Computational Methods Accompanying "Hijacking time: How *Ophiocordyceps* fungi could be using ant clocks to manipulate behavior"

Overview/Goals

This document provides a step-by-step guide that demonstrates how we:

(1) built a circadian gene co-expression network (GCN),

(2) functionally annotated the network using previously published data, and

(3) inferred functions of gene-clusters-of-interest

to answer which modules, if any, of the ant host's gene expression network are seemingly affected by *Ophiocordyceps* during manipulation, and (2) if the fungus targets modules in the host network that are under clock control or drive behavioral plasticity.

Step 1: Build circadian GCN

1.1 Load data

We have built a circadian GCN for the ant *Camponotus floridanus* using previously published time course RNASeq data (1). 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).

Prior to conducting the analyses listed below, we performed the usual steps (i.e., reads trimming, reads mapping to the genome, and normalizing mapped reads) to obtain the normalized gene expression data for each gene in the genome, at each time point, throughout the 24h day.

To build the GCN, we have organized the processed data into a [gene-expr X time-point] format, in a chronological order, as shown below.

```
# A tibble: 6 x 25
  gene_name     X2F   X4F     X6F     X8F    X10F   X12F    X14F    X16F  X18F    X20F    X22F   X24F     X2N
  <chr>       <dbl> <dbl>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
1 LOC10524… 23.5    18.6  19.9    19.4    17.3    17.8   19.1    20.9    20.0  22.6    22.7    23.8   18.8
2 LOC10524…  0       0     0.0180  0.0915  0       0      0.0246  0.0169  0     0.0163  0.0555  0.147  0.0729
3 LOC10524…  0       0     0       0       0       0      0       0       0     0       0       0      0
4 LOC10524…  0.0691  0     0       0       0       0      0       0       0     0       0       0      0
5 LOC10524…  0       0     0       0       0.0802  0      0       0       0     0       0       0      0
6 LOC10524…  0.713   2.13  0.911   3.50    0.997   0.227  0.759   0.209   2.59  0.396   0.337   0.444  0.237
# … with 11 more variables: X4N <dbl>, X6N <dbl>, X8N <dbl>, X10N <dbl>, X12N <dbl>, X14N <dbl>, X16N <dbl>,
#   X18N <dbl>, X20N <dbl>, X22N <dbl>, X24N <dbl>
```

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

Reading the data into R was done as follows:

```
# 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 (2). However, not all genes are expressed in the ant brain, and some are expressed at very low levels that are not biologically meaningful.

Therefore, we only kept 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 resulted in our cleaned, input data file. The daily expression for these 9139 genes has been used to create the circadian GCN of *Camponotus floridanus*.

1.3 Format data

To create the ant GCN, the expression similarity (co-expression) of different gene pairs needs to be calculated. To do so, we normalized the gene expression data by log2-transformation as demonstrated below (Figure 1).

```
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()
```

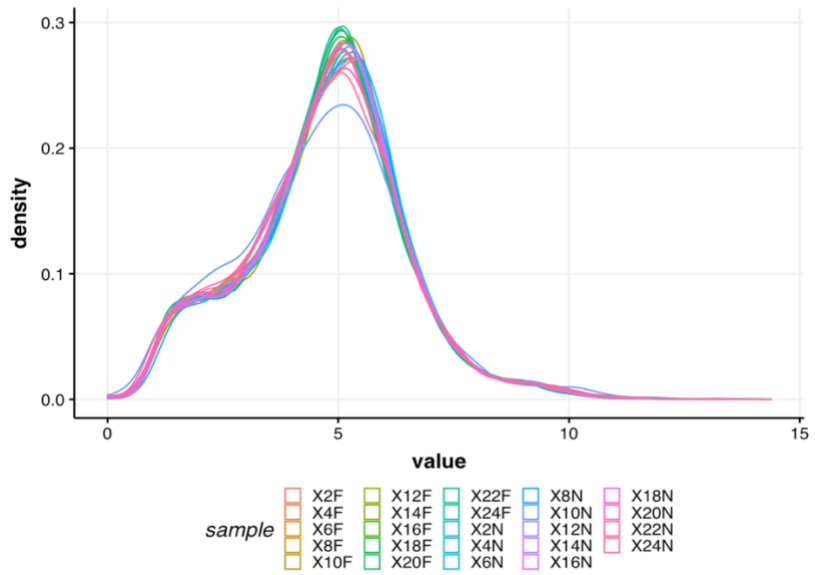

Figure 1. Normalized gene expression: The density plot above shows the distribution of log2-transformed gene expression values for each sample.

1.4 Calculate gene-gene similarity

Subsequently, we calculated the pairwise gene expression similarity for each of the 9139 genes and saved it to a matrix.

We 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 (3)(Figure 2).

