Enrichment Analysis

Jack Cazet

The code below is designed to identify transcription factors (TFs) that are likely to play an functional role in coordinating transcriptional cell states in *Hydra*. To do this, we test for motif enrichment in the regulatory sequences of coexpressed genes. We then identify TFs that (1) belong to the set of coexpressed genes and (2) can bind to one of the enriched motifs. TFs that meet these criteria are considered likely to play an important role in regulating that gene coexpression module.

To perform this analysis we need to know (1) which genes are coexpressed, (2) where putative regulatory sequences are in the genome, (3) which transcripts are likely to be TFs, and (4) what motifs those putative TFs are likely to bind. Information on (1) will be determined using non-negative factorization to identify coexpressed gene modules (metagenes). (2) will be drawn from a consensus peakset derived from whole-animal ATAC-seq data. (3) and (4) will use a combination of pfam anotations and the profile inference tool from the JASPAR RESTful API.

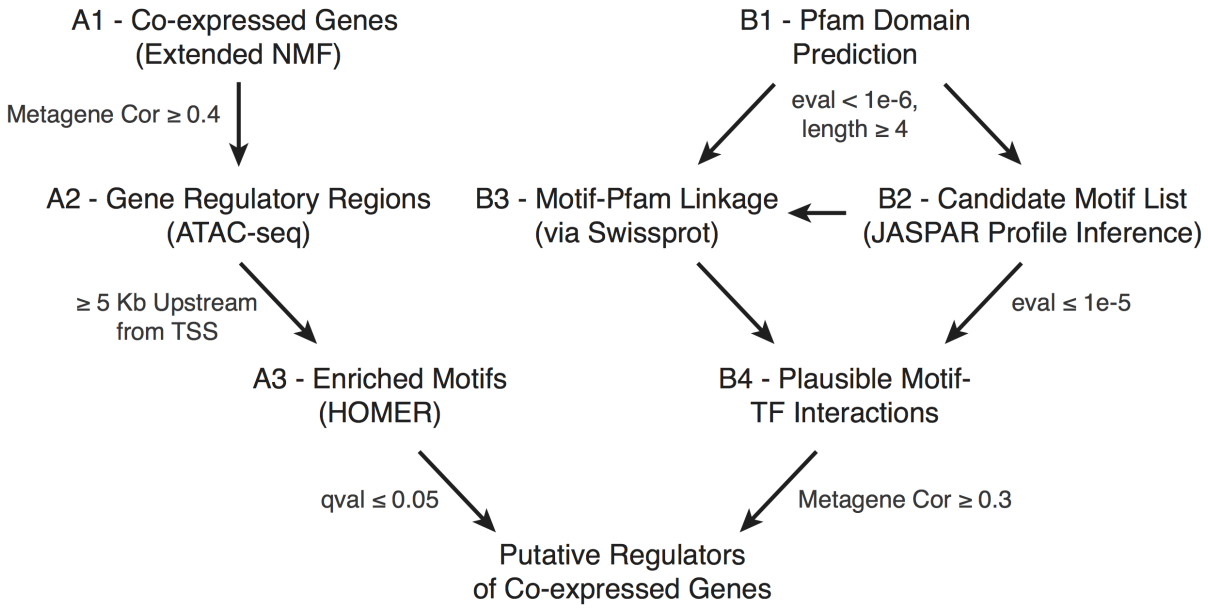

Figure 1: Workflow for the identification of master transcriptional regulators

Figure 1 is a visual summary of the analysis. Headers throughout this markdown will refer back to the corresponding steps in the workflow to orient the reader. For a more detailed written explanation of this figure, refer to supplemental figure S23.

Preparation for Analysis

In this chunk we establish our write path for file export and load several functions that we will need.

```
ds.ds <- readRDS("objects/genome_S1_pc30.rds")
```

```
writePath <- "./Enrichment_Results"
dir.create(writePath)
Sys.setenv(WRITEPATH = writePath)

hFind <- function(y, x) {
    return(y@data@Dimnames[[1]][grep(x, y@data@Dimnames[[1]],
        ignore.case = T)])
}

printGenetSNE <- function(y, x, w) {
    pdf(paste0(w, "/", x, ".pdf", sep = ""), width = 10, height = 10)
    FeaturePlot(y, x, cols.use = c("grey", "blue"), pt.size = 2)
    dev.off()
}
```

Identifying and Exporting Metagene-Associated Peaks (A1 & A2)

Subsetting the Consensus Peakset

First we need to isolate those genomic regions that are likely to regulate the transcription of a set of coexpressed genes (as determined by metagene association, see below). We will be using ATAC-seq peaks generated from three biological replicates of whole *Hydra*. These peaks pass an IDR threshold of 0.1 in at least one pairwise comparison between the three biological replicates (76476 total peaks).

