Host-parasite-clocks

Biplabendu Das

20 October, 2021

## Overview/Goals

This document provides a step-by-step tutorial that demonstrates how to: (1) build a circadian gene co-expression network (GCN), (2) how to functionally annotate the network using published data, (3) infer functions of your gene-clusters-of-interest.

## Step 1: Build circadian GCN

### 1.1 Load data

We will build a circadian GCN for the ant, *Camponotus floridanus*, using time-course RNASeq data collected in Das and de Bekker (2021; bioRxiv). The raw data is deposited in NCBI under BioProject PRJNA704762.

Description of the dataset: Three forager and three nurse ant brains were sampled and pooled for RNA extraction and Illumina sequencing, every 2h over a 24h period. This resulted in 24 RNASeq datasets for ant brains (12 forager and 12 nurse datasets over the course of a 24h LD 12:12 day).

One would need to perform the usual steps – trimming the reads, mapping the reads to the genome, and quantifying normalized gene counts – to obtain normalized gene expression data from the raw reads. At the end, for each gene in the genome, we should have the normalized expression for each time point, throughout the 24h day.

For the purpose of this tutorial, we assume that you have organized the processed data into a (gene-expr X time-point) format, in a chronological order, as shown below.



X2F = forager brain sampled at ZT2 (2h after lights were turned on), X4F = forager brain sampled at ZT4, and so on.

Now we read the data into R.

# loading database which contains data for Das and de Bekker 2021 (bioRxiv)  
db <- dbConnect(RSQLite::SQLite(), paste0(path\_to\_repo,"/data/databases/TC5\_data.db"))  
  
# extract the (gene-expr X time-point) data  
dat <-  
 db %>%  
 tbl(., "annot\_fpkm") %>%  
 select(gene\_name, X2F:X24N) %>%  
 collect()  
  
dim(dat)

## [1] 13813 25

### 

### 1.2 Clean data

The above dataset contains all genes (n=13,813) in the ant genome. However, not all of these genes are expressed in the ant brain, and some are expressed at very low levels that are not biologically meaningful.

Therefore, we will only keep the genes that are “expressed” (≥1 FPKM) in the ant brain, for at least half of all the sampled time points.

# Which genes are expressed throughout the day in both forager and nurses brains?  
daily.exp.genes <-  
 tbl(db, "expressed\_genes") %>% # note, the information is already available in the database  
 filter(exp\_half\_for == "yes" & exp\_half\_nur == "yes") %>%  
 collect() %>%  
 pull(gene\_name)  
  
# Subset the gene-expr X time-point file  
dat <- dat %>% filter(gene\_name %in% daily.exp.genes)  
dim(dat)

## [1] 9139 25

This is our cleaned, input data file.

The daily expression for these 9139 genes will be used to create the circadian GCN of *Camponotus floridanus*.

### 

### 1.3 Format data

To create the ant GCN, we will need to calculate the expression similarity (co-expression) of different gene pairs. Therefore, we would like to normalize the gene expression data by log2-transformation. Let’s do that and visualize the result.

datExpr = as.data.frame(t(log2(dat[-c(1)]+1)))  
names(datExpr) = dat$gene\_name  
rownames(datExpr) = names(dat)[-c(1)]

# ----------------------------------------------------------- #  
# USE THE FOLLOWING CODE TO CHECK IF YOU HAVE ANY BAD SAMPLES #  
# ----------------------------------------------------------- #  
 # gsg = goodSamplesGenes(datExpr0, verbose = 3);  
 # gsg$allOK  
 #  
 # sampleTree = hclust(dist(datExpr0), method = "average");  
 # # Plot the sample tree: Open a graphic output window of size 12 by 9 inches  
 # # The user should change the dimensions if the window is too large or too small.  
 # sizeGrWindow(12,9)  
 # #pdf(file = "Plots/sampleClustering.pdf", width = 12, height = 9);  
 # par(cex = 1);  
 # par(mar = c(0,4,2,0))  
 # plot(sampleTree, main = "Sample clustering to detect outliers", sub="", xlab="", cex.lab = 1.5,  
 # cex.axis = 1.5, cex.main = 2)  
# ----------------------------------------------------------- #  
  
# save the number of genes and samples  
# that will be used to create the circadian GCN  
nGenes = ncol(datExpr)  
nSamples = nrow(datExpr)  
  
# visualize the log-transformed data  
x = reshape2::melt(as.matrix(t(datExpr)))  
colnames(x) = c('gene\_id', 'sample', 'value')  
ggplot(x, aes(x=value, color=sample)) + geom\_density() + theme\_Publication()



Figure 1. Normalized gene expression: The density plot above shows the distribution of log2-transformed gene expression values for each sample.

### 1.4 Calculate gene-gene similarity

Now, we can calculate the pairwise gene expression similarity for each of the 9139 genes and save it to a matrix.

I calculated expression similarity for all gene pairs in a dataset using Kendall’s tau, which measures the ordinal relationship between two variables and is used in rhythmicity detection algorithms [1].

## Calculate Kendall's tau-b correlation for each gene-gene pair  
#  
# sim\_matrix <- cor((datExpr), method = "kendall") # this step takes time  
# save(sim\_matrix, file = paste0(path\_to\_repo, "/results/temp\_files/sim\_matrix\_for\_nur\_TC5.RData")) # might be useful to save the sim\_matrix and  
load(paste0(path\_to\_repo, "/results/temp\_files/sim\_matrix\_for\_nur\_TC5.RData")) # load it up  
  
## Let's display a chunk of the matrix (code from Hughitt 2016; github)  
heatmap\_indices <- sample(nrow(sim\_matrix), 500)  
gplots::heatmap.2(t(sim\_matrix[heatmap\_indices, heatmap\_indices]),  
 col=inferno(100),  
 labRow=NA, labCol=NA,  
 trace='none', dendrogram='row',  
 xlab='Gene', ylab='Gene',  
 main='Similarity matrix \n correlation method = "kendall" \n (500 random genes)',  
 density.info='none', revC=TRUE)

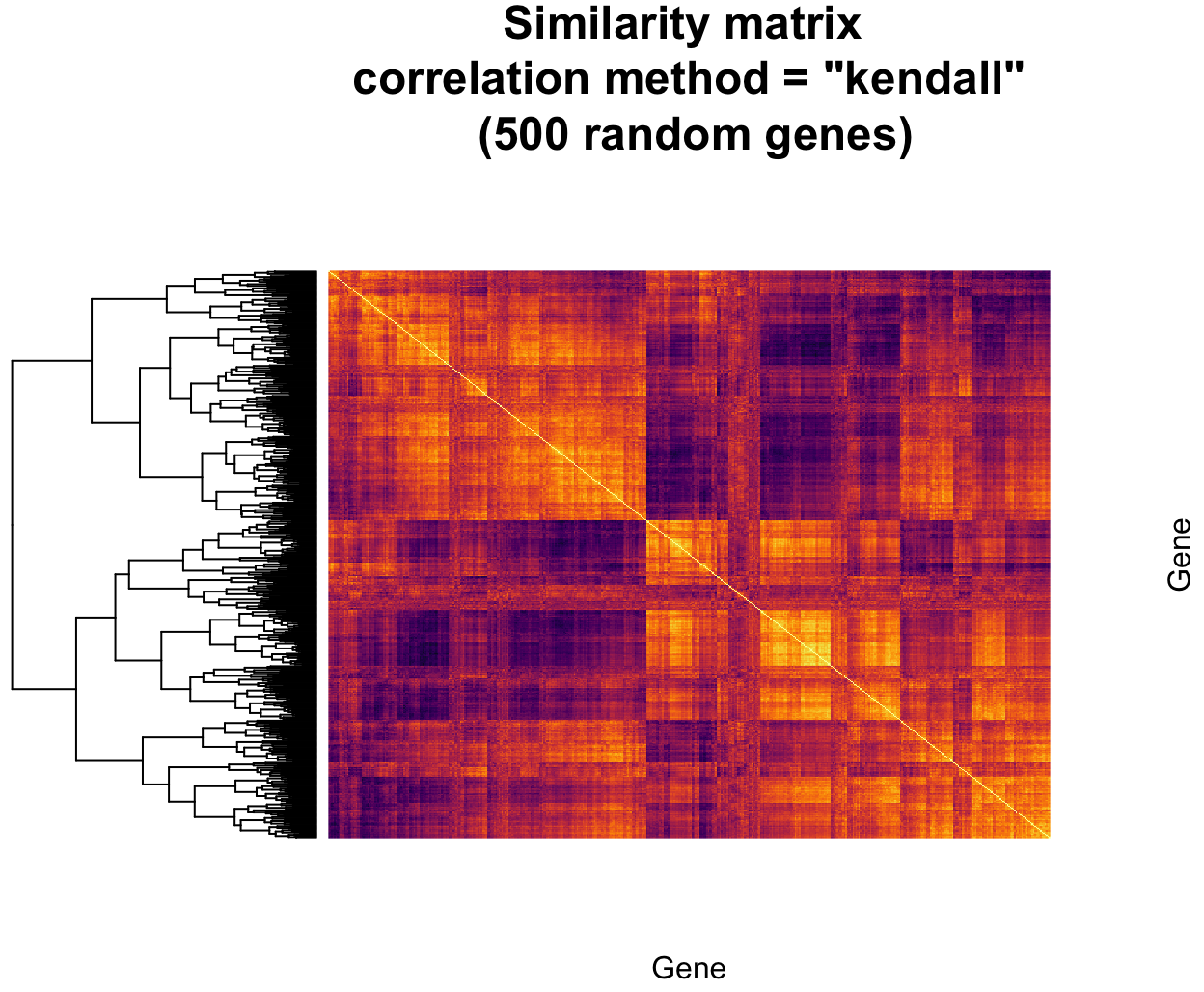


Figure 2. Similarity matrix: The heatmap shows the pairwise Kendall’s tau correlation for a set of 500 genes randomly pulled from the 9139 genes expressed in the ant brain.

### 1.5 Create adjacency matrix

From the above similarity matrix, we then need to create the adjacency matrix needed for constructing a gene co-expression network.

To create the adjacency matrix, we need to first identify the soft-thresholding power by calling the network topology analysis function from the WGCNA package [2].