```
## 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_nu
r_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 ge
nes)',
          density.info='none', revC=TRUE)
```

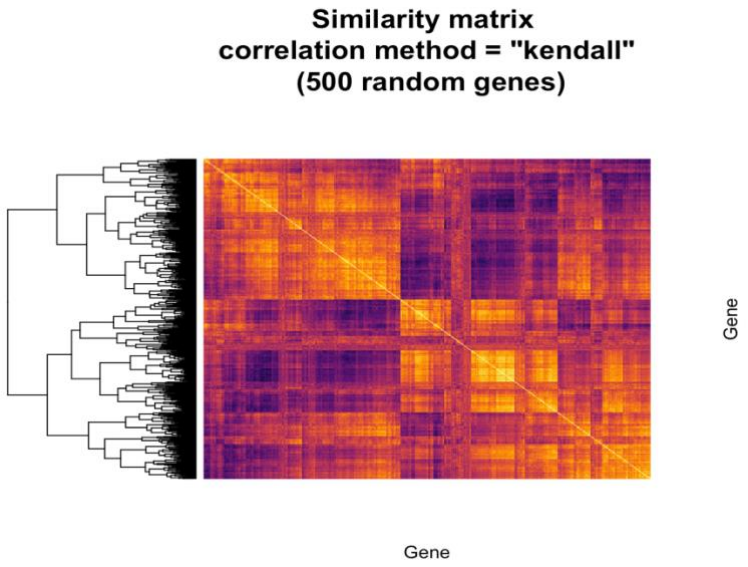

Figure 2. Similarity matrix: The heatmap shows the pairwise Kendall's tau correlation for a set of 500 genes randomly pulled from the 9139 genes expressed in the ant brain.

1.5 Create adjacency matrix

From the above similarity matrix, we created the adjacency matrix needed for constructing a gene co-expression network.

To create the adjacency matrix, we first identified the soft-thresholding power by calling the network topology analysis function from the WGCNA package (4).

```
# 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$fitIndices[,2],
     xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fit,signed R^2",ty
pe="n",
     main = paste("Scale independence"));
text(sft$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")
```

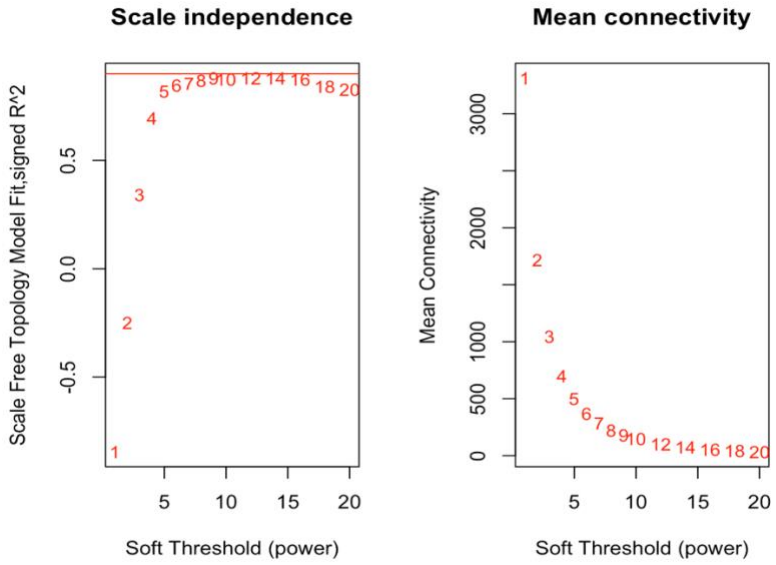

Figure 3. Soft-thresholding power: The plots show the effect of soft-thresholding power on the topology and the mean connectivity of the transformed similarity matrix (network).

The scale-free topology fit index reaches ~0.9 at a soft-thresholding power of 9 without drastically improving beyond that (Figure 3). As such, we set our soft thresholding power to 9 for creating the adjacency matrix (Figure 4).