To link peaks to genes, we used the UROPA package which uses a simple set of rules to perform peak annotation based on proximity to gene models. The output table used below was generated using the following settings:

“distance”: [“100000”], “direction”: [“any_direction”], “feature.anchor”: [“start”, “end”], “internals”: [“any”], “feature”: [“gene”], “attribute.value”: [“None”], “priority”: [“False”],“show.attributes”: [“ID”],“filter.attribute”: [“None”], “strand”: [“ignore”]

For this analysis we will limit the distance to within 5kb upstream of the gene model's start codon. This parameter can be easily changed to have greater or fewer peaks if desired.

```
# import the peak annotations from uropa
peaks <- read.table("enrichment_resources/Whole_2Rep_IDR_finalhits.txt",
    header = T, stringsAsFactors = F)

# import the bed file that we will ultimately be subsetting
bedpeaks <- read.table("enrichment_resources/2Rep.IDR.mod.bed",
    stringsAsFactors = F)

# focus on peaks that either overlap the TSS, or that are
# upstream of it (within 5kB)
peaks <- peaks[peaks$genomic_location %in% c("overlapStart",
    "upstream"), ]
peaks <- peaks[peaks$distance <= 5000, ]
```

Identifying Coexpressed Genes Using NMF Metagenes (A1)

To identify gene coexpression modules, we will pull metagene information from the NMF analysis conducted on the genome-mapped, whole animal dataset (wg_K84). The initial NMF was calculated on a set of ~1800

variable genes. Because we want to extend our analysis to genes not included in the initial variable gene list, we will determine additional metagene member genes based on how well they correlate with the NMF cell scores. Instead of considering the correlation across all cells, we limit the calculation to those cells that recieve a non-zero score for the metagene in question. This reduces the amount of computation required and is more favorable to genes that may also be expressed elsewhere (allows for non-exclusivity). We also exclude lowly expressed genes (must be expressed in at least 20 cells and have a mean expression (excluding zeros) above 0.5). For downstream analyses we consider a gene to be a member of a metagene if it's correlation score is >= 0.4.

```
# load DS dataset
ds.ds <- readRDS("objects/genome_S1_pc30.rds")

# pull cell scores from current NMF analysis (whole animal)
cellScores <- read.csv("nmf/wg_K84/GoodMeta_CellScores.csv",
    row.names = 1)

# remove scores of zero from cell scores so those cells are
# excluded from correlation analysis
cellScores[cellScores == 0] <- NA

# isolate expression data from seurat object (normalized,
# log-space)
expressData <- as.matrix(ds.ds@data)

# unload full DS dataset (reduces memory footprint)
rm(ds.ds)

# exclude lowly expressed genes
expressData <- expressData[which(rowSums(expressData) > 0), ]
expressData.mean <- apply(expressData, 1, function(x) mean(x[x !=
    0]))
expressData.count <- apply(expressData, 1, function(x) length(x[x >
    0]))
expressData <- expressData[which(expressData.mean > 0.5 & expressData.count >
    20), ]

# lineup rows for NMF and DS data to allow for correlation
# analysis
cellScores <- cellScores[match(rownames(cellScores), colnames(expressData)),
    ]

# function to pull cell scores for a particular metagene and
# check it's correlation with every gene passing expression
# thesholds in the DS dataset
metaCor <- function(x) {
    cellScores.use <- cellScores[, x]
    apply(expressData, 1, function(x) cor(x, cellScores.use,
        use = "pairwise.complete.obs"))
}

# Set up a parallel backend to perform a for loop that moves
# through all metagenes to check gene correlation
library(doParallel)
cl <- makeCluster(5)
```

```
registerDoParallel(cl)

corResults <- foreach(i = 1:length(colnames(cellScores))) %dopar%
    metaCor(i)

corResults.df <- as.data.frame(corResults)
colnames(corResults.df) <- colnames(cellScores)

# save the results for later, so you don't need to run this
# part of the analysis multiple times
save(corResults.df, file = paste0(writePath, "/", "corResults.Rdata"))
```

```
# convert NAs to 0
corResults.df[is.na(corResults.df)] <- 0

# Exclude all results with a correlation below threshold. The
# remaining entries are the member genes for each metagene
corResults.keep <- apply(corResults.df, 2, function(x) x[x >
    0.4])
```

