

Expression and network analysis of Illumina microarray data

Giovanni Coppola

UCLA

Kellen Winden

UCLA

Genevieve Konopka

UCLA

Fuying Gao

UCLA

Daniel Geschwind

UCLA

Method Article

Keywords: micorarrays, network analysis, functional genomics

Posted Date: November 13th, 2009

DOI: <https://doi.org/10.1038/nprot.2009.215>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Abstract

Introduction

Microarrays are commonly used tools for assessing changes in gene expression between two populations. Many algorithms exist for analyzing the hybridization signals from these arrays. Here, we present our code for assessing changes in expression from samples hybridized to Illumina whole genome microarrays. We also include our code for building weighted gene coexpression networks. Finally, we present how we determined the overlap from two independent microarray studies.

Equipment

1. Requires a personal computer running R/Bioconductor.

Procedure

```
1. ###This code is for analyzing the SH-SY5Y data from Illumina Human Ref8-v2 arrays using R ###The
following abbreviations are used: V= control cells; WT= cells overexpressing human FOXP2; M12= cells
overexpressing chimp FOXP2 Load packages library \ (Biobase) library \ (marray) library \ (limma) library \
(RColorBrewer) library \ (MASS) library \ (gplots) ###Read in the dataset ###samples not included in this
paper are excluded load \ ("line1367_noM125_justdata.R") mydata<-mydata[,-c \ (13:24)] #excluding line3
detsco<-detsco[,-c \ (13:24)] #excluding line3 sds<-sds[,-c \ (13:24)] #excluding line3 beads<-beads[,-c \
(13:24)] #excluding line3 Samples<-Samples[,-c \ (13:24),] #excluding line3 matriz <- as.matrix \ (mydata)
ROWlabels<-as.character \ (paste \ (Samples$Condition, Samples$CellLine, Samples$Replicate, sep=" _"))
phD <- Samples[,-c \ (1,3,4,5)] rownames \ (phD)<-ROWlabels metadata<-data.frame \ (labelDescription=c \
("Array", "Condition", "Replicate", "CellLine"), row.names=c \ ("Array", "Condition", "Replicate", "cellLine"))
pD<-new \ ("AnnotatedDataFrame",data=phD, varMetadata=metadata) ###Load the Illumina microarray
annotations ill.array <- read.csv \ (file="C:/Documents and Settings/dhglab.NEURODH/Desktop/Chip
info/HumanRef-8_V2_11223162_B.csv", header=T) ###Normalize between arrays matrizQ <-
normalizeBetweenArrays \ (matriz,method="quantile") ###Apply Combat samName<-labs combat.list<-
as.data.frame \ (cbind \ (samName, Samples[,9], Samples[,3])) colnames \ (combat.list)<- c \ ("Sample",
"Batch", "Condition") write.xls \ (matrizQ,file="ComBatExpr.xls", quote=F) write.xls \
(combat.list,file="ComBatList.txt",quote=F) source \ ("C:/Documents and
Settings/dhglab.NEURODH/Desktop/Study/ComBat.R",local=T) ComBat \
("ComBatExpr.xls","ComBatList.txt") ###Create the exprSet object #Read in Combat adjusted data
matrizQ_adj<-read.table \ ("Adjusted_ComBatExpr_1cond.xls", header=T) rownames \ (matrizQ_adj)<-
rownames \ (matriz) matrizQ_adj<-as.matrix \ (matrizQ_adj) eSet <- new \ ("ExpressionSet",
exprs=matrizQ_adj, phenoData=pD) eSet ftdexp<-exprs \ (eSet) ###Compare experimental conditions #all
samples all.samples<-as.data.frame \ (ftdexp) #ratios for heatmap and ratio output #selecting the
coefficients for single arrays #1. Controls all.contr.wt<-all.samples[,Samples$Condition == "WT"]
all.contrM.wt<-rowMeans \ (all.contr.wt) all.contr.v<-all.samples[,Samples$Condition == "V"] all.contrM.v<-
```

```

rowMeans\ (all.contr.v) #2. Exp all.exp.wt<-all.samples\[Samples$Condition == "WT"] all.exp.m12<-
all.samples\[Samples$Condition == "M12"] #3. Ratios all.coef.WTV <-all.exp.wt-all.contrM.v colnames\
(all.coef.WTV)<-paste\("vsV", colnames\ (all.coef.WTV), sep=".") all.coef.M12V<-all.exp.m12-all.contrM.v
colnames\ (all.coef.M12V)<-paste\("vsV", colnames\ (all.coef.M12V), sep=".") all.coef.M12WT<-
all.exp.m12-all.contrM.wt colnames\ (all.coef.M12WT)<-paste\("vsWT", colnames\ (all.coef.M12WT),
sep=".") #export the data mydata2<-cbind\ (rownames\ (all.coef.WTV), all.coef.WTV, all.coef.M12V,
all.coef.M12WT) ill.array.nodup<-ill.array\ [!duplicated\ (ill.array$Target),] my.all.data<- merge\
(mydata2,ill.array.nodup, by.x="Target",by.y="Target") mydata3<-cbind\ (rownames\
(all.samples),all.samples) colnames\ (mydata3)1<-"Target" colnames\ (mydata3)\ [2:length\ (mydata3)]<-
paste\ (colnames\ (mydata3)\ [2:length\ (mydata3)],"exp",sep=".") my.all.data.new<-merge\
(my.all.data,mydata3,by.x="Target",by.y="Target") write.xls\
(my.all.data.new,"Line167/rm_alldata_ratio_line167.xls",quote=F) ###Fit the data to a linear model TS<-
Samples$Condition TS <- factor\ (TS, levels=unique\ (TS)) design <- model.matrix\ (~0+TS) colnames\
(design) <- levels\ (TS) fit<- lmFit\ (eSet, design) cont.anova <- makeContrasts\ (WTvsV.L167= WT - V,
M12vsV.L167= M12 - V, M12vsWT.L167= M12 - WT, levels=design) fit2.anova<- contrasts.fit\ (fit,
cont.anova) fitb<- eBayes\ (fit2.anova) ###Implement a statistical threshold: chosen.adjust<-"none"
chosen.p<0.05 current.contrast<-"contrast" results<-decideTests\
(fit2.anova,adjust.method=chosen.adjust,p=as.numeric\ (chosen.p)) write.fit\
(fitb,file="dummy.xls",adjust=chosen.adjust,results=results) treat.de<-read.table\
(file="dummy.xls",head=T) ###Output the data for comparisons myNames<-names\ (treat.de) res.col<-
which\ (regexpr\ ("Res.",myNames)>0) anovalist<- which\ (apply\ (treat.de\[res.col],1,function\ (x)any\
(x,na.rm=T))) treat.de.anova<-treat.de[anovalist,] fitsel.tre2<-merge\ (treat.de.anova, ill.array.nodup,
by.x="ID",by.y="Target") colnames\ (fitsel.tre2)1<-"Target" fitsel.ratio<-merge\ (fitsel.tre2,my.all.data.new)
colnames\ (fitsel.ratio) myNames <-names\ (fitsel.ratio) #selects the relevant columns for output res.col<-
which\ (regexpr\ ("Res.",myNames)>0) coefs.col <- which\ (regexpr\ ("Coef.",myNames)>0) ts.col<-
coefs.col+length\ (coefs.col) pvals.col <- which\ (regexpr\ ("p.value.",myNames)>0) endcolumns.start<-
length\ (fitsel.ratio)-\ (length\ (mydata2)-2) endcolumns.end<-length\ (fitsel.ratio) fitsel.ratio2<-cbind\ (
Target= fitsel.ratio$Target, Transcript=fitsel.ratio$Accession, Symbol=fitsel.ratio$Symbol,
Definition=fitsel.ratio$Definition, fitsel.ratio\[coefs.col], fitsel.ratio\[pvals.col], F=fitsel.ratio$F,
F.p.value=fitsel.ratio$F.p.value, fitsel.ratio\[res.col], fitsel.ratio\[ts.col],
ProbeSequence=fitsel.ratio$Probe_Sequence, Ontology=fitsel.ratio$Ontology,
Synonym=fitsel.ratio$Synonym, fitsel.ratio\[29: 100] ) fitsel.ratio2<-fitsel.ratio2\[order\
(fitsel.ratio2$F,decreasing=T),] out.file<-paste\ (paste\
(current.contrast,chosen.adjust,chosen.p,sep="_"), "xls",sep=".") write.xls\ (fitsel.ratio2,file=paste\
("rm2BE_covariate_", out.file,sep=""),quote=F) ###Output the complete list of genes fitsel.treAll<-merge\
(treat.de, ill.array, by.x="ID",by.y="Target") colnames\ (fitsel.treAll)1<-"Target" fitsel.ratioAll<-merge\
(fitsel.treAll,my.all.data.new) myNames <-names\ (fitsel.ratio) #selects the relevant columns for output
res.col<- which\ (regexpr\ ("Res.",myNames)>0) coefs.col <- which\ (regexpr\ ("Coef.",myNames)>0) ts.col<-
coefs.col+length\ (coefs.col) pvals.col <- which\ (regexpr\ ("p.value.",myNames)>0) endcolumns.start<-
length\ (fitsel.ratio)-\ (length\ (mydata2)-2) endcolumns.end<-length\ (fitsel.ratio) fitsel.ratioN<-cbind\ (