```
## Specify the soft-thresholding-power
soft.power = 9

## Construct adjacency matrix
# adj_matrix <- adjacency.fromSimilarity(sim_matrix,
#                                        power=soft.power,
#                                        type='signed'
#                                         )
```

```r
# 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)
```

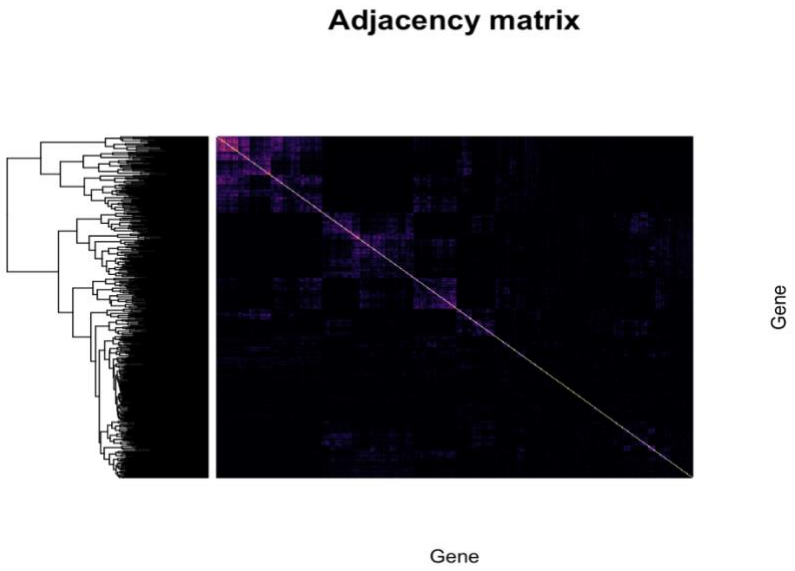

Figure 4. Adjacency matrix: The heatmap shows the result of the power-transformation on the similarity of the 500 random genes shown previously in Figure 2. Only the highest pair-wise correlations are retained whereas the weak correlations tend to zero.

Step 2: Identify gene clusters

2.1 Create topological overlap matrix

To identify clusters of similarly expressed genes, first a topological overlap matrix (TOM) needed to be constructed, which was subsequently used for hierarchical clustering (Figure 5).

```
# 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)
```

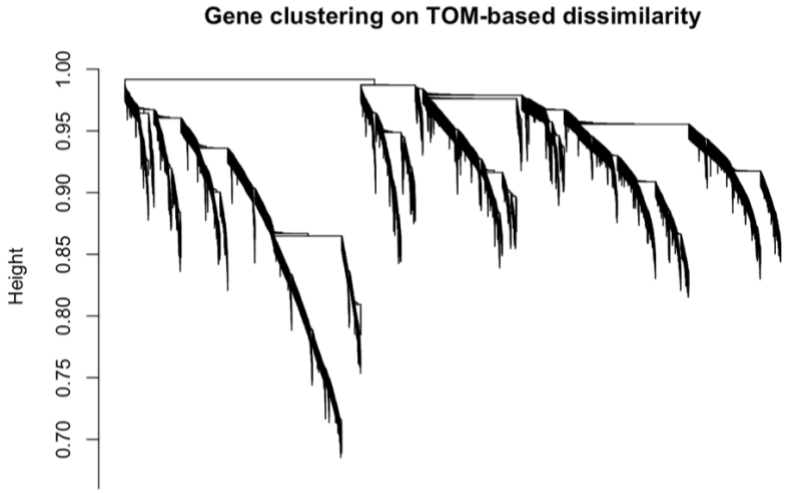

Figure 5. Dendrogram: The depicted clustering tree resulted from hierarchical clustering of the TOM-based dissimilarity matrix and is used for identifying modules of highly similar genes in the co-expression network.

2.2 Identify clusters

To cluster genes with similar daily expression pattern into modules, we used the cutreeDynamic() function from the WGCNA package (4). A minimum size for the identified modules needs to be provided. We aimed to identify only larger modules that are biologically meaningful (i.e., enriched in different GO/PFAM terms). As such, we set the minimum module size to 30. Additionally, cluster identification can be refined at a later stage by merging similar modules. As such, the initial 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 paramet
ers
                            pamRespectsDendro = FALSE,
                            minClusterSize = minModuleSize);
##  ..cutHeight not given, setting it to 0.99  ===>  99% of the (truncated) height ra
nge 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
```

2.3 Merge similar modules

In the initial module identification step, WGCNA found 30 modules. However, some of the identified modules might have very similar expression patterns, which would indicate that they are closely related and warrants their merging. We have done this based on the calculation and hierarchical clustering of the eigengene values of modules (Figure 6).

```
# 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")
```