Export Metagene-Associated Peaks (A2)

For our enrichment analysis, we will need a bed file containing the peaks associated with coexpressed genes for each metagene. This will be done based on the UROPA annotations described above. We will also output the list of genes that are considered members of each metagene as a text file of IDs.

```
# get rid of peaks for genes not considered in the analysis
peaks <- peaks[peaks$ID %in% gsub("[.]t.*", "", rownames(corResults.df)),
    ]

# pull just the gene names
metaMark.genes <- lapply(corResults.keep, function(x) names(x))

# return all peaks associated with the markers for each
# metagene cell population
metaPeaks <- lapply(metaMark.genes, function(x) peaks[peaks$ID %in%
    gsub("[.]t.*", "", x), ])

metaPeaks <- lapply(metaPeaks, function(x) bedpeaks[bedpeaks$V4 %in%
    x$peak_id, ])


# print bed files
invisible(lapply(1:length(metaPeaks), function(x) write.table(metaPeaks[[x]],
    file = paste0(writePath, "/", names(metaPeaks)[x], "_peaks.bed"),
    sep = "\t", row.names = F, col.names = F, quote = F)))

# also print gene lists
invisible(lapply(1:length(metaMark.genes), function(x) write.table(metaMark.genes[[x]],
    file = paste0(writePath, "/", names(metaMark.genes)[x], "_genes.txt"),
    sep = "\t", quote = F, row.names = F, col.names = F)))

# the code below will get the list of all peaks NOT
```

```
# associated with a particular metagene (one option for a
# control in the enrichment analysis; uncomment if control
# peaks are desired)

# controlIDs <- lapply(metaPeaks, function(x)
# peaks[!(peaks$peak_id %in% x$V4),]) controlPeaks <-
# lapply(controlIDs, function(x) bedpeaks[bedpeaks$V4 %in%
# x$peak_id,]) #print control bed files
# invisible(lapply(1:length(controlPeaks), function(x)
# write.table(controlPeaks[[x]],file =
# paste0(writePath,'/',names(controlPeaks)[x],'_Controlpeaks.bed'),
# sep = '\t', row.names = F, col.names = F, quote = F)))
```

If you want to verify the co-expression of the predicted metagene gene sets, execute the code below to generate tSNE plots for all metagene genes.

```
# get a list of all files files in the output directory

fileList <- list.files(writePath)

# subset to include only the gene list files
fileList <- fileList[grepl("genes.txt", fileList)]


# go through each gene list and make plots
for (i in 1:length(fileList)) {

    plotDir <- paste0(writePath, "/", "plots_", gsub(".txt",
        "", fileList[i]))

    dir.create(plotDir, showWarnings = T)

    geneTable <- read.table(paste0(writePath, "/", fileList[i]),
        stringsAsFactors = F)

    for (y in 1:length(geneTable[, 1])) {

        printGenetSNE(ds.ds, hFind(ds.ds, gsub("[|].*", "", geneTable[y,
            1])), plotDir)

    }
}
```

Identifying Putative Transcription Factors (B1)

Select Gene Models with Predicted DNA Binding Domains (B1)