```

```

Target= fitssel.ratioAll$Target, Transcript=fitssel.ratioAll$Accession, Symbol=fitssel.ratioAll$Symbol,
Definition=fitssel.ratioAll$Definition, fitssel.ratioAll[,coefs.col], fitssel.ratioAll[,pvals.col], F=fitssel.ratioAll$F,
F.p.value=fitssel.ratioAll$F.p.value, fitssel.ratioAll[,res.col], fitssel.ratioAll[,ts.col],
ProbeSequence=fitssel.ratioAll$Probe_Sequence, Ontology=fitssel.ratioAll$Ontology,
Synonym=fitssel.ratioAll$Synonym, fitssel.ratioAll[,29:100] ) fitssel.ratioN<-fitssel.ratioN[!\duplicated\
(fitssel.ratioN$Target),] fitssel.ratioN<-fitssel.ratioN[order\((fitssel.ratioN$F,decreasing=T),] write.xls\
(fitssel.ratioN,file=paste\("rm2BE_covariate_genelist_all.xls",sep="/"),quote=F) 2. ###This code is for
analyzing the brain tissue data from Illumina Human Ref8-v3 arrays using R ###The following
abbreviations are used: FP=frontal pole; caud=caudate nucleus; hipp=hippocampus Load packages
library \((Biobase) library \((marray) library \((limma) library \((RColorBrewer) library \((MASS) library \((gplots)
###Read in the dataset Samples<-read.delim\
(file="E:/Study/Human_Solexa_number2/Human_Solexa_Illumina_targets2.txt", header=T) labs<-
as.character\((paste\((substr\((Samples$Status,1,2), Samples$ConditionS, Samples$Replicate, sep="_"))
pD <- Samples[,c\((1,2,3,5)] rownames\((pD)<-labs metadata<-data.frame\((labelDescription=c\
("ArrayID", "Genotype", "Condition", "Status"), row.names= c\("ArrayID", "Genotype", "Condition", "Status"))
pD<-new\("AnnotatedDataFrame",data=pD, varMetadata=metadata) illumina71<- read.delim \
(file="2008_071A_Sample_probe_profile.txt", header=TRUE) rownames\((illumina71)<-
illumina71$PROBE_ID illumina71<-illumina71[,3:74] illumina43<- read.delim \((file="Sample Probe profile
2008-043.txt", header=TRUE) rownames\((illumina43)<-illumina43$PROBE_ID illumina43<-illumina43\
[,11:12] illumina61<- read.delim \((file="Sample probe profile 2008-061A.txt", header=TRUE) rownames\
(illumina61)<-illumina61$PROBE_ID illumina61<-illumina61[,3:50] illumina2<-cbind\((illumina61,
illumina71) #add 43 later as it has 2 columns #signal values nsampl<-40 colnames\((illumina2)\[seq\((1,\
(nsampl*3),3)] mydata<-illumina2[,seq\((1,\(nsampl*3),3)] mydata2<-cbind\((mydata[,1:16], illumina43\
[,1], mydata[, 17:40]) colnames\((mydata2)<-labs mydata<-mydata2 #detection scores colnames\
(illumina2)\[seq\((1,\(nsampl*3),3)+1] detsco<-illumina2[,seq\((1,\(nsampl*3),3)+1] detsco2<-cbind\
(detsco[,1:16], illumina43[,2], detsco[, 17:40]) detsco<- detsco2 colnames\((detsco)<-labs detsco<-1-
detsco #pvalue to real mydata.notlog<-mydata mydata=log2\((mydata) matriz <- as.matrix\((mydata)
###Load the Illumina microarray annotations ill.array <- read.csv\((file="C:/Documents and
Settings/dhglab.NEURODH/Desktop/Chip info/HumanRef-8_V3_0_R0_11282963_A.csv", header=T)
#remove probes that are not a perfect match to either human or chimp genome bad.probe<-read.table\
("E:/Study/Human_Solexa_number2/Bad_Illumina_probes_for_Chimp_comp.txt", header=T) dim\
(bad.probe) #1 9305 1 dim\((mydata) #1 24526 41 removeProbes = match\((bad.probe$PSID, rownames\
(mydata)) mydata.g<-mydata[,-removeProbes,] detsco.g<-detsco[,-removeProbes,] dim\((mydata.g) #
15221 41 dim\((detsco.g) save\((mydata.g, detsco.g, bad.probe, mydata, file="good_probes_data")
matriz<- as.matrix\((mydata.g) ###Normalize between arrays matrizQQ <-normalizeBetweenArrays\
(matriz,method="quantile") ###Apply Combat samName<-labs combat.list<- as.data.frame\((cbind\
(samName, Samples[,7], Samples[,4], Samples[,5])) colnames\((combat.list)<- c\("SampleName",
"Batch", "Condition") write.xls\((matrizQ,file="ComBatExpr.xls", quote=F) write.xls\
(combat.list,file="ComBatList.txt",quote=F) source\("C:/Documents and
Settings/dhglab.NEURODH/Desktop/Study/ComBat.R",local=T) ComBat\