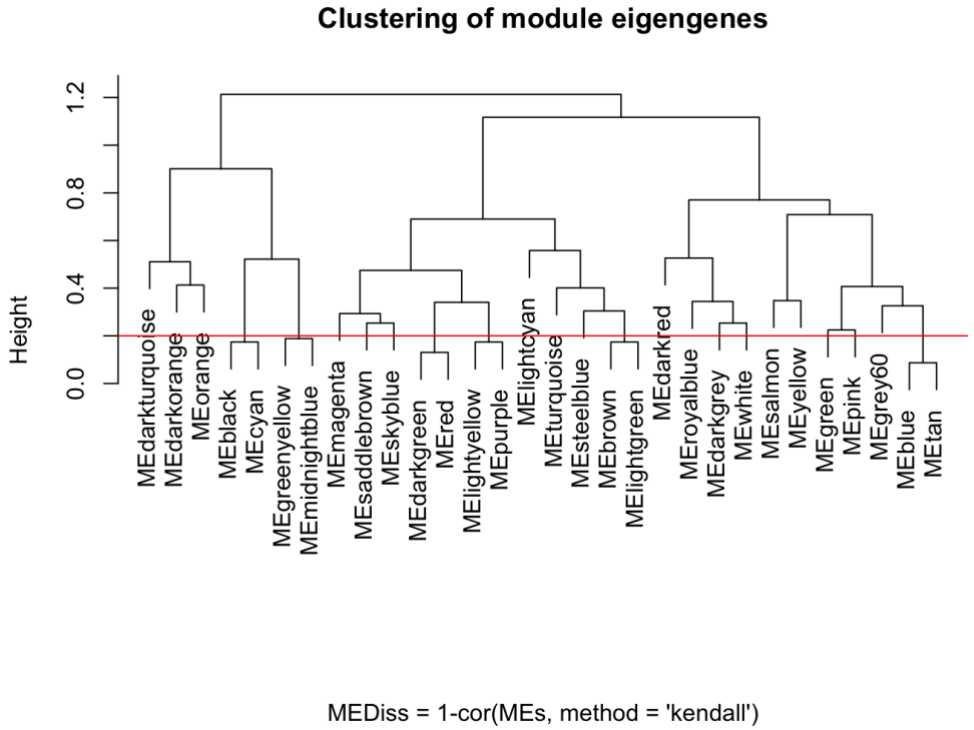

Figure 6. Merging similar modules: The dendrogram shows the similarity of the different gene modules using hierarchical clustering of the module's eigenvalue (eigengene expression). The horizontal red line shows the cutoff used to merge similar modules.

We chose a cut off height of 0.2 (Figure 6), corresponding to correlation of 0.8, to merge similar modules. Although arbitrary, the cutoff was motivated by the number of modules we wanted to retain in the GCN; in our case, a 0.2 threshold resulted in a total of 12 modules in the GCN (see below). After merging the similar modules, we visualized the module assignments before and after merging (Figure 7).

```
# 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)
```

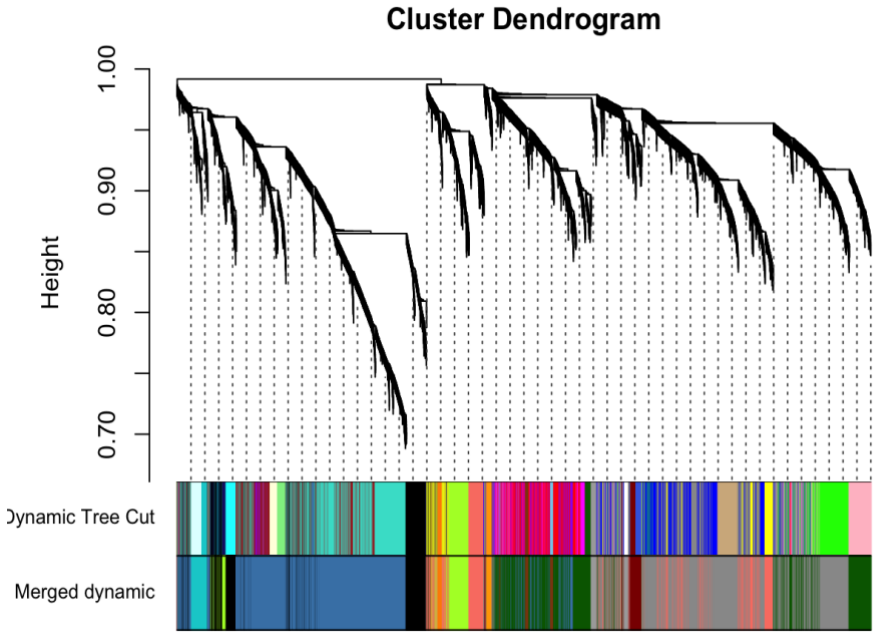

Figure 7. Modules of highly co-expressed genes: The dendrogram and color scheme (representing colors assigned to individual modules) shows the results of module identification before (Dynamic Tree Cut) and after (Merged dynamic) similar modules were merged.

```
# Rename to moduleColors
moduleColors = mergedColors

# Construct numerical labels corresponding to the colors
```

```
colorOrder = c("grey", standardColors(50));
moduleLabels = match(moduleColors, colorOrder)-1
```