# Choose a set of soft-thresholding powers  
powers = c(c(1:10), seq(from = 12, to=20, by=2))  
# # Call the network topology analysis function  
sft = pickSoftThreshold(datExpr, powerVector = powers, verbose = 5)

## pickSoftThreshold: will use block size 4895.  
## pickSoftThreshold: calculating connectivity for given powers...  
## ..working on genes 1 through 4895 of 9139  
## ..working on genes 4896 through 9139 of 9139  
## Power SFT.R.sq slope truncated.R.sq mean.k. median.k. max.k.  
## 1 1 0.845 1.900 0.995 3310.0 3390.00 4730  
## 2 2 0.248 0.276 0.930 1720.0 1710.00 3200  
## 3 3 0.343 -0.284 0.907 1050.0 988.00 2410  
## 4 4 0.696 -0.580 0.922 701.0 616.00 1930  
## 5 5 0.818 -0.762 0.951 499.0 402.00 1600  
## 6 6 0.847 -0.896 0.942 371.0 272.00 1360  
## 7 7 0.854 -0.992 0.933 285.0 190.00 1180  
## 8 8 0.868 -1.060 0.935 225.0 136.00 1030  
## 9 9 0.879 -1.110 0.940 181.0 99.10 919  
## 10 10 0.874 -1.160 0.928 148.0 73.40 824  
## 11 12 0.879 -1.220 0.928 103.0 42.20 676  
## 12 14 0.879 -1.280 0.921 74.8 25.40 568  
## 13 16 0.874 -1.310 0.916 56.1 15.80 485  
## 14 18 0.842 -1.360 0.884 43.2 10.10 420  
## 15 20 0.827 -1.390 0.874 34.0 6.64 367

# Plot the results:  
# sizeGrWindow(9, 5)  
# par(mfrow = c(1,2));  
cex1 = 0.9;  
# Scale-free topology fit index as a function of the soft-thresholding power  
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])\*sft$fitIndices[,2],  
 xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fit,signed R^2",type="n",  
 main = paste("Scale independence"));  
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])\*sft$fitIndices[,2],  
 labels=powers,cex=cex1,col="red");  
# this line corresponds to using an R^2 cut-off of h  
abline(h=0.90,col="red")

# Mean connectivity as a function of the soft-thresholding power  
plot(sft$fitIndices[,1], sft$fitIndices[,5],  
 xlab="Soft Threshold (power)",ylab="Mean Connectivity", type="n",  
 main = paste("Mean connectivity"))  
text(sft$fitIndices[,1], sft$fitIndices[,5], labels=powers, cex=cex1,col="red")

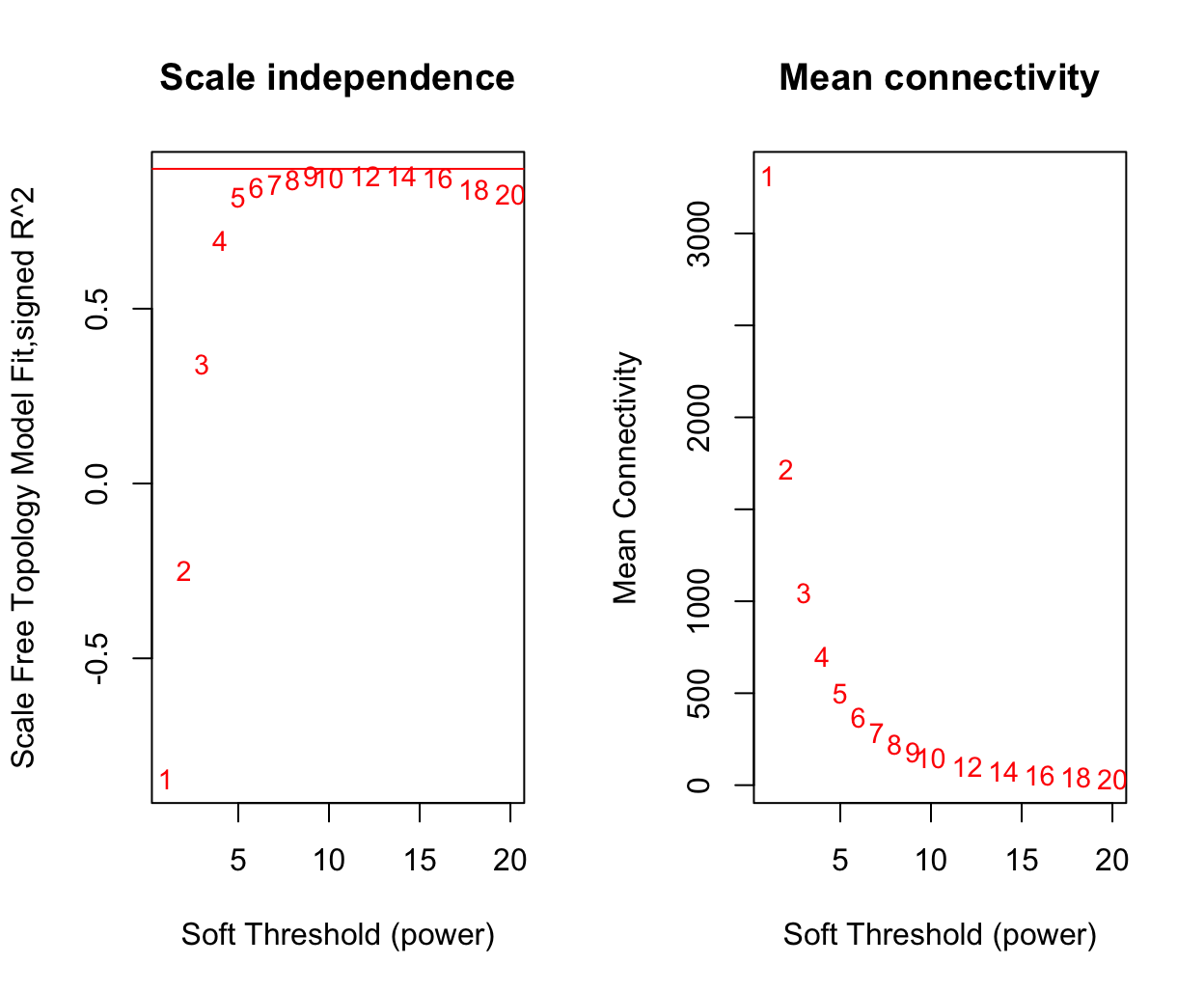


Figure 3. Soft-thresholding power: The above plots show the effect of soft-thresholding power on the topology and the mean connectivity of the transformed similarity matrix (network).

NOTE: The scale-free topology fit index reaches ~0.9 at a soft-thresholding power of 9 and it does not improve drastically beyond that.

So, we will set our soft thresholding power to 9 for creating the adjacency matrix.

## Specify the soft-thresholding-power  
soft.power = 9  
  
## Construct adjacency matrix  
# adj\_matrix <- adjacency.fromSimilarity(sim\_matrix,  
# power=soft.power,  
# type='signed'  
# )  
# save(adj\_matrix, file = paste0(path\_to\_repo, "/results/temp\_files/adj\_matrix\_for\_nur\_TC5.RData")) # might be useful to save the sim\_matrix and  
load(paste0(path\_to\_repo, "/results/temp\_files/adj\_matrix\_for\_nur\_TC5.RData")) # load it up

# Convert adj\_matrix to matrix  
gene\_ids <- rownames(adj\_matrix)  
  
adj\_matrix <- matrix(adj\_matrix, nrow=nrow(adj\_matrix))  
rownames(adj\_matrix) <- gene\_ids  
colnames(adj\_matrix) <- gene\_ids  
  
## Same heatmap as before, but now with the power-transformed adjacency matrix  
gplots::heatmap.2(t(adj\_matrix[heatmap\_indices, heatmap\_indices]),  
 col=inferno(100),  
 labRow=NA, labCol=NA,  
 trace='none', dendrogram='row',  
 xlab='Gene', ylab='Gene',  
 main='Adjacency matrix',  
 density.info='none', revC=TRUE)

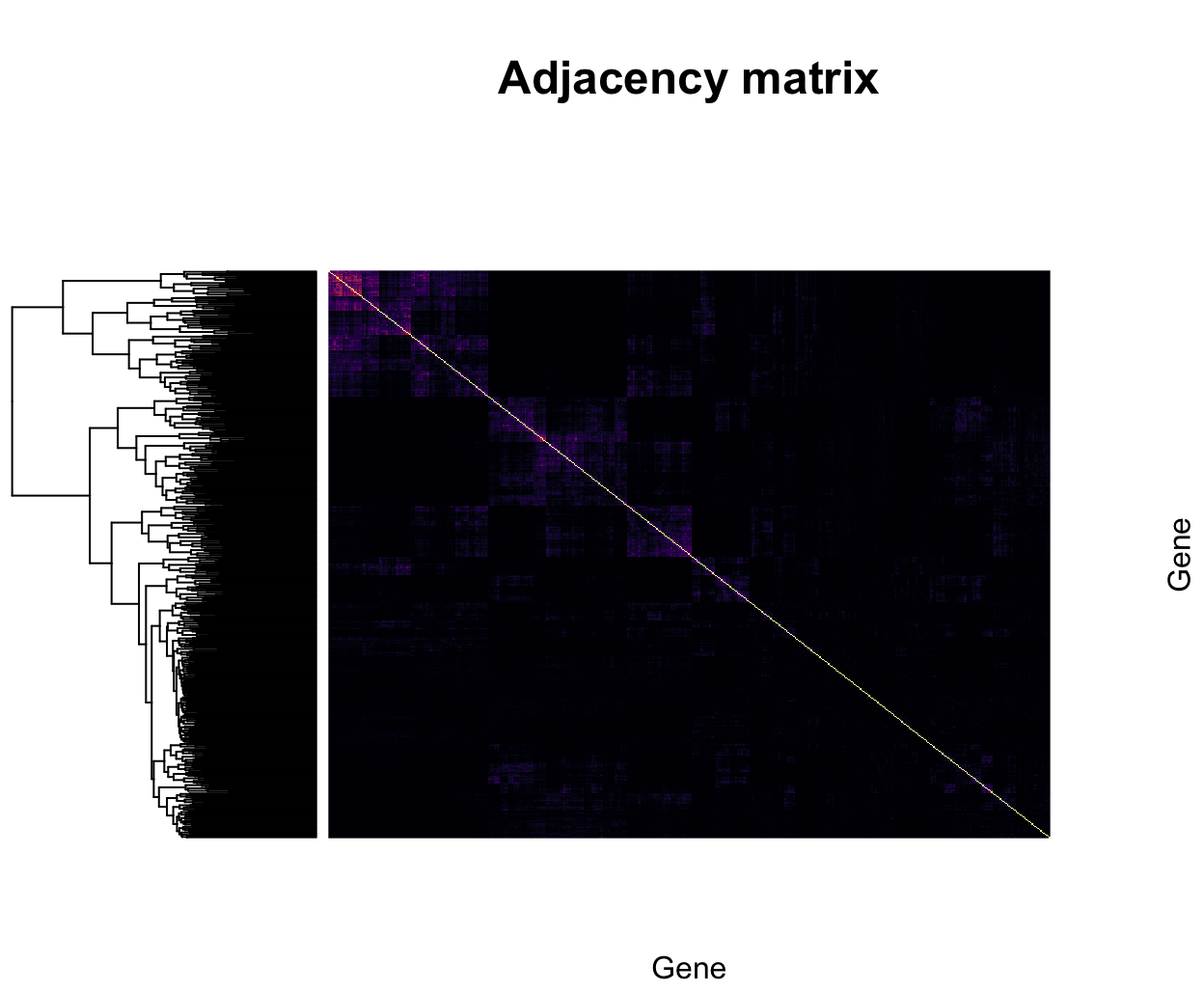


Figure 4. Adjacency matrix: The heatmap shows the result of the power-transformation on the similarity of the 500 random genes shown previously in Figure 2. As you can see, only the highest pair-wise correlations are retained whereas the weak correlations tend to zero.