```

```

("ComBatExpr.xls","ComBatList.txt") ###Create the exprSet object #Read in Combat adjusted data
matrizQ_adj<-read.table\("Adjusted_ComBatExpr.xls", header=T) rownames\ (matrizQ_adj)<- rownames\
(matriz.g) matrizQ_adj<-as.matrix\ (matrizQ_adj) eSet <- new\("ExpressionSet", exprs=matrizQ_adj,
phenoData=pD.g) eSet ftdexp<-exprs\ (eSet) ###Compare experimental conditions #all samples
all.samples<-as.data.frame\ (ftdexp) #1. Controls, each condition as a control all.contr.h <-all.samples\
[Samples$Status=="human" ] all.contrM.h <-rowMeans\ (all.contr.h) all.contr.hh <-all.samples\
[Samples$Status=="human" & Samples$ConditionS=="hipp"] all.contrM.hh <-rowMeans\ (all.contr.hh)
all.contr.hc <-all.samples\ [Samples$Status=="human" & Samples$ConditionS=="caud"] all.contrM.hc <-
rowMeans\ (all.contr.hc) all.contr.hf <-all.samples\ [Samples$Status=="human" &
Samples$ConditionS=="FP"] all.contrM.hf <-rowMeans\ (all.contr.hf) all.contr.cc <-all.samples\
[Samples$Status=="chimp" & Samples$ConditionS=="caud"] all.contrM.cc <-rowMeans\ (all.contr.cc)
all.contr.cf <-all.samples\ [Samples$Status=="chimp" & Samples$ConditionS=="FP"] all.contrM.cf <-
rowMeans\ (all.contr.cf) all.contr.caud <-all.samples\ [Samples$ConditionS=="caud"] all.contrM.caud <-
rowMeans\ (all.contr.caud) all.contr.fp <-all.samples\ [Samples$ConditionS=="FP"] all.contrM.fp <-
rowMeans\ (all.contr.fp) #2. Exp, each condition as a treatment all.exp.c<- all.samples\
[Samples$Status=="chimp"] all.exp.hipp<- all.samples\ [Samples$ConditionS=="hipp"] all.exp.caud<-
all.samples\ [Samples$ConditionS=="caud"] all.exp.hh<- all.samples\ [Samples$Status=="human" &
Samples$ConditionS=="hipp"] all.exp.hc<- all.samples\ [Samples$Status=="human" &
Samples$ConditionS=="caud"] all.exp.ch<- all.samples\ [Samples$Status=="chimp" &
Samples$ConditionS=="hipp"] all.exp.cc<- all.samples\ [Samples$Status=="chimp" &
Samples$ConditionS=="caud"] all.exp.cf<- all.samples\ [Samples$Status=="chimp" &
Samples$ConditionS=="FP"] #3. Ratios all.coef.hhc<- all.exp.hh - all.contrM.hc colnames\ (all.coef.hhc)<-
paste\ ("vsCAUD", colnames\ (all.coef.hhc), sep=".") all.coef.hhf<- all.exp.hh - all.contrM.hf colnames\
(all.coef.hhf)<-paste\ ("vsFP", colnames\ (all.coef.hhf), sep=".") all.coef.hcf<- all.exp.hc - all.contrM.hf
colnames\ (all.coef.hcf)<-paste\ ("vsFP", colnames\ (all.coef.hcf), sep=".") all.coef.chc<- all.exp.ch -
all.contrM.cc colnames\ (all.coef.chc)<-paste\ ("vsCAUD", colnames\ (all.coef.chc), sep=".") all.coef.chf<-
all.exp.ch - all.contrM.cf colnames\ (all.coef.chf)<-paste\ ("vsFP", colnames\ (all.coef.chf), sep=".")
all.coef.ccf<- all.exp.cc - all.contrM.cf colnames\ (all.coef.ccf)<-paste\ ("vsFP", colnames\ (all.coef.ccf),
sep=".") all.coef.ch.hipp<- all.exp.ch - all.contrM.hh colnames\ (all.coef.ch.hipp)<-paste\ ("CHvsHUM",
colnames\ (all.coef.ch.hipp), sep=".") all.coef.ch.caud<- all.exp.cc - all.contrM.hc colnames\
(all.coef.ch.caud)<-paste\ ("CHvsHUM", colnames\ (all.coef.ch.caud), sep=".") all.coef.ch.fp<- all.exp.cf -
all.contrM.hf colnames\ (all.coef.ch.fp)<-paste\ ("CHvsHUM", colnames\ (all.coef.ch.fp), sep=".")
all.coef.hipp.caud<- all.exp.hipp - all.contrM.caud colnames\ (all.coef.hipp.caud)<-paste\ ("HippvsCaud",
colnames\ (all.coef.hipp.caud), sep=".") all.coef.hipp.fp<- all.exp.hipp - all.contrM.fp colnames\
(all.coef.hipp.fp)<-paste\ ("HippvsFP", colnames\ (all.coef.hipp.fp), sep=".") all.coef.caud.fp<- all.exp.caud
- all.contrM.fp colnames\ (all.coef.caud.fp)<-paste\ ("CaudvsFP", colnames\ (all.coef.caud.fp), sep=".")
all.coef.CH<- all.exp.c - all.contrM.h colnames\ (all.coef.CH)<-paste\ ("CHvsHUM", colnames\ (all.coef.CH),
sep=".") #export the data mydata2<-cbind\ (rownames\ (all.coef.hhc), all.coef.hhc, all.coef.hhf,
all.coef.hcf, all.coef.chc, all.coef.chf, all.coef.ccf, all.coef.ch.hipp, all.coef.ch.caud, all.coef.ch.fp,
all.coef.hipp.caud, all.coef.hipp.fp, all.coef.caud.fp, all.coef.CH) colnames\ (mydata2)1<- "Target"