Using the method, we identified 12 modules in the ant GCN, the size of each of these modules are shown below.

greenyellow	darkorange	salmon	grey60	darkgrey	darkred
403	56	922	2269	127	179
black	steelblue	darkturquoise	orange	darkgreen	saddlebrown
664	2616	209	66	1533	95

The standard output of WGCNA names the different modules as colors (see above), which have no specific meaning and can complicate data communication. Therefore, we renamed the modules according to the following convention prior to proceeding:

old_labels	new_labels
greenyellow	module-1
darkorange	module-2
salmon	module-3
grey60	module-4
darkgrey	module-5
darkred	module-6
black	module-7
steelblue	module-8
darkturquoise	module-9
orange	module-10
darkgreen	module-11
saddlebrown	module-12

2.4 Calculate module-module similarity

After we created the ant GCN (adjacency matrix) and identified 12 modules of highly co-expressed genes in the network, we investigated how the different modules are connected to each other in the GCN. We calculated the module-module similarity using Kendall's tau-b correlation for pairwise module-eigengene expression and used the resulting similarity matrix to create the module adjacency matrix (Figure 8).

```
# 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)
```

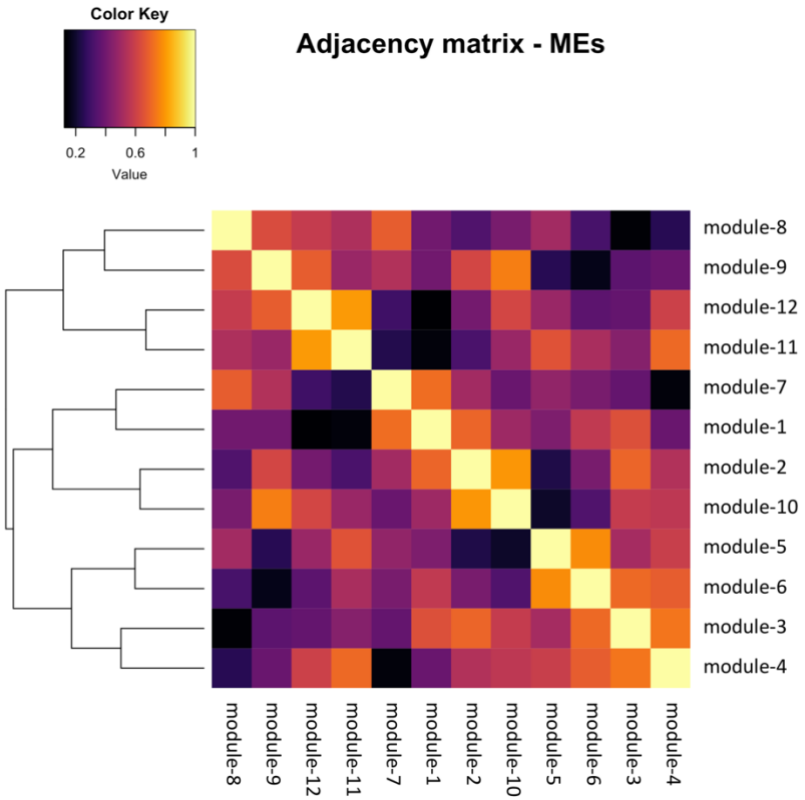

Figure 8. Module-module relationships: The heatmap shows the pairwise Kendall's tau correlation (similarity) of the twelve modules identified in the ant GCN. Darker shades indicate low correlations and brighter shades indicate high correlations, as indicated by the Color Key.

2.5 Visualize the network

To better visualize the global network – how the modules are connected to each other – we simplified the network by removing most of the weak edges of the network and retaining only the strong module-module correlations. To remove weak edges, we set all correlations less than 0.6 to be zero. To simplify further, we assigned the same edge weight for all correlations between 0.6 and 0.8 (i.e., 0.5), and a different edge weight for correlations ≥ 0.8 (i.e., 1). We used the igraph package (5) in R to simplify and visualize the module-module relationships in the network (Figure 9).

```
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"
)
```

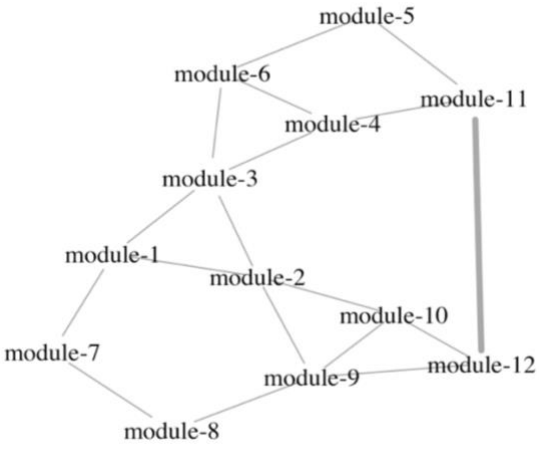

Figure 9. Visualizing the ant GCN: A simplified view of the connectivity patterns between the different gene modules of the ant GCN are shown. In our case, thick edges between two modules indicate correlations ≥ 0.8, thinner edges indicate correlations between (0.6, 0.8), and no edges indicate correlations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

Step 3: Annotate the network

After having created the ant GCN, we functionally annotated the network by identifying which modules contained our genes of interest. As such, we checked for significant overlap between a module in the network and our genes of interest using Fisher's exact tests.

