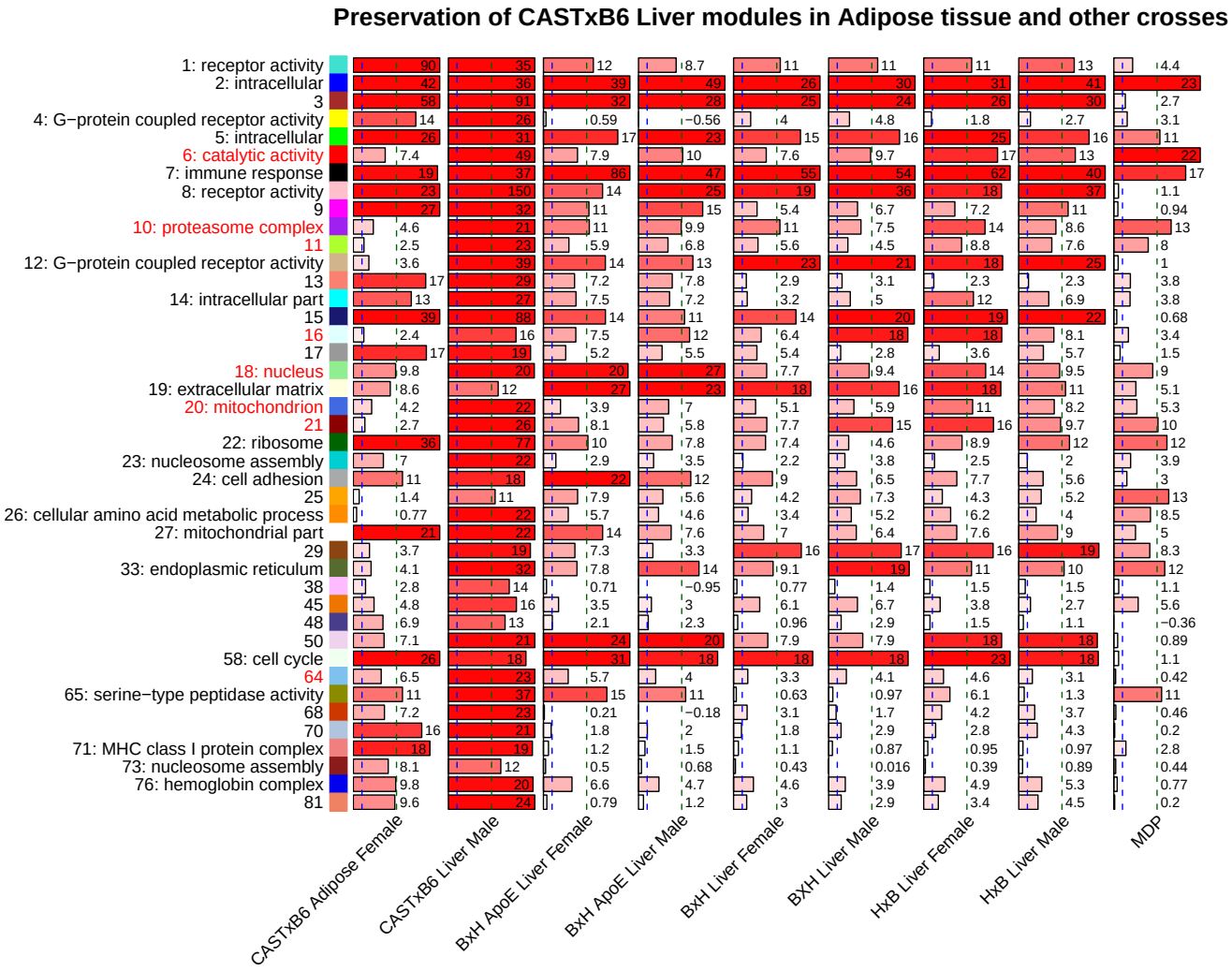

Figure 1: Summary preservation scores of CASTxB6 female liver modules (rows) in liver expression data from adipose female CASTxB6, liver male CASTxB6, and from liver tissues of other crosses. The test data sets (columns) are indicated at the bottom. Dashed lines indicate thresholds of $Z_{summary} = 2$ (weak preservation) and $Z_{summary} = 10$ (strong preservation). Modules are labeled by their numeric label, GO label (where the enrichment is strong), and color. HDL-related modules are indicated red text labels.