## Step 2: Identify gene clusters

### 2.1 Create topological overlap matrix

# Turn adjacency into topological overlap  
# TOM = TOMsimilarity(adj\_matrix);  
# dissTOM = 1-TOM  
# save(dissTOM, file = paste0(path\_to\_repo, "/results/temp\_files/dissTOM\_for\_nur\_TC5.RData")) # might be useful to save the sim\_matrix and  
load(paste0(path\_to\_repo, "/results/temp\_files/dissTOM\_for\_nur\_TC5.RData")) # load it up  
  
# Call the hierarchical clustering function  
geneTree = hclust(as.dist(dissTOM), method = "average")  
  
# Plot the resulting clustering tree (dendrogram)  
# sizeGrWindow(12,9)  
plot(geneTree, xlab="", sub="", main = "Gene clustering on TOM-based dissimilarity",  
 labels = FALSE, hang = 0.04)

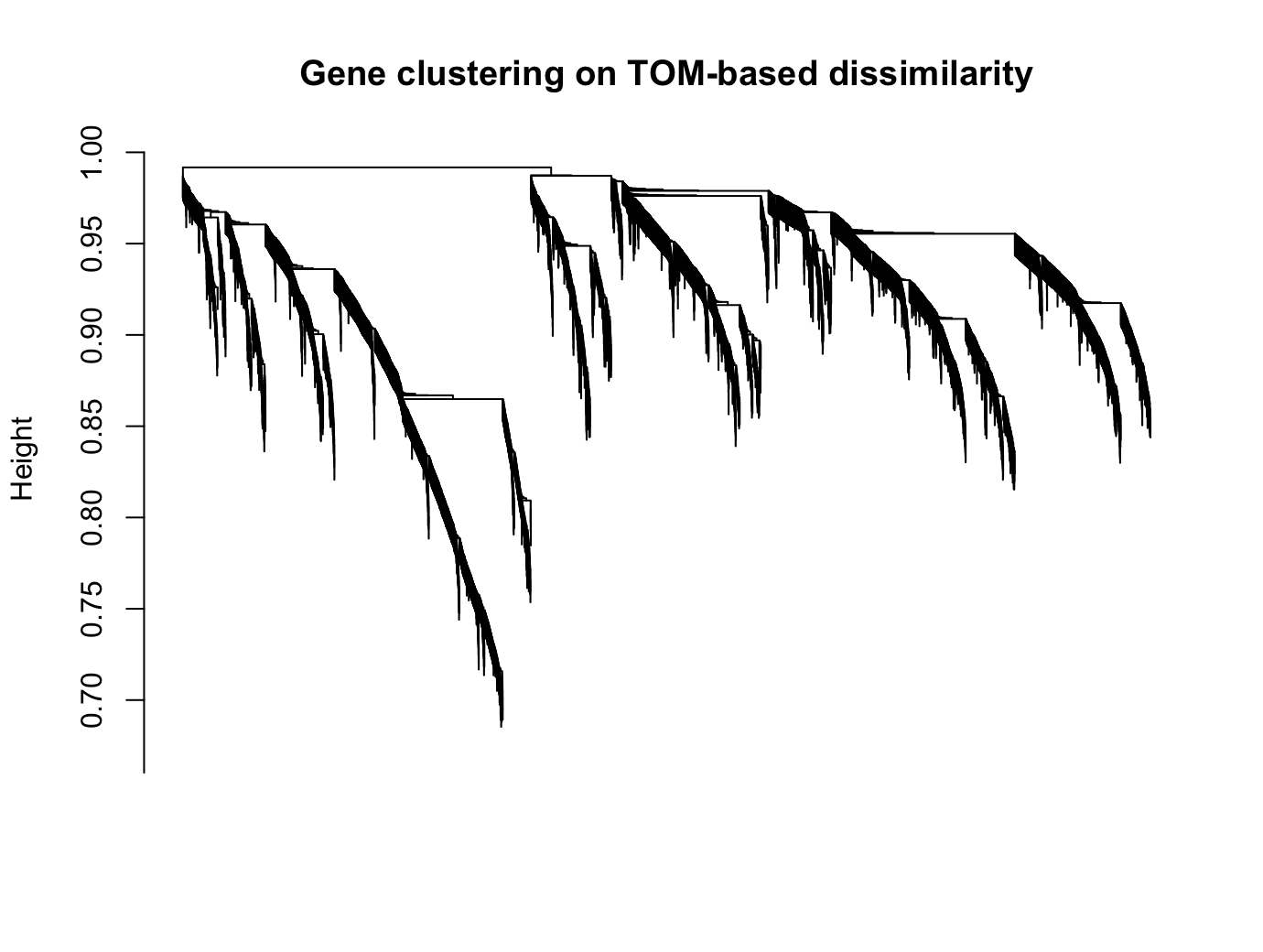


Figure 5. Clustering tree: The figure shows the clustering tree (dendrogram) that results from hierarchical clustering of the TOM-based dissimilarity matrix and will be used for identifying modules of highly similar genes in the co-expression network.

### 

### 2.2 Identify clusters

To cluster genes with similar daily expression pattern, we use the cutreeDynamic() function from the WGCNA package.

We need to provide a minimum size for the identified clusters or modules. This can set depending on the user’s question. In our case, we want to identify fairly large modules that are biologically meaningful (i.e., enriched in different GO/PFAM terms). As such, we set the minimum module size to 30. However, as you will see later, we will refine our cluster identification by merging very similar modules. As such, the choice of minimum module size should not affect cluster identification drastically.

# We like large modules, so we set the minimum module size relatively high:  
minModuleSize = 30;  
  
# Module identification using dynamic tree cut:  
dynamicMods= cutreeDynamic(dendro = geneTree,  
 distM = dissTOM,  
 method = "hybrid",  
 verbose = 4,  
 deepSplit = 3, # see WGCNA for more info on tuning parameters  
 pamRespectsDendro = FALSE,  
 minClusterSize = minModuleSize);

## ..cutHeight not given, setting it to 0.99 ===> 99% of the (truncated) height range in dendro.  
## ..Going through the merge tree  
##   
## ..Going through detected branches and marking clusters..  
## ..Assigning Tree Cut stage labels..  
## ..Assigning PAM stage labels..  
## ....assigned 5531 objects to existing clusters.  
## ..done.

# Convert numeric lables into colors  
dynamicColors = labels2colors(dynamicMods)  
table(dynamicColors)

## dynamicColors  
## black blue brown cyan darkgreen   
## 515 1172 742 149 79   
## darkgrey darkorange darkred darkturquoise green   
## 72 56 88 75 720   
## greenyellow grey60 lightcyan lightgreen lightyellow   
## 269 112 134 107 99   
## magenta midnightblue orange pink purple   
## 369 134 66 506 299   
## red royalblue saddlebrown salmon skyblue   
## 579 91 45 199 50   
## steelblue tan turquoise white yellow   
## 32 265 1337 55 723

**In the initial cluster (module) identification step, WGCNA finds 30 modules. However, some of the identified modules might have very similar expression pattern and we would rather merge this closely related modules into one.**

We do that in the next step.

### 2.3 Merge similar modules

# Calculate eigengenes  
MEList = moduleEigengenes(datExpr, colors = dynamicColors)  
MEs = MEList$eigengenes  
  
# Calculate dissimilarity of module eigengenes  
MEDiss = 1-cor(MEs, method = "kendall");  
  
# Cluster module eigengenes  
METree = hclust(as.dist(MEDiss), method = "average");  
# Plot the result  
# sizeGrWindow(7, 8)  
plot(METree, main = "Clustering of module eigengenes",  
 xlab = "", sub = "MEDiss = 1-cor(MEs, method = 'kendall')")  
  
# We choose a height cut of 0.2, corresponding to correlation of 0.8, to merge  
MEDissThres = 0.2 # user-specified parameter value; see WGCNA manual for more info  
  