3.1 Define your genes of interest

We first aimed to identify the GCN modules that contained 24h oscillating genes (for.rhy = 24h-rhythmic genes in forager brains, nur.rhy = 24h-rhythmic genes in nurses), 12h oscillating genes (for.rhy.12, nur.rhy.12), and 8h-rhythmic genes (for.rhy.8, nur.rhy.8). Significant transcript rhythmicity has been determined for each gene using empirical JTK-Cycle (6) as reported by Das and de Bekker (2022) in the study from which we obtained the time course dataset that we use here (1) (Supplementary File 2).

```
# 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)
```

We used a Fisher's exact test to ask if there was a significant overlap between genes with significantly rhythmic transcripts (e.g., for.rhy) and the 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) were overrepresented in module-7, we first created a contingency table. To do so, we needed 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 were drawn. In our case, this consisted of the 9139 genes that were used to build the GCN. Using this information, we created the contingency table as shown below.

```
            in.module not.module
in.for.rhy        477       3092
not.for.rhy       187       5383
```

We found that 477 genes were present in both, for.rhy and module-7, genesets. Additionally, 3092 genes were present in for.rhy but not in module-7, whereas 187 genes occurred only in module-7 but not in for.rhy. Finally, 5383 genes of the background geneset were neither in rhy.for nor in module-7.

Having built the contingency table we conducted a Fisher's exact test using the fisher.test() function in R. The results are shown below:

```
p-value < 2.2e-16
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 3.719093 5.317281
sample estimates:
odds ratio
  4.440042
```

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

3.2 Where are my genes of interest located?

To perform multiple Fisher's exact test needed for our comparisons, we made use of the GeneOverlap package in R (7).

```
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")
```

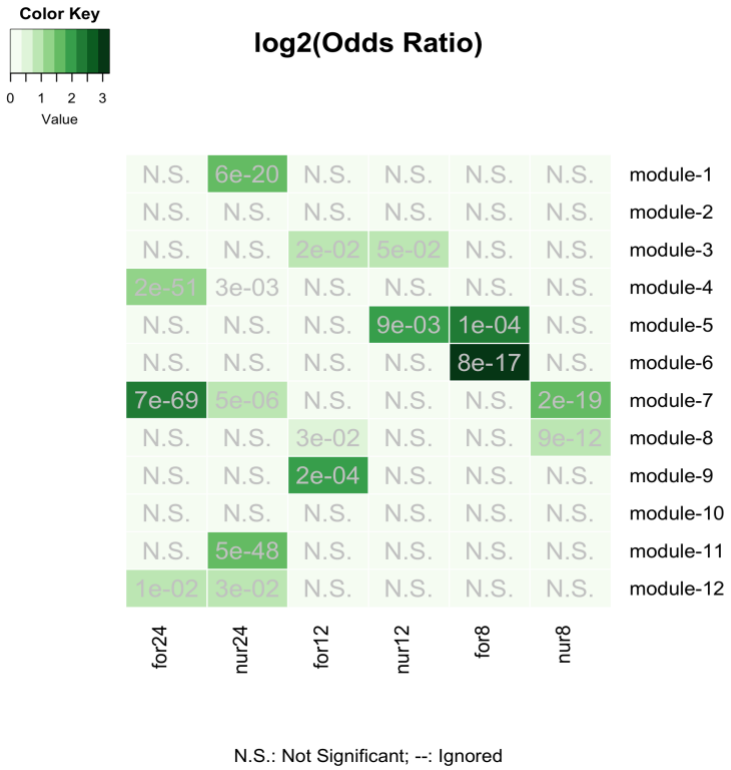

Figure 10. Gene-clusters with rhythmic genes: The matrix shows the results of the Fisher's exact test performed for each module-geneset pair. The color of the boxes represents the odds-ratio (darker the green, higher is the odds-ratio) and the p-values are shown. The p-values were corrected for multiple-hypothesis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

We found that the 24h-rhythmic genes were significantly overrepresented in five of the twelve modules of the ant GCN (module-1, module-4, module-7, module-11, and module-12) (Figure 10).