[2] Cheng Li and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

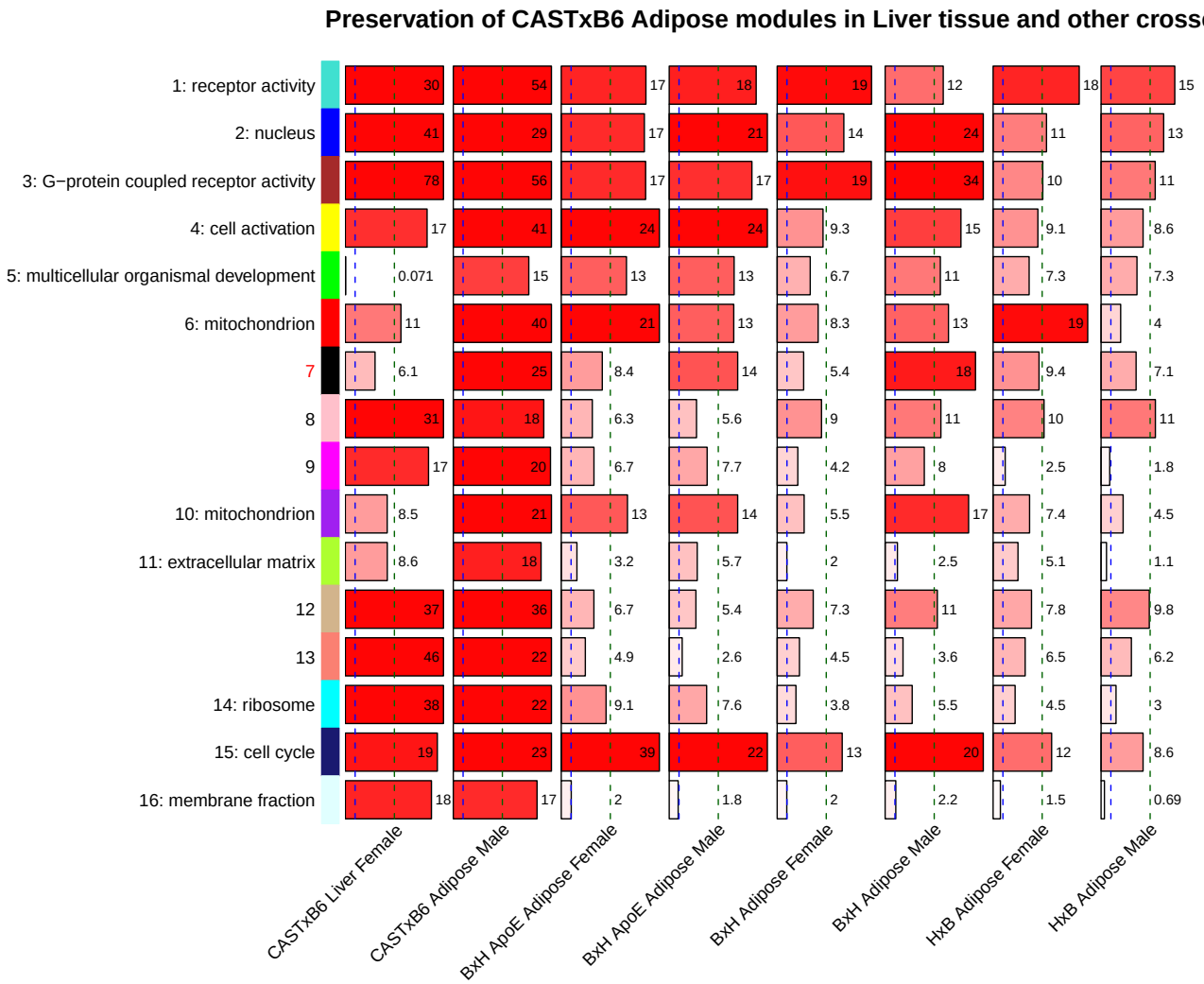

Figure 2: Summary preservation scores of CASTxB6 female **adipose** modules (rows) in liver expression data from liver female CASTxB6, adipose male CASTxB6, and from adipose tissues of other crosses. The test data sets (columns) are indicated at the bottom. Dashed lines indicate thresholds of $Z_{summary} = 2$ (weak preservation) and $Z_{summary} = 10$ (strong preservation). Modules are labeled by their numeric label, GO label (where the enrichment is strong), and color. HDL-related modules are indicated red text labels.