```

```

ill.array.nodup<-ill.array\[!duplicated\((ill.array$Probe_Id),] my.all.data<- merge\((mydata2,ill.array.nodup,
by.x="Target",by.y="Probe_Id") mydata3<-cbind\((rownames\((all.samples),all.samples) colnames\
(mydata3)1<-"Target" colnames\((mydata3)\[2:length\((mydata3)]<- paste\((colnames\((mydata3)\
[2:length\((mydata3)],"exp",sep=".") my.all.data.new<-merge\
(my.all.data,mydata3,by.x="Target",by.y="Target") ratio_exp<-merge\
(mydata2,mydata3,by.x="Target",by.y="Target") write.xls\((my.all.data.new,"alldata_ratio.xls",quote=F)
###Fit the data to a linear model TS<- paste\((Samples$Status, Samples$ConditionS, sep="_") TS <-
factor\((TS, levels=unique\((TS)) design <- model.matrix\((~0+TS) colnames\((design) <- levels\((TS) fit<-
lmFit\((eSet, design) cont.anova <- makeContrasts\((HIPPIvsCAUD_human= human_hipp - human_caud,
HIPPIvsFP_human= human_hipp - human_FP, CAUDvsFP_human= human_caud - human_FP,
HIPPIvsCAUD_chimp= chimp_hipp - chimp_caud, HIPPIvsFP_chimp= chimp_hipp - chimp_FP,
CAUDvsFP_chimp= chimp_caud - chimp_FP, CHvsHUM_hipp= chimp_hipp - human_hipp,
CHvsHUM_caud= chimp_caud - human_caud, CHvsHUM_FP= chimp_FP - human_FP, HIPPIvsCAUD = \
(human_hipp + chimp_hipp)/2 - \((human_caud + chimp_caud)/2, HIPPIvsFP = \((human_hipp +
chimp_hipp)/2 - \((human_FP + chimp_FP)/2, CAUDvsFP = \((human_caud + chimp_caud)/2 - \
(human_FP + chimp_FP)/2, CHIMPvsHUMAN = \((chimp_hipp+ chimp_caud+ chimp_FP)/3 - \
(human_hipp+ human_caud+ human_FP)/3, levels=design) fit2.anova<- contrasts.fit\((fit, cont.anova)
fitb<- eBayes\((fit2.anova) ###Implement a statistical threshold: chosen.adjust<-"none" chosen.p<-0.05
current.contrast<-"contrast" results<-decideTests\((fitb,adjust.method=chosen.adjust,p=as.numeric\
(chosen.p)) write.fit\((fitb,file="dummyrr.xls",adjust=chosen.adjust,results=results) treat.de<-read.table\
(file="dummyrr.xls",head=T) myNames<-names\((treat.de) res.col<- which\((regexpr\("Res.",myNames)>0)
anovalist<- which\((apply\((treat.de[,res.col],1,function\((x)any\((x,na.rm=T))) treat.de.anova<-treat.de\
[anovalist,] ill.array.nodup<-ill.array\[!duplicated\((ill.array$Probe_Id),] fitsel.tre2<-merge\((treat.de.anova,
ill.array.nodup, by.x="ID",by.y="Probe_Id") colnames\((fitsel.tre2)1<-"Probe" fitsel.ratio<-merge\
(fitsel.tre2,ratio_exp, by.x="Probe", by.y="Target") ###Output the data for comparisons myNames <-
names\((fitsel.ratio) #selects the relevant columns for output res.col<- which\((regexpr\
("Res.",myNames)>0) coefs.col <- which\((regexpr\("Coef.",myNames)>0) ts.col<- coefs.col+length\
(coefs.col) pvals.col <- which\((regexpr\("p.value.",myNames)>0) endcolumns.start<-length\((fitsel.ratio)-\
(length\((mydata2)-2) endcolumns.end<-length\((fitsel.ratio) fitsel.ratio2<-cbind\(( Probe= fitsel.ratio$Probe,
Accession=fitsel.ratio$Accession, Symbol=fitsel.ratio$Symbol, Definition=fitsel.ratio$Definition,
fitsel.ratio[,coefs.col], fitsel.ratio[,pvals.col], F=fitsel.ratio$F, F.p.value=fitsel.ratio$F.p.value, fitsel.ratio\
[,res.col], fitsel.ratio[,ts.col], A= fitsel.ratio$A, fitsel.ratio[,84:226], fitsel.ratio[,57:66], fitsel.ratio[,69:77],
fitsel.ratio[,79:83] ) fitsel.ratio2<-fitsel.ratio2\[order\((fitsel.ratio2$F,decreasing=T),] out.file<-paste\((paste\
(current.contrast,chosen.adjust,chosen.p,sep="_"),"xls",sep=".") write.xls\((fitsel.ratio2,
"combat_significant_genelist_0.05.xls", quote=F) ###Output the complete list of genes fitsel.treAll<-
merge\((treat.de, ill.array.nodup, by.x="ID",by.y="Probe_Id") colnames\((fitsel.treAll)1<-"Probe"
fitsel.ratioAll<-merge\((fitsel.treAll, ratio_exp, by.x="Probe", by.y="Target") myNames <-names\
(fitsel.ratioAll) #selects the relevant columns for output res.col<- which\((regexpr\("Res.",myNames)>0)
coefs.col <- which\((regexpr\("Coef.",myNames)>0) ts.col<- coefs.col+length\((coefs.col) pvals.col <-
which\((regexpr\("p.value.",myNames)>0) endcolumns.start<-length\((fitsel.ratio)-\((length\((mydata2)-2)