We further annotated the rhythmic modules by identifying which of these five modules contained genes that peaked during the day- and night-time in ant brains. To identify day- and night-peaking modules, we visualized the daily expression of all genes in the rhythmic modules as well as the module's median gene expression (Figure 11). We found that module-4, module-11, and module-12 were day-peaking modules, whereas module-1 and module-7 were night-peaking modules.

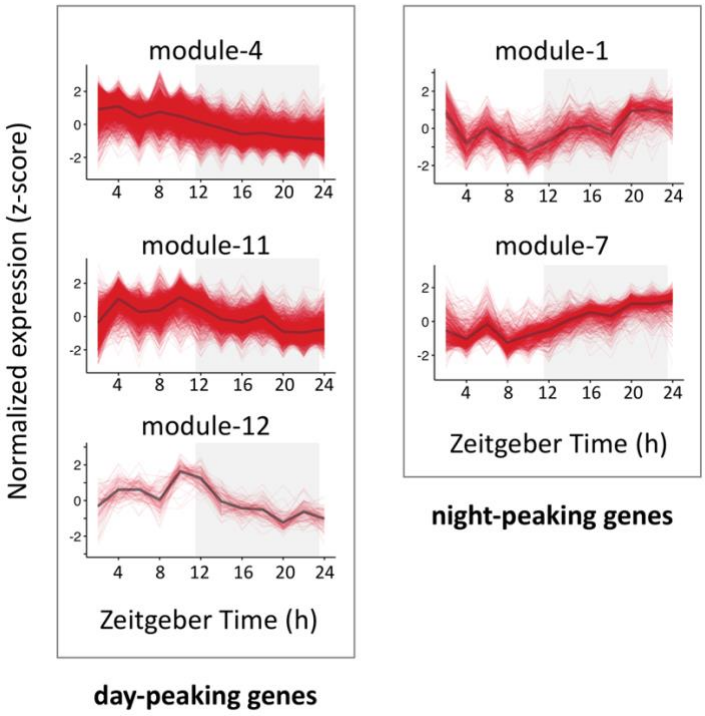

Figure 11. Daily expression patterns of genes in rhythmic modules: The daily expression pattern of all genes in a given module as well as the module's median gene expression are shown. For a module, each red line represents the expression of one gene, every 2h over a 24h day and the black line represents the module's median gene expression. The x-axis shows the time-of-day or Zeitgeber Time in hours, 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).

Using the same approach as above, we aimed to identify the ant modules that putatively underlie behavioral plasticity, as well as the modules that contained a significant number of genes that were affected during *Ophiocordyceps*-induced behavioral manipulation. As such, we determined the overlap between the various modules and "behavioral plasticity genes" (e.g., for-UP) – genes with differential expression in foragers as compared to nurses, as reported by Das and de Bekker (2022) (1) - and "behavioral manipulation genes" (e.g., ophio-UP) – genes differentially expressed during manipulation as reported in a previous transcriptomics study of *Ophiocordyceps*-infected *C. floridanus* (8) (Figure 12) (Supplementary File 2). The raw data of the behavioral manipulation study is deposited in NCBI under BioProject PRJNA600972.

```
## 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 manip
ulation
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")
```

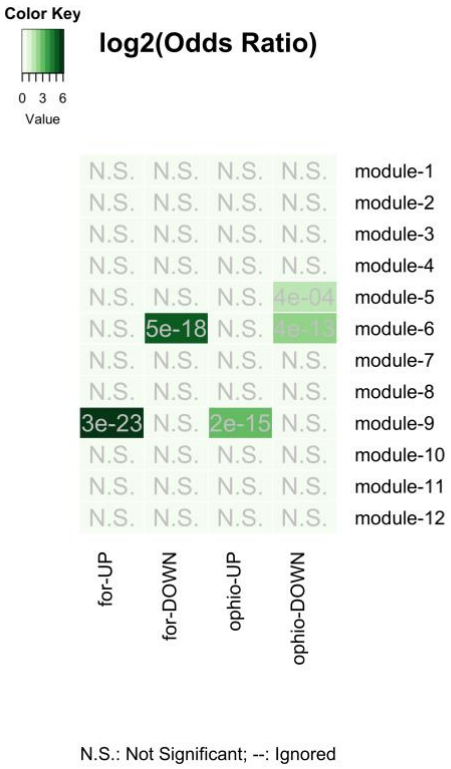

Figure 12. Gene-clusters with behavioral plasticity and parasitic behavioral manipulation genes: The heatmap identifies the different ant gene modules that are overrepresented in genes previously found to underlie behavioral plasticity (genes differentially expressed between foragers and nurses) (1) *and parasitic behavioral manipulation (genes differentially expressed in foragers during manipulated biting behavior)* (7).