# Plot the cut line into the dendrogram  
abline(h=MEDissThres, col = "red")

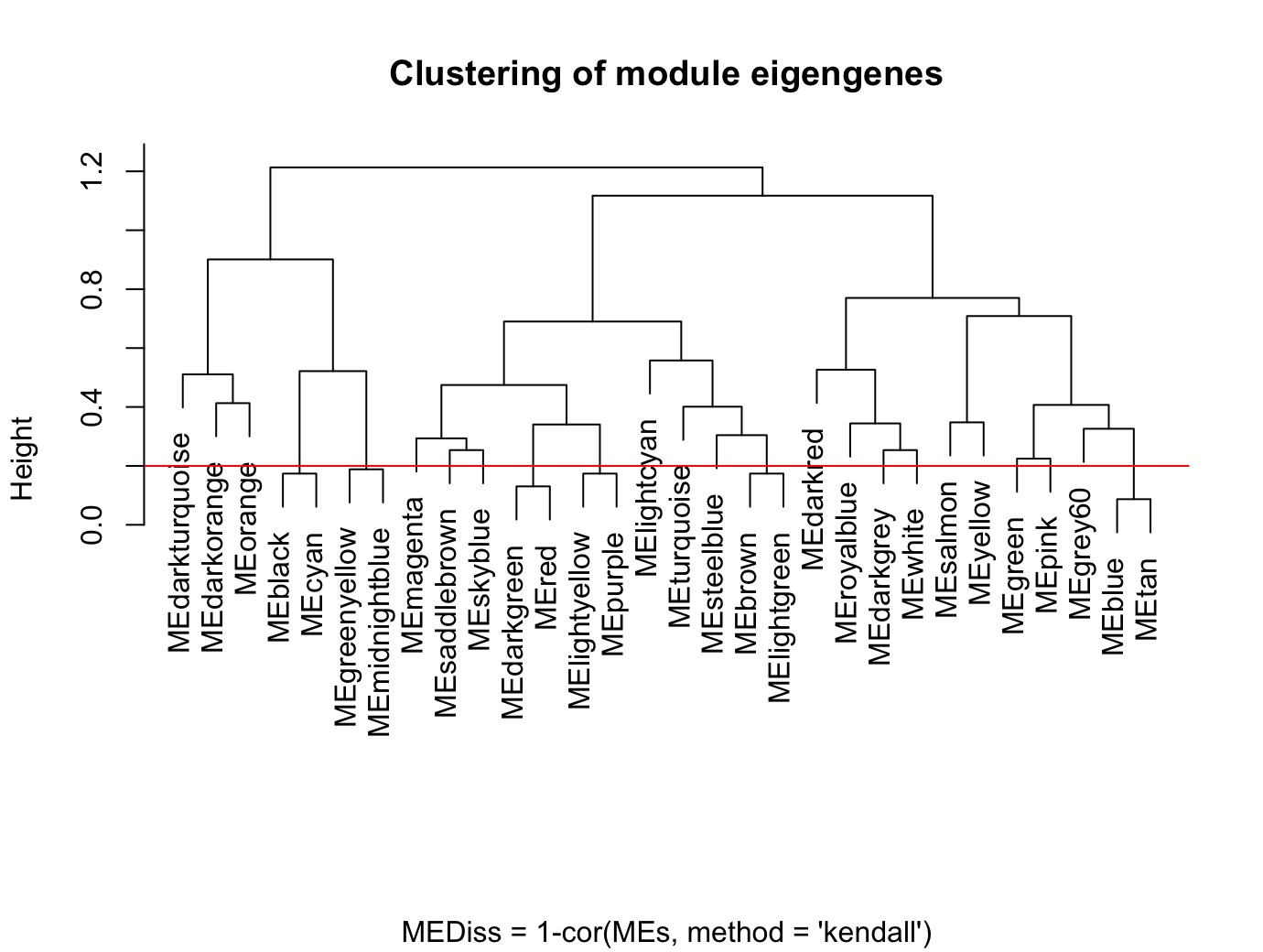


Figure 6. Merging similar modules: The above figure shows the similarity of the different gene modules using hierarchical clustering of the module’s eigenvalue (eigengene expression). The horizontal red line shows the cutoff used to merge similar modules.

We choose a cut height of 0.2, corresponding to correlation of 0.8, to merge similar modules. Although arbitrary, the cutoff was motivated by the number of modules we would like to retain in the GCN; in our case, a 0.2 threshold resulted in a total of 12 modules in the GCN (see below).

In the following code, we merge the similar modules and visualize the module assignments before and after merging.

# Call an automatic merging function  
merge = mergeCloseModules(datExpr, dynamicColors, cutHeight = MEDissThres, verbose = 3)

## mergeCloseModules: Merging modules whose distance is less than 0.2  
## multiSetMEs: Calculating module MEs.  
## Working on set 1 ...  
## moduleEigengenes: Calculating 30 module eigengenes in given set.  
## multiSetMEs: Calculating module MEs.  
## Working on set 1 ...  
## moduleEigengenes: Calculating 13 module eigengenes in given set.  
## multiSetMEs: Calculating module MEs.  
## Working on set 1 ...  
## moduleEigengenes: Calculating 12 module eigengenes in given set.  
## Calculating new MEs...  
## multiSetMEs: Calculating module MEs.  
## Working on set 1 ...  
## moduleEigengenes: Calculating 12 module eigengenes in given set.

# The merged module colors  
mergedColors = merge$colors;  
# Eigengenes of the new merged modules:  
mergedMEs = merge$newMEs;  
  
# sizeGrWindow(12, 9)  
plotDendroAndColors(geneTree,  
 cbind(dynamicColors, mergedColors),  
 c("Dynamic Tree Cut", "Merged dynamic"),  
 dendroLabels = FALSE, hang = 0.03,  
 addGuide = TRUE, guideHang = 0.05)

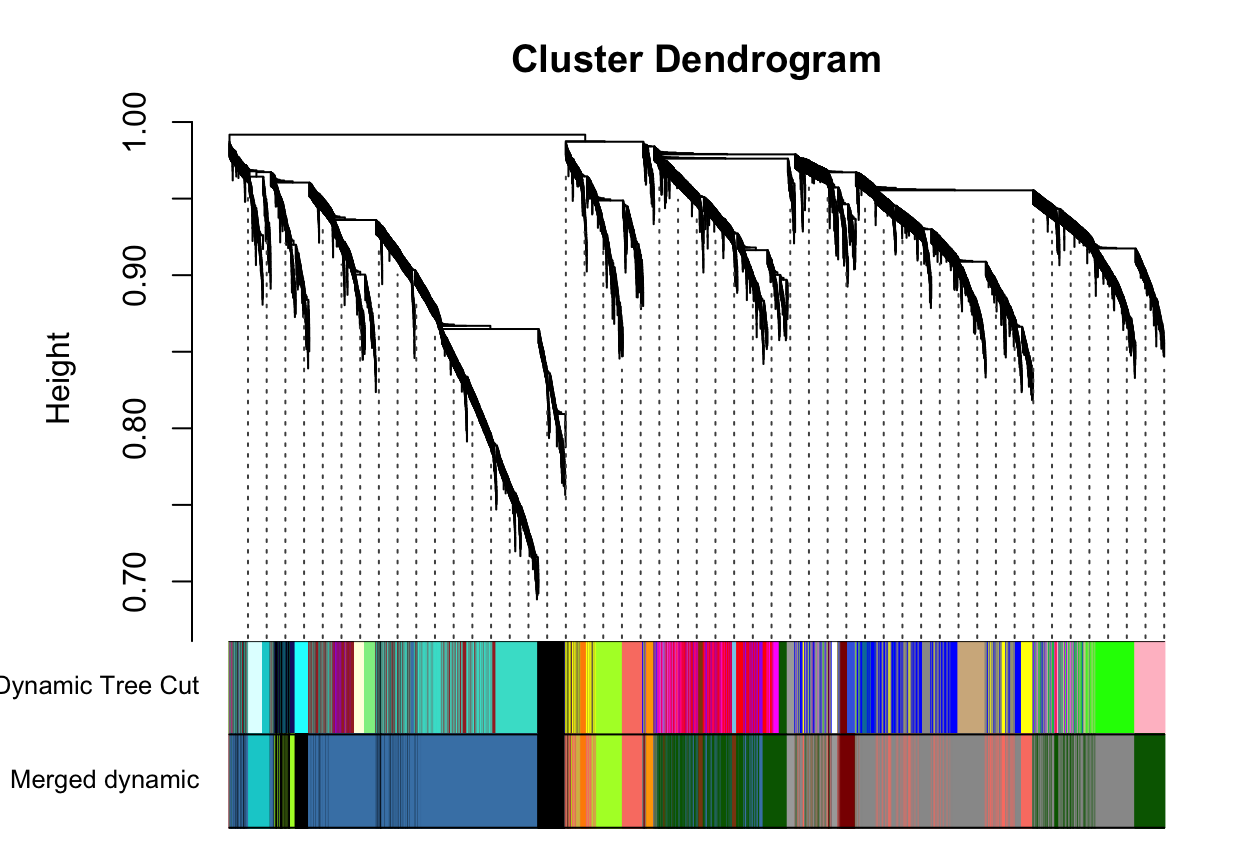
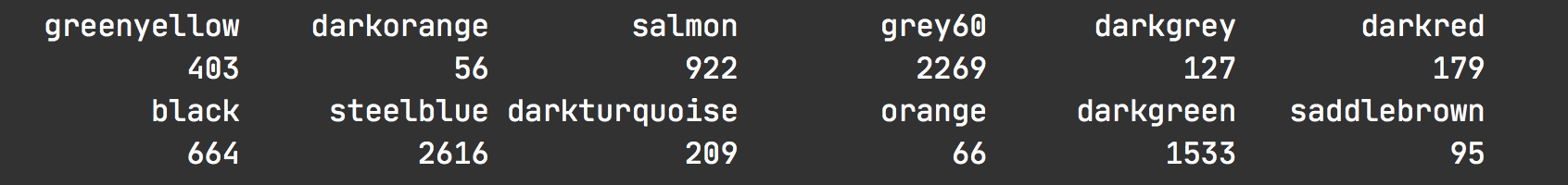


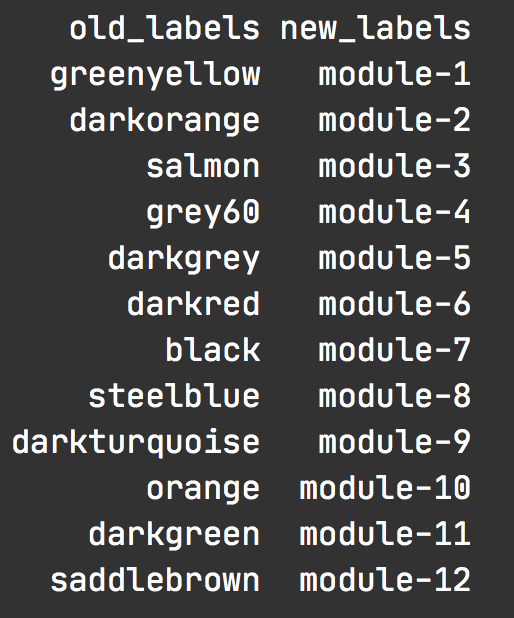
Figure 7. Modules of highly co-expressed genes: The above plot shows the results of module identification before (Dyanamic Tree Cut) and after (Merged dyanamic) similar modules were merged.

# Rename to moduleColors  
moduleColors = mergedColors  
  
# Construct numerical labels corresponding to the colors  
colorOrder = c("grey", standardColors(50));  
moduleLabels = match(moduleColors, colorOrder)-1

**We identified 12 modules in the ant GCN, the size of each of these modules are shown below.**



Note, WGCNA names the different modules as colors (see above), and the colors have no meaning. Therefore, it might be useful to rename the modules. In the next step we rename all the modules according to the following convention:



### 2.4 Calculate module-module similarity

Thus far, we have created the ant GCN (adjacency matrix) and identified 12 modules of highly co-expressed genes in the network.