```

```

endcolumns.end<-length\((fitsel.ratio) fitsel.ratioN<-cbind\(( Probe= fitsel.ratioAll$Probe,
Accession=fitsel.ratioAll$Accession, Symbol=fitsel.ratioAll$Symbol, Definition=fitsel.ratioAll$Definition,
fitsel.ratioAll[,coefs.col], fitsel.ratioAll[,pvals.col], F=fitsel.ratioAll$F, F.p.value=fitsel.ratioAll$F.p.value,
fitsel.ratioAll[,res.col], fitsel.ratioAll[,ts.col], A= fitsel.ratioAll$A, fitsel.ratioAll[,84:226], fitsel.ratioAll[,57:66], fitsel.ratioAll[,69:77], fitsel.ratioAll[,79:83] ) dim\((fitsel.ratioN) fitsel.ratioN<-fitsel.ratioN[,order\
(fitsel.ratioN$F,decreasing=T),] write.xls\((fitsel.ratioN,file= "combat_complete_genelist.xls",quote=F) 3.
###This code is for analyzing the SH-SY5Y microarray data using WGCNA \((weighted gene coexpression
network analysis) using R ###Paste the following functions into R PickSoftThreshold=function\
(datExpr1,RsquaredCut=0.85, powervector=c\((seq\((1,10,by=1),seq\
(12,20,by=2)),removeFirst=FALSE,no.breaks=10) \{ no.genes <- dim\((datExpr1)\[2] no.genes <- dim\
(datExpr1)\[2] no.samples= dim\((datExpr1)\[1] colname1=c\("Power", "scale law R2", "slope", "truncated
R2","mean\((k)","median\((k)","max\((k)") datout=data.frame\((matrix\((666,nrow=length\
(powervector),ncol=length\((colname1) )) names\((datout)=colname1 datout\[1]=powervector if\((exists\
("fun1")) rm\((fun1) fun1=function\((x) \{ corx=abs\((cor\((x,datExpr1,use="p")) out1=rep\((NA, length\
(powervector) ) for \((j in c\((1:length\((powervector))) \{out1\[j]=sum\((corx^powervector\[j])\} out1 } # end
of fun1 datk=t\((apply\((datExpr1,2,fun1)) for \((i in c\((1:length\((powervector) ) ) \{ nolinkshelp <- datk\[i]-1
cut2=cut\((nolinkshelp,no.breaks) binned.k=tapply\((nolinkshelp,cut2,mean) freq1=as.vector\((tapply\
(nolinkshelp,cut2,length)/length\((nolinkshelp)) # The following code corrects for missing values etc
breaks1=seq\((from=min\((nolinkshelp),to=max\((nolinkshelp),length=no.breaks+1) hist1=hist\
(nolinkshelp,breaks=breaks1,equidist=F,plot=FALSE,right=TRUE) binned.k2=hist1$mids binned.k=ifelse\
(is.na\((binned.k),binned.k2,binned.k) binned.k=ifelse\((binned.k==0,binned.k2,binned.k) freq1=ifelse\
(is.na\((freq1),0,freq1) xx= as.vector\((log10\((binned.k)) if\((removeFirst) \{freq1=freq1\[1]; xx=xx\[1]\}
plot\((xx,log10\((freq1+.000000001),xlab="log10\((k)",ylab="log10\((p\((k))" ) lm1= lm\((as.numeric\((log10\
(freq1+.000000001))~ xx ) lm2=lm\((as.numeric\((log10\((freq1+.000000001))~ xx+1\((10^xx) ) datout\
[i,2]=summary\((lm1)$adj.r.squared datout\[i,3]=summary\((lm1)$coefficients\[2,1] datout\[i,4]=summary\
(lm2)$adj.r.squared datout\[i,5]=mean\((nolinkshelp) datout\[i,6]= median\((nolinkshelp) datout\[i,7]=
max\((nolinkshelp) } datout=signif\((datout,3) print\((data.frame\((datout)); # the cut-off is chosen as
smallest cut with R^2>RsquaredCut ind1=datout\[2]>RsquaredCut indcut=NA indcut=ifelse\((sum\
(ind1)>0,min\((c\((1:length\((ind1))\[ind1]),indcut) # this estimates the power value that should be used. #
Don't trust it. You need to consider slope and mean connectivity as well\(! power.estimate=powervector\
[indcut]\[1] list\((power.estimate, data.frame\((datout)); } TOMdist1=function\((adjmat1, maxADJ=FALSE) \
{ diag\((adjmat1)=0; adjmat1\[is.na\((adjmat1)]=0; maxh1=max\((as.dist\((adjmat1) ); minh1=min\((as.dist\
(adjmat1) ); if \((maxh1>1 | minh1 < 0 ) \{print\((paste\("ERROR: the adjacency matrix contains entries that
are larger than 1 or smaller than 0\(!\(!, max=",maxh1,", min=",minh1)) } else \{ if \(( max\((c\((as.dist\((abs\
(adjmat1-t\((adjmat1))))>0 ) \{print\("ERROR: non-symmetric adjacency matrix\(!\(!") } else \{ kk=apply\
(adjmat1,2,sum) maxADJconst=1 if \((maxADJ==TRUE) maxADJconst=max\((c\((as.dist\((adjmat1 )))
Dhelp1=matrix\((kk,ncol=length\((kk),nrow=length\((kk)) denomTOM= pmin\((as.dist\((Dhelp1),as.dist\((t\
(Dhelp1))) +as.dist\((maxADJconst-adjmat1); gc\();gc\(); numTOM=as.dist\((adjmat1 %*% adjmat1
+adjmat1); #TOMmatrix=numTOM/denomTOM # this turns the TOM matrix into a dissimilarity out1=1-
as.matrix\((numTOM/denomTOM) diag\((out1)=1 out1 } } } makeNetwork = function\((dat,beta,signed=0)\{

```

```

if\(\signed==0)\{ a = abs\(\cor\(\dat,use="p")^beta) t = TOMdist1\(\a) return\(\hclust\(\as.dist\
(t),method="av")) }else\{ a = cor\(\dat,use="p") b = a < 0 a\[b] = 0 a = a^beta t = TOMdist1\(\a) return\
(hclust\(\as.dist\(\t),method="av")) } } cutreeDynamic = function\(\hierclust, maxTreeHeight=1,
deepSplit=TRUE, minModuleSize=50, minAttachModuleSize=100, nameallmodules=FALSE,
useblackwhite=FALSE) \{ if\(\maxTreeHeight >=1)\{ staticCutCluster = cutTreeStatic\(\hiercluster=hierclust,
heightcutoff=0.99, minsize1=minModuleSize) }else\{ staticCutCluster = cutTreeStatic\
(hiercluster=hierclust, heightcutoff=maxTreeHeight, minsize1=minModuleSize) } #get tree height for every
singleton #node_index tree_height demdroHeiAll= rbind\(\ cbind\(\hierclust$merge\[1], hierclust$height),
cbind\(\hierclust$merge\[2], hierclust$height) ) #singletons will stand at the front of the list myorder =
order\(\demdroHeiAll\[1]) #get # of singletons no.singletons = length\(\hierclust$order) demdroHeiAll.sort
= demdroHeiAll\[myorder,] demdroHei.sort = demdroHeiAll.sort\[c\(\1:no.singletons),] demdroHei =
demdroHei.sort\[seq\(\no.singletons, 1, by=-1),] demdroHei\[1] = -demdroHei\[1] # combine with
preliminary cluster-cutoff results demdroHei = cbind\(\demdroHei, as.integer\(\staticCutCluster)) # re-order
the order based on the dendrogram order hierclust$order demdroHei.order = demdroHei\[hierclust$order,]
static.clupos = locateCluster\(\demdroHei.order\[1, 3]) if \(\is.null\(\static.clupos) )\{ module.assign = rep\(\0,
no.singletons) colcode.reduced = assignModuleColor\(\module.assign, minsize1=minModuleSize,
anameallmodules=nameallmodules, auseblackwhite=useblackwhite) return \(\ colcode.reduced ) }
static.no = dim\(\static.clupos)1 static.clupos2 = static.clupos static.no2 = static.no #split individual
cluster if there are sub clusters embedded mcycle=1 while\(\1==1)\{ clupos = NULL for \(\i in c\
(1:static.no))\{ mydemdroHei.order = demdroHei.order\[ c\(\static.clupos\[i,1]:static.clupos\[i,2]),] #index
to \[1, clusterSize] mydemdroHei.order\[ 1] = mydemdroHei.order\[ 1] - static.clupos\[i, 1] + 1 #cat\("Cycle
", as.character\(\mcycle), "cluster \(", static.clupos\[i,1], static.clupos\[i,2], ") \n") #cat\("i=", as.character\(\i),
"\n") iclupos = processInvididualCluster\(\mydemdroHei.order, cminModuleSize = minModuleSize,
cminAttachModuleSize = minAttachModuleSize) iclupos\[1] = iclupos\[1] + static.clupos\[i, 1] -1
#recover the original index iclupos\[2] = iclupos\[2] + static.clupos\[i, 1] -1 clupos = rbind\(\clupos,
iclupos) #put in the final output buffer } if\(\deepSplit==FALSE)\{ break } if\(\dim\(\clupos)1 != static.no) \{
static.clupos = clupos static.no = dim\(\static.clupos)1 }else\{ break } mcycle = mcycle + 1 #static.clupos }
final.cnt = dim\(\clupos)1 #assign colors for modules module.assign = rep\(\0, no.singletons)
module.cnt=1 for \(\i in c\(\1:final.cnt) ) \{ sdx = clupos\[i, 1] #module start point edx = clupos\[i, 2]
#module end point module.size = edx - sdx +1 if\(\module.size = minsize1 no.modules=sum\
(modulebranch) colorhelp = rep\(\-1, length\(\labelpred) ) if \(\ no.modules==0)\{ print\("No module
detected") } else\{ for \(\i in c\(\1:no.modules)) \{ colorhelp=ifelse\(\labelpred==modulename\[i],i ,colorhelp)
} } colorhelp } locateCluster = function\(\clusterlabels) \{ no.nodes = length\(\clusterlabels)
clusterlabels.shift = c\(\clusterlabels1, c\(\clusterlabels\[1:\(no.nodes-1)]) ) #a non-zero point is the start
point of a cluster and it previous point is the end point of the previous cluster label.diff = abs\
(clusterlabels - clusterlabels.shift) #process the first and last positions as start/end points if they belong
to a cluster instead of no cluster "-1" if\(\clusterlabels1 >0) \{label.diff1=1} if\(\clusterlabels\[no.nodes]>0)
\{label.diff\[no.nodes]=1} flagpoints.bool = label.diff > 0 if\(\ sum\(\flagpoints.bool) ==0)\{ return\(\NULL) }
flagpoints = c\(\1:no.nodes)\[flagpoints.bool] no.points = length\(\flagpoints) myclupos=NULL for\(\i in c\
(1:\(no.points-1)) )\{ idx = flagpoints\[i] if\(\clusterlabels\[idx]>0)\{ myclupos = rbind\(\myclupos, c\(\idx,