We found that the modules that contain an overrepresentation of genes that putatively underlie ant behavioral plasticity (i.e., differential expression in foragers as compared to nurses) and the modules that contain genes significantly affected during *Ophiocordyceps*-induced behavioral manipulation, are the same (i.e., modules 6 and 9) (Figure 12). 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 (Figure 13). The modules that contain these genes appear to be closely connected to modules that contain an overrepresentation of rhythmic genes, suggesting that the expression of rhythmic genes and processes could also be (indirectly) affected by *Ophiocordyceps* infection.

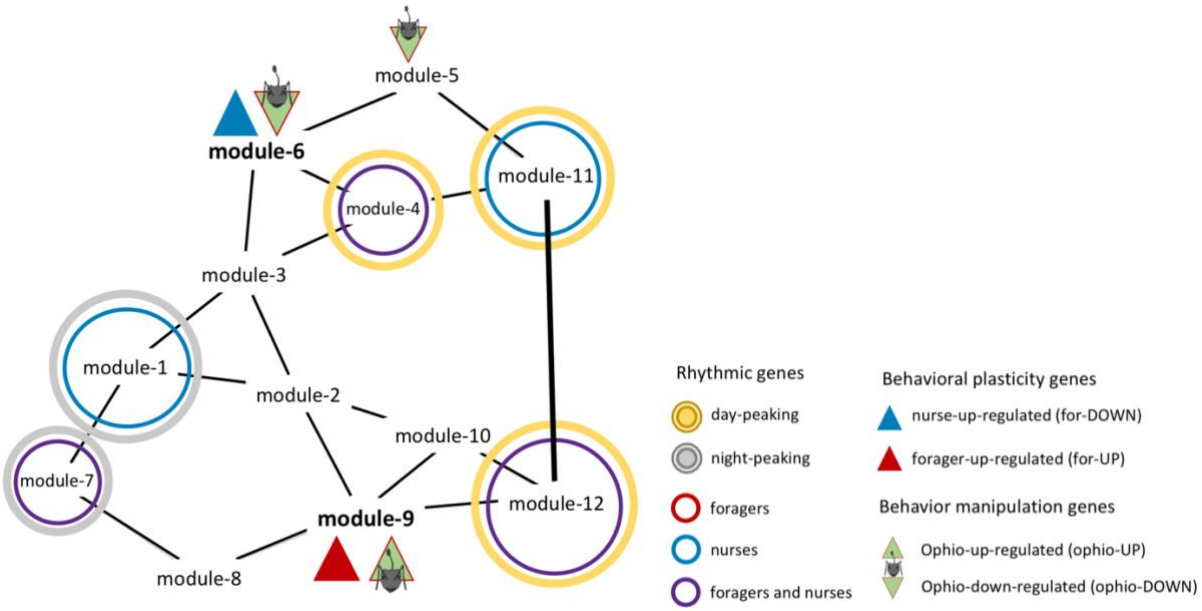

Figure 13. Annotated ant GCN: The annotated gene co-expression network summarizes the findings from our analyses and identifies different modules of interest that are putatively important for the interplay of rhythmicity, behavioral plasticity and parasitic behavioral manipulation.

References

1. Das B, de Bekker C. Time-course RNASeq of *Camponotus floridanus* forager and nurse ant brains indicate links between plasticity in the biological clock and behavioral division of labor. BMC Genomics 2022;23:57.
2. Shields EJ, Sheng L, Weiner AK, Garcia BA, Bonasio R. High-quality genome assemblies reveal long non-coding RNAs expressed in ant brains. Cell Reports 2018;23(10):3078–90.
3. Hutchison AL, Allada R, Dinner AR. Bootstrapping and empirical Bayes methods improve rhythm detection in sparsely sampled data. Journal of Biological Rhythms 2018;33(4):339–49.
4. Langfelder P, Horvath S. WGCNA: An R package for weighted correlation network analysis. BMC Bioinformatics. 2008;9(1):1–13.
5. Csárdi G, Nepusz T. The igraph software package for complex network research. R Package; 2007.
6. Hutchison AL, Allada R, Dinner AR. Bootstrapping and empirical bayes methods improve rhythm detection in sparsely sampled data. J Biol Rhythms. 2018; 33(4):339-49.
7. Shen L. GeneOverlap: An R package to test and vi-sualize gene overlaps. R package; 2018.
8. Will I, Das B, Trinh T, Brachmann A, Ohm RA, de Bekker C. Genetic underpinnings of host manipulation by *Ophiocordyceps* as revealed by comparative transcriptomics. G3. 2020;10(7):2275–96.