Next, we investigate how the different modules are connected to each other in the GCN. To do so, we calculate the module-module similarity (Kendall’s tau-b correlation for pairwise module-eigengene expression) and then use the similarity matrix to create the module adjacency matrix.

The following code calculates the module adjacency matrix and visualizes it as a heatmap.

# Calculate similarity of the eigen-genes  
sim\_matrix\_ME <- cor(mergedMEs, method = "kendall")  
  
# calculate adj\_matrix  
adj\_matrix\_ME <- adjacency.fromSimilarity(sim\_matrix\_ME,  
 power=1, # DO NOT power transform  
 type='signed'  
)  
  
# coerce into a matrix  
  
## GET THE NAMES OF THE MODULES  
# module\_ids <- rownames(adj\_matrix\_ME)  
## CHANGE THE NAMES OF THE MODULES  
module\_ids <- data.frame(old\_labels = rownames(adj\_matrix\_ME),  
 new\_labels = paste0("module-", 1:nrow(adj\_matrix\_ME)))  
  
adj\_matrix\_ME <- matrix(adj\_matrix\_ME, nrow=nrow(adj\_matrix\_ME))  
rownames(adj\_matrix\_ME) <- module\_ids$new\_labels  
colnames(adj\_matrix\_ME) <- module\_ids$new\_labels

gplots::heatmap.2(t(adj\_matrix\_ME),  
 col=inferno(100),  
 # labRow=NA, labCol=NA,  
 trace='none', dendrogram='row',  
 xlab='', ylab='',  
 # main='Similarity matrix - MEs \n correlation method = "kendall")',  
 main='Adjacency matrix - MEs \n modified edge weights)',  
 density.info='none', revC=TRUE)

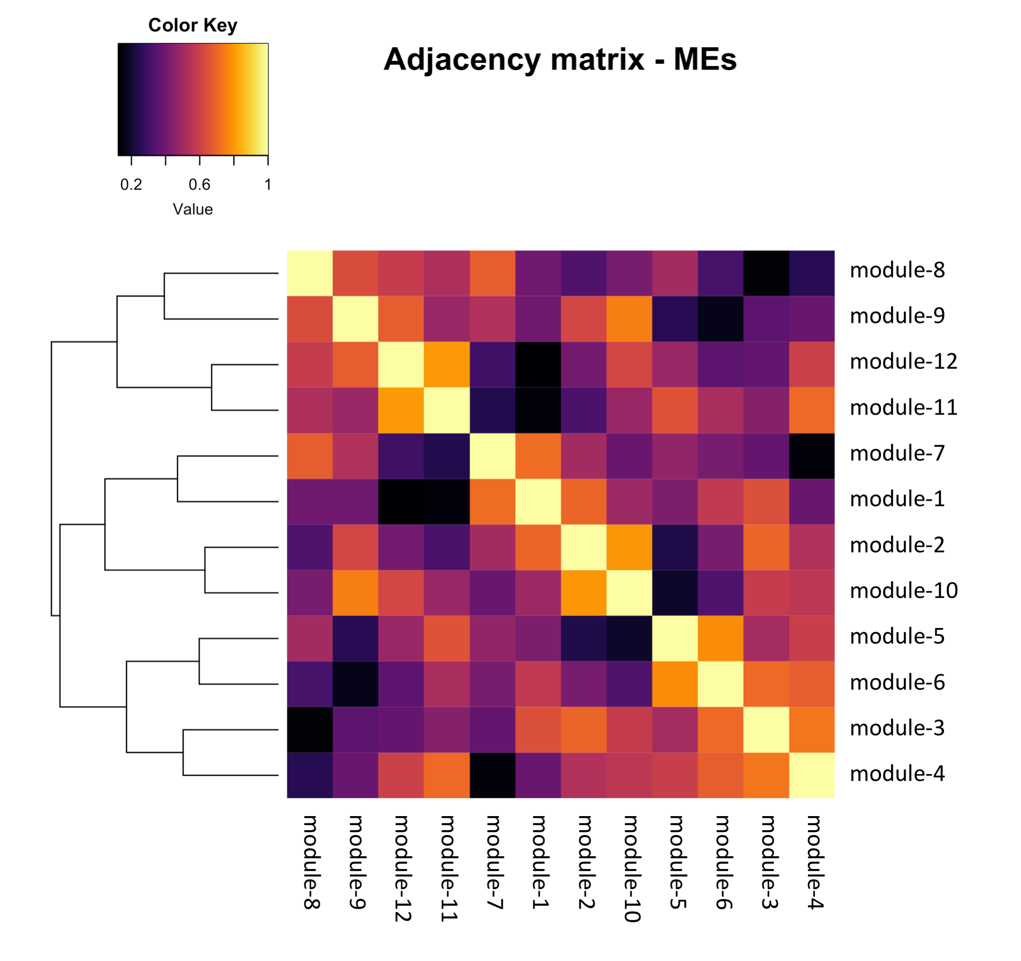


Figure 8. Module-module relationships: The heatmap above shows the pairwise Kendall’s tau correlation (similarity) of the twelve modules identified in the ant GCN. Darker shades indicate low correlations and brighter shades indicate high correlations, as shown in the Color Key.

### 2.5 Visualize the network

To better visualize the global network – how the modules are connected to each other – we can simplify the network. That is, we remove most of the weak edges of the network and retain only the strong module-module correlations.

For example, to remove weak edges, we can set all correlations less than 0.6 to be zero. This will help us obtain a fairly clean network for visualization. To simplify further, we can assign the same edge weight for all correlations between 0.6 and 0.8 (e.g., 0.5), and a different edge weight for correlations ≥ 0.8 (e.g., 1).

The following code uses the igraph package in R to simplify and visualize the module-module relationships in the network.

pacman::p\_load(igraph)  
  
# get rid of low correlations (0.6 & 0.8 are arbitrary)  
adj\_matrix\_ME[adj\_matrix\_ME < 0.6] <- 0  
adj\_matrix\_ME[adj\_matrix\_ME < 0.8 & adj\_matrix\_ME>0] <- 0.5  
adj\_matrix\_ME[adj\_matrix\_ME >= 0.8] <- 1  
  
# build\_network  
network <- graph.adjacency(adj\_matrix\_ME,  
 mode = "upper",  
 weighted = T,  
 diag = F)  
  
# simplify network  
network <- igraph::simplify(network) # removes self-loops

colors <- mergedMEs %>% names() %>% str\_split("ME", 2) %>% sapply("[", 2)  
V(network)$color <- colors  
  
genes\_ME <- factor(moduleColors, levels=colors) %>% summary()  
V(network)$size <- log2(genes\_ME)\*2  
  
V(network)$label.color <- "black"  
V(network)$frame.color <- "white"  
  
E(network)$width <- E(network)$weight^2\*4  
E(network)$edge.color <- "gray80"  
  
  
## Circular layout  
plot(network,  
 layout=layout.kamada.kawai,  
 vertex.shape="none"  
)

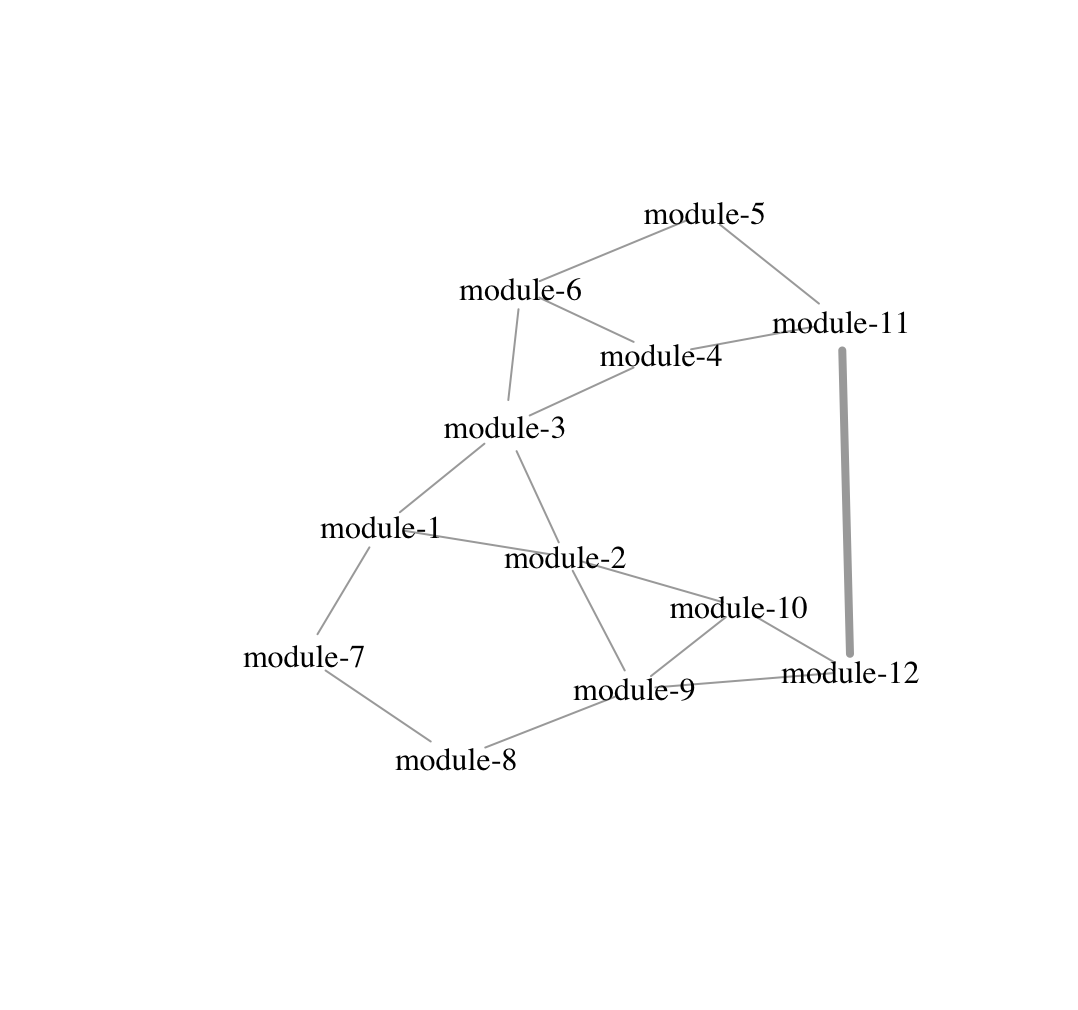


Figure 9. Visualizing the ant GCN: A simplified view of the connectivity patterns between the different gene modules of the ant GCN are shown. In our case, thick edges between two modules indicate correlations ≥ 0.8, thinner edges indicate correlations between (0.6, 0.8), and no edges indicate correlations < 0.6.