```

```

flagpoints\[i+1]-1) ) } } myclupos } processInvididualCluster = function\(clusterDemdroHei,
cminModuleSize=50, cminAttachModuleSize=100, minTailRunlength=12, useMean=0){ #for debug: use
all genes #clusterDemdroHei =demdroHei.order no.cnodes = dim\(clusterDemdroHei)1 cmaxhei = max\
(clusterDemdroHei\[, 2]) cminhei = min\(clusterDemdroHei\[, 2]) cmeanhei = mean\(clusterDemdroHei\[,
2]) cmidhei = \((cmeanhei + cmaxhei)/2.0 cdwnhei = \((cmeanhei + cminhei)/2.0 if \((useMean==1)\{
comphei = cmidhei }else if \((useMean==-1)\{ comphei = cdwnhei }else{ #normal case comphei =
cmeanhei } # compute height difference with mean height heidiff = clusterDemdroHei\[,2] - comphei
heidiff.shift = shiftSequence\(heidiff, -1) # get cut positions # detect the end point of a cluster, whose
height should be less than meanhei # and the node behind it is the start point of the next cluster which
has a height above meanhei cuts.bool = \((heidiff<0) & \((heidiff.shift > 0) cuts.bool1 = TRUE cuts.bool\
[no.cnodes] = TRUE if\((sum\((cuts.bool)==2)\{ if \((useMean==0)\{ new.clupos=processInvididualCluster\
(clusterDemdroHei=clusterDemdroHei, cminModuleSize=cminModuleSize,
cminAttachModuleSize=cminAttachModuleSize, useMean=1) }else if\((useMean==1)\{
new.clupos=processInvididualCluster\(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize, cminAttachModuleSize=cminAttachModuleSize, useMean=-1) }else\{
new.clupos = rbind\(c\((1, no.cnodes)) } return \((new.clupos) } #a good candidate cluster-end point should
have significant # of ahead nodes with head < meanHei cutindex = c\((1:no.cnodes)\[cuts.bool] no.cutps =
length\((cutindex) runlens = rep\((999, no.cutps) cuts.bool2 = cuts.bool for\((i in c\((2:\(no.cutps-1)) )\{ seq
= c\(\((cutindex\[i-1]+1):cutindex\[i] ) runlens\[i] = runlengthSign\((heidiff\[seq], leftOrright=-1, mysign=-1)
if\((runlens\[i] < minTailRunlength)\{ #cat\("run length=", runlens\[i], "\n") cuts.bool2\[ cutindex\[i] ] =
FALSE } } #attach SMALL cluster to the left-side BIG cluster if the small one has smaller mean height
cuts.bool3=cuts.bool2 if\((sum\((cuts.bool2) > 3) \{ curj = 2 while \((1==1)\{ cutindex2 =c\((1:no.cnodes)\
[cuts.bool2] no.clus = length\((cutindex2) -1 if \((curj>no.clus)\{ break } pre.sdx = cutindex2\[ curj-1 ]+1
#previous module start point pre.edx = cutindex2\[ curj ] #previous module end point pre.module.size =
pre.edx - pre.sdx +1 pre.module.hei = mean\((clusterDemdroHei\[c\((pre.sdx:pre.edx) , 2]) cur.sdx =
cutindex2\[ curj ]+1 #previous module start point cur.edx = cutindex2\[ curj+1 ] #previous module end
point cur.module.size = cur.edx - cur.sdx +1 cur.module.hei = mean\((clusterDemdroHei\[c\
((cur.sdx:cur.edx) , 2]) #merge to the leftside major module, don't change the current index "curj" #if\(\
((pre.module.size >minAttachModuleSize)&\((cur.module.hei 2)\{ if\(\ \((cutindex2\[no.cutps] - cutindex2\
[no.cutps-1]+1) < cminModuleSize )\{ cuts.bool2\[ cutindex2\[no.cutps-1] ] =FALSE } } if\((1==2)\{
myseqnce = c\((2300:3000) cutdisp = ifelse\((cuts.bool2==T, "red","grey" ) #re-order to the normal one with
sequential singleton index par\((mfrow=c\((3,1), mar=c\((0,0,0,0) ) plot\((h1row, labels=F,
xlab="",ylab="",main="",sub="",axes = F) barplot\((heidiff\[myseqnce], col= "black", space=0,
border=F,main="", axes = F, axisnames = F) barplot\((height=rep\((1, length\((cutdisp\[myseqnce])), col=
as.character\((cutdisp\[myseqnce]), space=0, border=F,main="", axes = F, axisnames = F) par\((mfrow=c\
((1,1), mar=c\((5, 4, 4, 2) + 0.1) } cutindex2 = c\((1:no.cnodes)\[cuts.bool2] cutindex21=cutindex21-1 #the
first no.cutps2 = length\((cutindex2) if\((no.cutps2 > 2)\{ new.clupos = cbind\(( cutindex2\[c\((1:\(no.cutps2-
1)))+1, cutindex2\[c\((2:no.cutps2)) ) }else\{ new.clupos = cbind\(( 1, no.cnodes) } if \(( dim\((new.clupos)1
== 1 )\{ if \((useMean==0)\{ new.clupos=processInvididualCluster\((clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize, cminAttachModuleSize=cminAttachModuleSize, useMean=1) }else