## Step 3: Annotate the network

Now that we have created the ant GCN, we can functionally annotate the network by identifying which modules contain our genes of interest. To do so, we will check for significant overlap between a module in the network and our genes of interest using Fisher’s exact test.

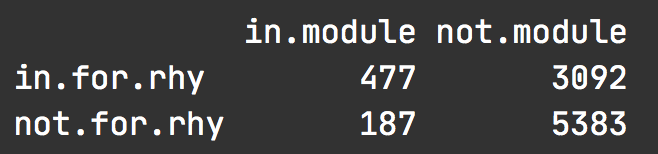
### 3.1 Define your genes of interest

For example, we want to identify the GCN modules that contain our 24h oscillating genes (for.rhy = 24h-rhythmic genes in forager brains, nur.rhy = 24h-rhythmic genes in nurses).

# DEFINE GENES OF INTEREST  
  
rhy.trait.24 <- tbl(db, "ejtk\_all") %>% select(gene\_name:rhy) %>% collect()  
# pull the genes  
for.rhy <- rhy.trait.24 %>% filter(caste=="for" & rhy=="yes") %>% pull(gene\_name)  
nur.rhy <- rhy.trait.24 %>% filter(caste=="nur" & rhy=="yes") %>% pull(gene\_name)  
  
rhy.trait.8 <- tbl(db, "ejtk\_8h\_all") %>% select(gene\_name:rhy) %>% collect()  
for.rhy.8 <- rhy.trait.8 %>% filter(caste=="for" & rhy=="yes") %>% pull(gene\_name)  
nur.rhy.8 <- rhy.trait.8 %>% filter(caste=="nur" & rhy=="yes") %>% pull(gene\_name)  
  
rhy.trait.12 <- tbl(db, "ejtk\_12h\_all") %>% select(gene\_name:rhy) %>% collect()  
for.rhy.12 <- rhy.trait.12 %>% filter(caste=="for" & rhy=="yes") %>% pull(gene\_name)  
nur.rhy.12 <- rhy.trait.12 %>% filter(caste=="nur" & rhy=="yes") %>% pull(gene\_name)

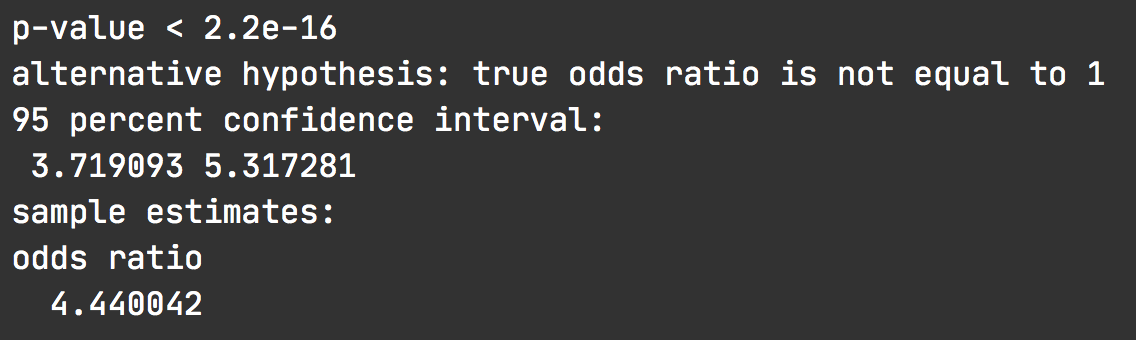
Now that we have defined our genes of interest (rhythmic genes), we can use a Fisher’s exact test to check for significant overlap between our genes of interest (e.g., for.rhy) and modules in the GCN (e.g., module-7) in a pairwise manner.

For example, to check if the 24h-rhythmic genes in forager brains (for.rhy) are overrepresented in module-7, we first create the contingency table. To do so, we need to define the background number of genes, i.e., the size of the set that contains all possible genes from which module-7 and for.rhy genesets are drawn. In our case, that would be the 9139 genes that was used to build the GCN. Using this information, we can create the contingency table as shown below.

****

The above table shows that 477 genes are found in both, for.rhy and module-7. However, there are 3092 genes that are found in for.rhy but not in module-7, whereas 187 genes occur only in module-7 but not in for.rhy. Finally, we have 5383 genes that are in the background geneset but neither in rhy.for nor in module-7.

Now, we can run the Fisher’s exact test using the fisher.test() function in R. The results of which are shown below:



The output shows that the odds-ratio is approximately 4, which is significantly higher than 1 (p-value < 2e-16). In other words, the genes that show 24h-rhythms in forager brains are significantly overrepresented in module-7 and vice-versa, or that the two sets show significant overlap.

### 3.2 Where are my genes of interest located?

Since we need to perform multiple Fisher’s exact test for our comparisons, we will make use of the GeneOverlap package in R [3].

pacman::p\_load(GeneOverlap)  
# https://www.bioconductor.org/packages/devel/bioc/vignettes/GeneOverlap/inst/doc/GeneOverlap.pdf  
  
# Make a list that returns gene names for a given cluster  
module\_color = colors  
module = names(mergedMEs)  
module\_colors <-  
 data.frame(module\_label=module) %>%  
 mutate(module\_color = str\_replace(module\_label, "ME", ""))  
  
module\_genes <- list()  
module\_color <- module\_colors$module\_color  
# Get the genes from each of the modules  
for (i in 1:length(module\_color)) {  
  
 module\_genes[[i]] <- names(datExpr)[which(moduleColors==module\_color[[i]])]  
 names(module\_genes)[[i]] <- module\_color[[i]]  
}  
# change the name of the modules  
names(module\_genes) <- module\_ids$new\_labels

## MAKE YOUR LIST OF GENES OF INTEREST ##  
  
# LIST ONE - WGCNA modules  
list1 <- module\_genes  
sapply(list1, length)

## module-1 module-2 module-3 module-4 module-5 module-6 module-7 module-8   
## 403 56 922 2269 127 179 664 2616   
## module-9 module-10 module-11 module-12   
## 209 66 1533 95

## LIST TWO - rhythmic genes  
list2 <- list(for.rhy, nur.rhy, for.rhy.12, nur.rhy.12, for.rhy.8, nur.rhy.8)  
names(list2) <- c("for24", "nur24", "for12","nur12", "for8", "nur8")  
sapply(list2, length)

## for24 nur24 for12 nur12 for8 nur8   
## 3569 1367 148 193 229 550

## CHECK FOR OVERLAP  
## make a GOM object  
gom.1v2 <- newGOM(list1, list2,  
 genome.size = nGenes)  
  
drawHeatmap(gom.1v2,  
 adj.p=T,  
 cutoff=0.05,  
 what="odds.ratio",  
 # what="Jaccard",  
 log.scale = T,  
 note.col = "grey80")

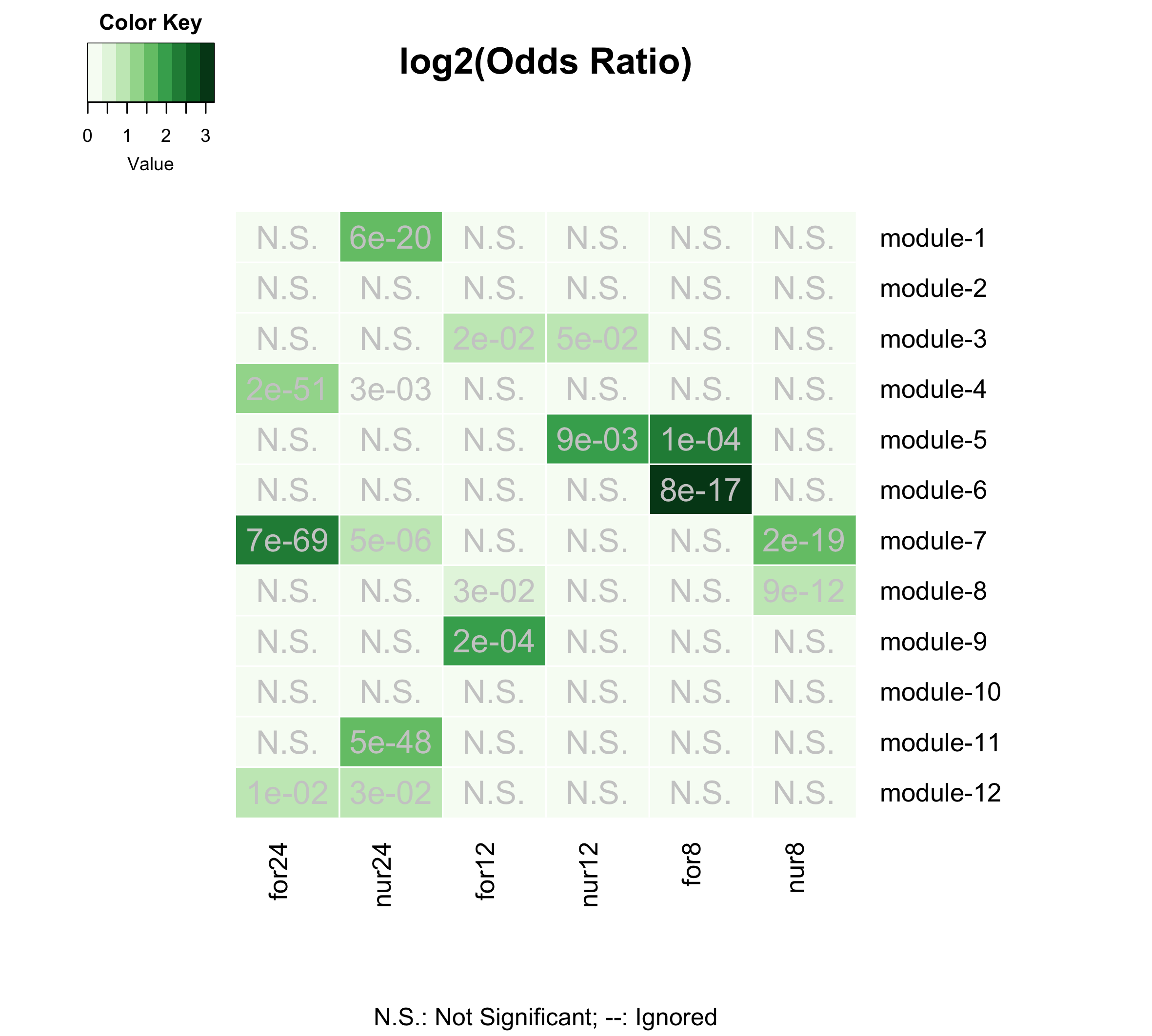


Figure 10. Gene-clusters with rhythmic genes: The above plot shows the result of the Fisher’s exact test performed for each module-geneset pair. The color of the boxes represents the odds-ratio (darker the green, higher is the odds-ratio) and the p-values are shown. The p-values shown were corrected for multiple-hypothesis testing using Benjamini-Hochberg method. Non-significant overlaps between module and geneset are indicated with a N.S. inside the box.

From the plot, we can see that the 24h-rhythmic genes are located in five of the twelve modules of the ant GCN (module-1, module-4, module-7, module-11, and module-12).

We can further annotate the rhythmic modules by identifying which of these five modules peak during the day and which peak at night in ant brains. For the purpose of this study, we focus on the daily expression patterns only in the forager brain because the *Ophiocordyceps* are most likely to infect forager ants that are perform bulk of the outside-nest tasks. To identify day- and night-peaking modules, we can visualize the daily expression of all genes in these rhythmic modules as well as their module eigengene expression (shown below).

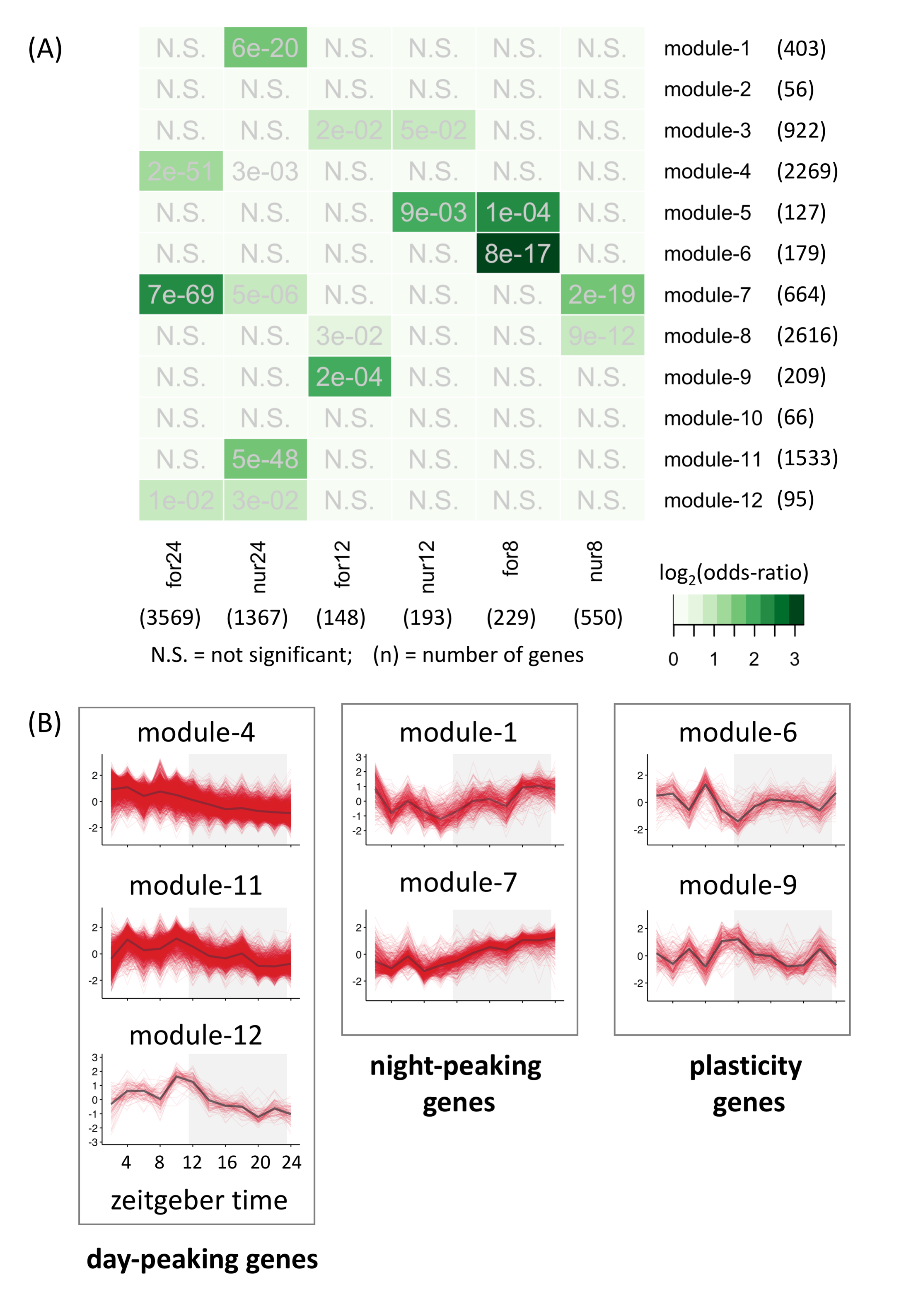


Figure 11. Daily expression patterns of genes in rhythmic modules: The daily expression pattern of all genes in a given module as well as the module’s eigengene expression are shown. For a module, each red line represents the expression of one gene and the black line represents the eigengene expression. The x-axis shows the time-of-day or Zeitgeber Time whereas the y-axis shows normalized gene expression (z-scores calculated from log2-transformed expression data). The 12h:12h light-dark cycles during which the samples were collected are also shown; white background indicates the light phase (lights on at ZT24/ZT0) and grey background indicates the dark phase (lights turned off at ZT12).

**We found that module-4, module-11, and module-12 were day-peaking modules, whereas module-1 and module-7 were night-peaking modules.**

Using the same approach as above, we can identify the ant modules that putatively underlie behavioral plasticity, as well as the modules that are affected during *Ophiocordyceps*-induced behavioral manipulation.

## Genes underlying behavioral plasticity  
 ## DEGS (foragers v. nurses)  
 # genes higher expressed in forager brains (v. nurse brains)  
 for.up <- tbl(db, "TC5\_DEGs\_all") %>% filter(upregulation=="for") %>% collect() %>% pull(gene\_name)  
 # genes lower expressed in for. brains (v. nurse brains)  
 for.down <- tbl(db, "TC5\_DEGs\_all") %>% filter(upregulation=="nur") %>% collect() %>% pull(gene\_name)  
  
## Genes underlying parasite-induced behavioral manipulation  
 ## DEGs (ophio-ant v. control-ant)  
 ophio.dat <- tbl(db, "ophio\_biting\_control") %>% collect() %>% select(gene, value\_1, value\_2, q\_value:logFC)  
 ophio.dat <- ophio.dat %>%  
 filter(abs(logFC) >= 1 & significant=="yes" & q\_value < 0.05) %>%  
 mutate(ophio = ifelse(logFC > 0, "down", "up"))  
 # genes higher expressed in ant heads during Ophio-manipulated biting (v. controls)  
 ophio.up <- ophio.dat %>% filter(ophio=="up") %>% pull(gene)  
 # genes lower expressed in ant heads during manipulated biting (v. controls)  
 ophio.down <- ophio.dat %>% filter(ophio=="down") %>% pull(gene)  
  
## LIST THREE - genes underlying behavioral plasticity and parasitic behavioral manipulation  
list3 <- list(for.up, for.down, # same as list three  
 ophio.up, ophio.down)  
names(list3) <- c("for-UP", "for-DOWN",  
 "ophio-UP", "ophio-DOWN")  
  
## CHECK FOR OVERLAP  
  
## make a GOM object  
gom.1v3 <- newGOM(list1, list3,  
 genome.size = nGenes)  
## visualize the overlaps  
drawHeatmap(gom.1v3,  
 adj.p=T,  
 cutoff=0.05,  
 what="odds.ratio",  
 # what="Jaccard",  
 log.scale = T,  
 note.col = "grey80")