```

```

if(useMean==1){ new.clupos=processIndividualCluster(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize, cminAttachModuleSize=cminAttachModuleSize, useMean=-1) } }
new.clupos } shiftSequence = function(mysequence, delta){ seqlen = length(mysequence) if(delta>0){
{ finalseq=c(mysequence[1:delta], mysequence[1:(seqlen-delta)]) }else{ posdelta = -delta finalseq=c(
(mysequence[(posdelta+1):seqlen], mysequence[(seqlen-posdelta+1):seqlen]) } finalseq }
runlengthSign = function(mysequence, leftOrright=-1, mysign=-1){ seqlen = length(mysequence) if(
(leftOrright<0){ pseq = rev(mysequence) }else{ pseq = mysequence } if(mysign<0){ #see where the
first POSITIVE number occurs nonezero.bool = (pseq > 0) }else{ #see where the first NEGATIVE number
occur nonezero.bool = (pseq < 0) } if( sum(nonezero.bool) > 0){ runlength = min( c(1:seqlen)[
nonezero.bool] ) - 1 }else{ runlength = 0 } } assignModuleColor = function(labelpred, minsize1=50,
anameallmodules=FALSE, auseblackwhite=FALSE) { # here we define modules by using a height cut-off
for the branches #labelpred= cutree(hiercluster,h=heightcutoff) #cat(labelpred) #"0", grey module
doesn't participate color assignment, directly assigned as "grey" labelpredNoZero = labelpred[ labelpred
>0 ] sort1=sort(-table(labelpredNoZero)) sort1 modulename= as.numeric(names(sort1))
modulebranch= sort1 >= minsize1 no.modules=sum(modulebranch) # now we assume that there are
fewer than 10 modules #colorcode=c#
("turquoise","blue","brown","yellow","green","red","black","purple","orange","pink",
#"greenyellow","lightcyan","salmon","midnightblue","lightyellow","chartreuse",
"chocolate","gold","springgreen","tan","violet","plum","aliceblue","aquamarine",
"azure","bisque","burlywood","coral","cornsilk","cyan","darkorchid","deeppink",
"firebrick","forestgreen","gainsboro","wheat") orderofcolors = order(runif(length(colors))) colorcode =
colors() colorcode = colorcode[orderofcolors] notgrey = substr(colorcode,1,4) "gray" | substr(
(colorcode,1,4) "grey" colorcode = colorcode[!notgrey] #"grey" means not in any module; colorhelp=rep(
("grey",length(labelpred)) if ( no.modules==0){ print("No module detected\n") } else{ if ( no.modules
> length(colorcode) ){ print(length(colorcode)) print( paste("Too many modules \n", as.character(
(no.modules)) ) } if ( (anameallmodules==FALSE) || (no.modules <=length(colorcode)) ){
labeledModules = min(no.modules, length(colorcode) ) for (i in c(1:labeledModules)) {
colorhelp=ifelse(labelpred==modulename[i],colorcode[i],colorhelp) } colorhelp=factor(
(colorhelp,levels=c(colorcode[1:labeledModules],"grey"))) }else{#anameallmodules==TRUE and
no.modules >length(colorcode) maxcolors=length(colorcode) labeledModules = no.modules
extracolors=NULL blackwhite=c("red", "black") for(i in c((maxcolors+1):no.modules)){ if(
(auseblackwhite==FALSE){ icolor=paste("module", as.character(i), sep="") }else{#use balck white
alternatively represent extra colors, for display only #here we use the ordered label to avoid put the same
color for two neighboring clusters icolor=blackwhite[1+(as.integer(modulename[i])%%2) ] }
extracolors=c(extracolors, icolor) } #combine the true-color code and the extra colorcode into a uniform
colorcode for #color assignment allcolorcode=c(colorcode, extracolors) for (i in c(1:labeledModules)) {
colorhelp=ifelse(labelpred==modulename[i],allcolorcode[i],colorhelp) } colorhelp=factor(
(colorhelp,levels=c(allcolorcode[1:labeledModules],"grey"))) } } colorhelp } hclustplot1=function(
(hier1,couleur,title1="Colors sorted by hierarchical clustering") { if (length(hier1$order) != length(
(couleur) ) { print("ERROR: length of color vector not compatible with no. of objects in the hierarchical

```

```

tree"))}; if (length(hier1$order) == length(couleur) ) {\ barplot(height=rep(1, length(couleur)), col=
as.character(couleur[hier1$order]),border=NA, main=title1,space=0, axes=F)} } plotDend=function\
(clust,colorh){ par(mfrow=c(2,1),mar=c(2,2,2,2)) plot(clust,labels=F) hclustplot1(clust,colorh) }
MakeVis = function(data,colorMod,vis.dir,pwr1,symbols){ q = table(colorMod) n = length(q) for(i in c\
(1:n)){ if(names(q[i]) != "grey"){ collect_garbage() newNet = pairwiseCode(data\
[,colorMod==names(q[i])],filename=paste(vis.dir,names(q\
[i]),".txt",sep=""),pwr=pwr1,geneAnnot=symbols,this.col=names(q[i])) print(q[i]) if(!exists\
("newNetwork")){ newNetwork = newNet } else { newNetwork = rbind(newNetwork,newNet) } } } return\
(newNetwork) } MakePlots = function(data,colorMod,plots.dir){ PC = ModulePrinComps1\
(data,colorMod) q = table(colorMod) n = length(q) u = q > 1 for(i in c(1:n)){ if(u[i]){ c = dimnames\
(PC[1])[2] indC = substr(c[i],3,25) filename = paste(plots.dir,indC,".png",sep="") png(filename)
showHeat(indC,data,colorMod,PC) dev.off() print(paste(filename," printed to file")) }else{ indC =
substr(c[i],3,25) print(paste(filename," only contains 1 gene")) } } } showHeat = function\
(color,data,colorh,PC){ q = table(colorh) z = names(q) == color Probe_sets = dimnames(data)[2] par\
(mfrow=c(2,1),mar=c(2,2,2,2)) par(mar=c(2,3,7,3)) par(oma=c(0,0,2,0)) datcombined=data.frame\
(rbind(data[,colorh==color])) samplelabel=as.character(dimnames(data)[1]) plot.mat(t(scale\
(datcombined)), rlabels=as.character(Probe_sets[colorh==color]), clabels=samplelabel, rcols=color)
par(mar=c(2,2,0,2)) barplot(PC[1][z],space=0,beside=TRUE,axes=FALSE,col=color) }
ModulePrinComps1=function(datexpr,couleur) {\ modlevels=levels(factor(couleur))
PrinComps=data.frame(matrix(666,nrow=dim(datexpr)[1],ncol= length(modlevels))) varexplained=
data.frame(matrix(666,nrow= 5,ncol= length(modlevels))) names(PrinComps)=paste\
("PC",modlevels,sep="") for(i in c(1:length(modlevels)) ){\ print(i) modulenname = modlevels[i]
restrict1= as.character(couleur== modulenname # in the following, rows are genes and columns are
samples datModule=t(datexpr[, restrict1]) datModule=impute.knn(as.matrix(datModule))
datModule=t(scale(t(datModule))) svd1=svd(datModule) mtitle=paste("PCs of ", modulenname,"
module", sep="") varexplained[,i]= (svd1$d[1:5])^2/sum(svd1$d^2) # this is the first principal component
pc1=svd1$v[,1] signh1=sign(sum(cor(pc1, t(datModule)))) if (signh1 != 0) pc1=signh1* pc1
PrinComps[,i]= pc1 } ModuleConformity= rep(666,length=dim(datexpr)[2]) for(i in 1:(dim(datexpr)\
[2])) ModuleConformity[i]=abs(cor(datexpr[,i], PrinComps[,match(couleur[i], modlevels)],
use="pairwise.complete.obs")) list(PrinComps=PrinComps, varexplained=varexplained,
ModuleConformity=ModuleConformity) } pairwiseCode = function\
(vals,filename="forvis.txt",pwr,geneAnnot,links=1000,this.col){ n = dimnames(vals)[2] newPC =
ModulePrinComps1(vals,rep("black",dim(vals)[2])) newSim = cor(vals,use="p") newAdj = abs\
(newSim^pwr) diag(newAdj) = 0 newDeg = apply(newAdj,1,sum) newDeg = newDeg/max(newDeg)
newKme = cor(vals,newPC[1][1],use="p") newTom = TOMdist1(newAdj) newTom = 1-newTom sz =
dim(newAdj)[1] TomList = vector("logical",sz^2) for(j in c(1:sz)){ TomList[(j-1)**sz+1):(j\
**sz)] = newTom[,j] } ord = order(TomList,decreasing=TRUE) if(length(n) > 33){ len = links } else {\
len = length(n)^2 } first = ord[1:len] %% sz + 1 second = ord[1:len] %% sz sec = second == 0 first[sec] =
first[sec]-1 second[sec] = sz id = geneAnnot[n,2] datout = matrix(0,len,6) for(j in c(1:len)){ datout\
[j,1] = paste(id[first[j]]) datout[j,2] = paste(id[second[j]]) datout[j,3] = 0 if(newSim[first[j],second\

```

```

[j]] > 0){ datout[j,4] = as.character("M0011") } else { datout[j,4] = as.character("M0015") } datout[j,5]
= TomList[ord[j]] datout[j,6] = newSim[first[j],second[j]] } write.table(datout,file=filename,sep="\t")
return(cbind(n,newDeg[n],as.numeric(newKme[n,1]),as.character(geneAnnot[n,2]),this.col)) }
modpcas = function(dat,mod){ q = names(table(mod)) u = vector("list",length(q)) d = vector(
("list",length(q)) v = vector("list",length(q)) names(u) = q names(d) = q names(v) = q for(i in c(
(1:length(q)))){ b = mod == q[i] impDat = impute.knn(as.matrix(dat[names(mod[b]),])) sclDat = t(
(scale(t(impDat))) svdDat = svd(sclDat) for(j in c(1:min(dim(dat)))){ signh1 = sign(sum(cor(
(svdDat$v[j],t(sclDat)))) if(signh1 != 0){ svdDat$v[j]=signh1*svdDat$v[j] svdDat$u
[j]=signh1*svdDat$u[j] } u[[i]] = svdDat$u dimnames(u[[i]])[1] = names(mod[b]) d[[i]] =
svdDat$d v[[i]] = svdDat$v } } return(list(u=u,d=d,v=v)) } removePC = function(pc,start){ dat = vector(
("logical",dim(pc$v[1])[1]) for(i in c(1:length(pc$u))){ end = dim(pc$u)[2] D = diag(pc$d[[i]]) e =
pc$u[[i]][start:end] %**% D[start:end,start:end] %**% t(pc$v[[i]][start:end]) dat = rbind(dat,e) }
return(dat[2:dim(dat)[1],]) } ###Description of variables normalizedData – the normalized microarray
data with genes in rows and samples in columns, should also have gene identifiers for row names
power_to_scale_connections – obtained with guidance from PickSoftThreshold (below), see Zhang et
al., (2005) for more information geneAnnotation – matrix with gene names in the second column and
gene identifiers as row names ###Create the co-expression network PickSoftThreshold(t(
(normalizedData)) power_to_scale_connections = 6 genesClusteredOnTO = makeNetwork(t(
(normalizedData),power_to_scale_connections) ###Identify modules and plot dendrogram
genesAssignedToModules = cutreeDynamic(genesClusteredOnTO) plotDend(
(genesClusteredOnTO,genesAssignedToModules) ###Plot module heatmaps and make files for VisAnt
MakePlots(normalizedData,genesAssignedToModules,"./heatmaps/") Network = MakeVis(
(normalizedData,genesAssignedToModules,"./visant/",geneAnnotation,power_to_scale_connections)
write.table(Network,file="network.csv",sep=",") ###Remove first principal component names(
(genesAssignedToModules) = dimnames(normalizedData)[1] principalComponents = modpcas(
(normalizedData,genesAssignedToModules) dataWithoutFirstPrincipalComponent = removePC(
(principalComponents,2) 4. ###This code is for calculating the hypergeometric probability of the overlap
of two datasets with permutations using R hyperFast = function(
(pop1,subpop1,pop2,subpop2,perm,b=0){ if(b==1){browser()} p1 = seq(1,pop1,by=1) p2 = seq(
(1,pop2,by=1) out = vector(mode="logical",perm) i = 1 while(i < perm){ over = sample(p1,subpop1)
%in% sample(p2,subpop2) if(length(table(over)) > 1){ out[i] = table(over)[2] } else { out[i] = 0 } i =
i + 1 } return(out) } x=hyperFast(a,b,c,d,e) #a=the total number of probesets in the first gene list #b=the
total number of differentially expressed genes in the first gene list #c=the total number of probesets in the
second gene list #d=the total number of differentially expressed genes in the second gene list #e=the
number of permutations #to calculate Z-score, use the following, where f=the number of overlapping
genes in the two lists: y=(f-mean(x))/sd(x) #to calculate a PValue for the Z-score: z=pnorm(
(y,log=TRUE) 1-(2^z)

```

References

Smyth, G.K., Gentleman, R., Carey, V., Dudoit, S., Irizarry, R., and Huber, W. (2005). Limma: linear models for microarray data. In *Bioinformatics and Computational Biology Solutions using R and Bioconductor* (Springer), pp. 397-420. Gentleman, R.C., Carey, V.J., Bates, D.M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., et al. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol*, 5. Zhang, B., and Horvath, S. (2005) A general framework for weighted gene co-expression network analysis. *Stat Appl Genet Mol Biol*, 4: Article 17. Langfelder, P., Zhang, B., and Horvath, S. (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *Bioinformatics*, 24(5): 719-20.