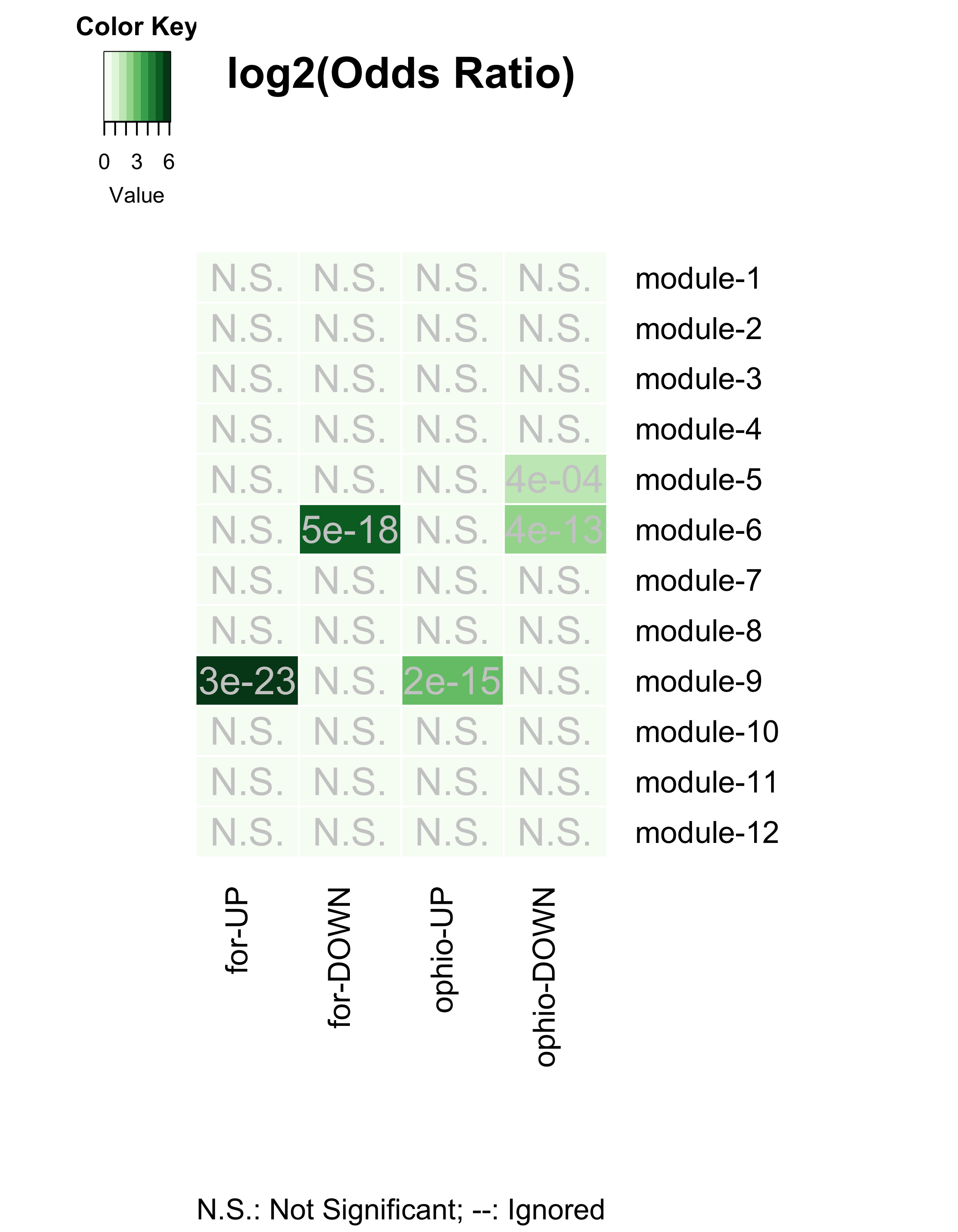


Figure 12. Gene-clusters behavioral plasiticity and parasitic behavioral manipulation: The heatmap identifies the different gene modules that show an overrepresentation of the genes previously found to underlie behavioral plasticity (genes differentially expressed between foragers and nurses) and parasitic behavioral manipulation (genes differentially expressed in foragers during manipulated biting behavior).

The figure above shows that the gene-modules that putatively underlie behavioral plasticity (forager-nurse differentiation) and the ones that are affected during *Ophiocordycpes*-induced behavioral manipulation, are the same.

**In other words, to induce the characteristic manipulated biting behavior, the manipulating fungal parasite seems to be targeting the same genes and processes that otherwise allow ants to display behavioral plasticity (shown in the summary figure below).**

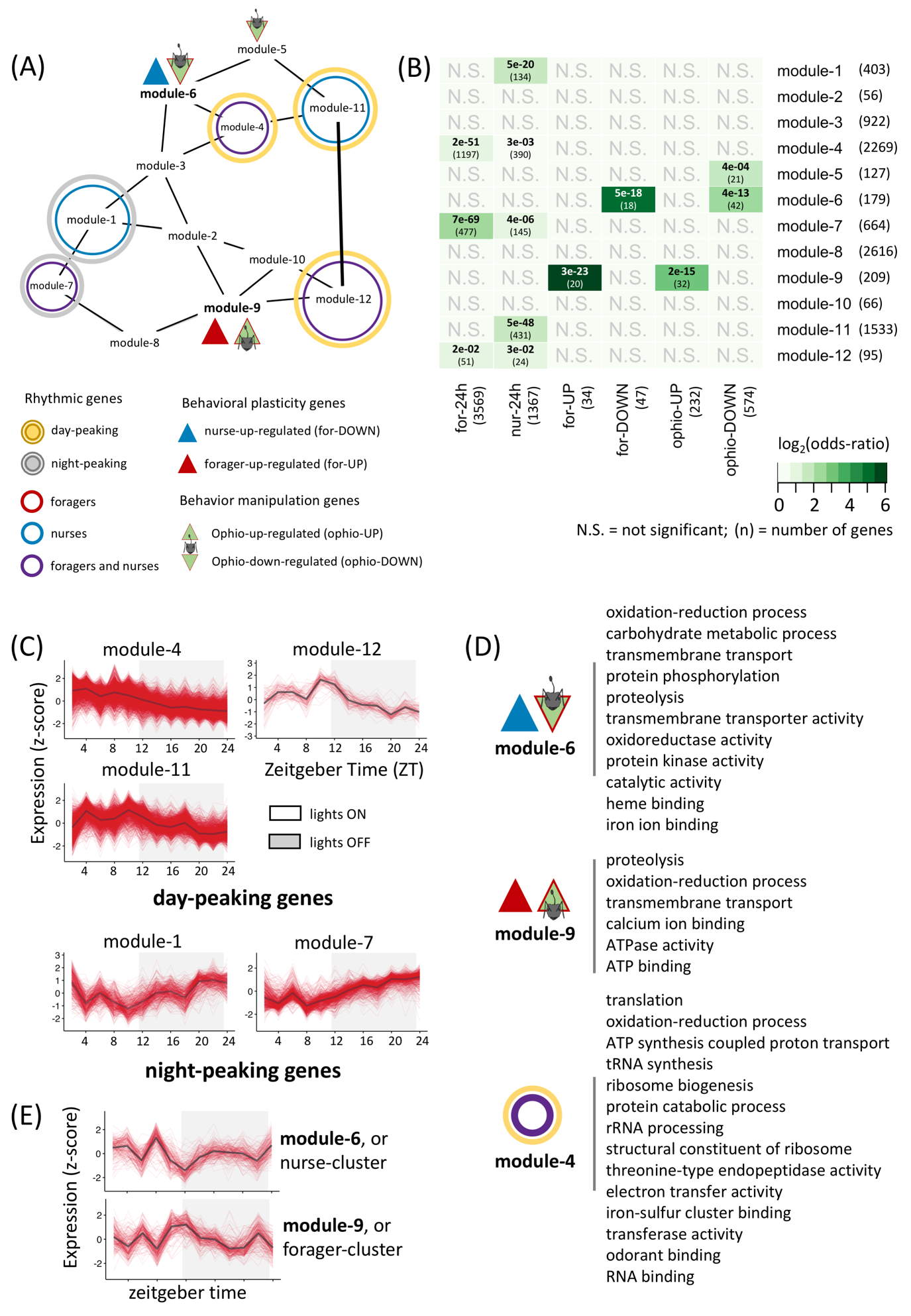
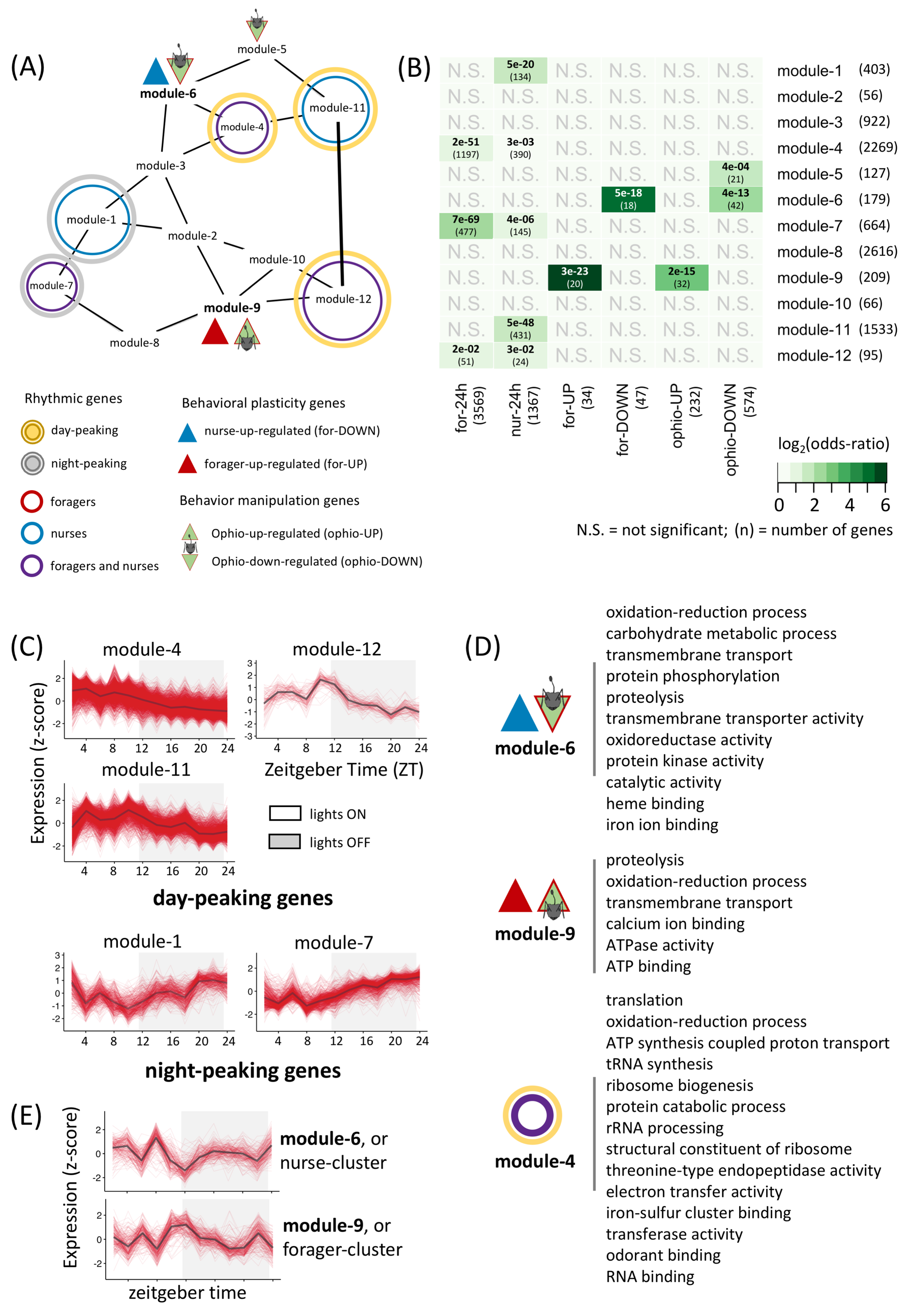


Figure 13. Annotated ant GCN: The above figure summarizes the findings from our analysis and identifies different modules of interest in the ant GCN that are important for the interplay of rhythmicity, behavioral plasticity and parasitic behavioral manipulation.

## REFERENCES

1. Hutchison AL, Allada R, Dinner AR: **Bootstrapping and empirical bayes methods improve rhythm detection in sparsely sampled data.** *J Biol Rhythms* 2018, **33:**339-349.

2. Langfelder P, Horvath S: **WGCNA: an R package for weighted correlation network analysis.** *BMC Bioinformatics* 2008, **9:**559.

3. Shen L: **GeneOverlap: An R package to test and visualize gene overlaps.** *R Package* 2014.