In order to identify those transcripts that could plausibly regulate transcription, we will pull all genes that recieved a Pfam annotaion for a DNA binding domain, which will serve as our initial list of putative TFs. As a starting point we will use the list of DNA binding domains from Mendoza et al., 2013. We supplemented this list with POU, PAX, and COE domains.

```
# Load List of TF domains from Mendoza et al., 2013
pfam.TFdomains <- read.table("enrichment_resources/TF_domains.txt",
    stringsAsFactors = F)

# add POU, COE, and PAX domains
pfam.TFdomains <- c(pfam.TFdomains[, 1], "PF00157", "PF00292",
    "PF16422", "PF16423")

# load genome pfam annotations
pfam.hits <- read.csv("enrichment_resources/hydra.augustus.pfam.filtered.csv",
    stringsAsFactors = F, row.names = 1)

# remove extra columns
pfam.hits <- pfam.hits[, 1:4]

# eliminate stuff after decimal for transcriptome IDS
pfam.hits$V2 <- gsub("[.].*", "", pfam.hits$V2)

# find transcripts with TF pfam hit
pfam.hits.TFs <- pfam.hits[pfam.hits$V2 %in% pfam.TFdomains,
    ]

# get rid of duplicate hits
pfam.hits.TFs <- pfam.hits.TFs[!duplicated(pfam.hits.TFs$V4),
    ]

pfam.hits.TFs <- pfam.hits.TFs[c(1, 4)]
```

Identify Transcription Factors Correlated with Gene Coexpression Modules

At this stage we can start identifying putative regulators of co-expressed genes based on TF correlation alone. While these candidates may or may not be supported by motif enrichment, they may still be of interest. Run this chunk of code to generate plots for all TFs that have a correlation score at or above 0.3 for each metagene (we opted to slightly lower the correlation cutoff to catch more marginal TFs that might still be of interest).

```
# add columns for TF rank for each metagene
corResults.df$ID <- gsub("[|].*", "", rownames(corResults.df))

pfam.hits.TFs.scored <- merge(pfam.hits.TFs, corResults.df, by.x = "V4",
    by.y = "ID")

# get list of top twenty TFs for each metagene
topTfs <- apply(pfam.hits.TFs.scored[, 3:length(pfam.hits.TFs.scored)],
    2, function(x) {
        df <- data.frame(name = paste0(pfam.hits.TFs.scored[,
            1], "__", pfam.hits.TFs.scored[, 2]), score = x)
        df <- df[df$score > 0.3, ]
        df
    })

# eliminate empty DFs from list
topTfs[sapply(topTfs, function(x) nrow(x) == 0)] <- NULL
```

```
# mod ploting function for this particular application
printGenetSNE <- function(y, x, z) {
    pdf(paste0(z, "/", x, ".pdf", sep = ""), width = 10, height = 10)
    FeaturePlot(y, hFind(ds.ds, gsub("__.*", "", x)), cols.use = c("grey",
        "blue"), pt.size = 2)
    dev.off()
}

# reload seurat object
ds.ds <- readRDS("objects/genome_S1_pc30.rds")

# plot top TFs for each metagene
for (i in 1:length(topTfs)) {

    plotDir <- paste0(writePath, "/", "topTF_plots_", names(topTfs[i]))

    dir.create(plotDir, showWarnings = T)

    geneTable <- topTfs[[i]]

    for (y in 1:length(geneTable[, 1])) {

        printGenetSNE(ds.ds, geneTable[y, 1], plotDir)

    }
}

# export text files with gene IDs for top TFs for each
# metagene
invisible(lapply(1:length(topTfs), function(x) write.table(topTfs[[x]][,
    1], file = paste0(writePath, "/", names(topTfs)[x], "_TF_IDs.txt"),
    quote = F, sep = "\t", row.names = F, col.names = F)))
```

Identifying Potential Binding Motifs for *Hydra* TFs (B2 & B3)

In order to link regulatory sequence to potential regulators, we need to know the sequences bound by *Hydra* TFs. There are no *Hydra* TF binding data, so we need to rely on homology to assign binding motifs. JASPAR provides a means of doing this (through the RESTful API; **requires internet connection**) by identifying DNA binding regions in provided amino acid sequences and matching them to homologous domains for which the binding preference has been determined. It returns the motifs for the homologous domains. We can then search for these sequences in promoters to test for enrichment. If we do observe enrichment for a particular motif we can then link it to the TFs that were predicted to bind that motif.

Identify Binding Motifs Using JASPAR Profile Inference (B2)

```
# import dovetail v.1 protein sequences
dovetailProteins <- readAAStringSet("enrichment_resources/hydra.augustus.nameMod.fastp",
    format = "fasta")

# limit to proteins with predicted DNA binding domains
```

```
useThese <- which(names(dovetailProteins) %in% gsub("[.]t.*",
    "", pfam.hits.TFs$V4))

dovetailProteins <- dovetailProteins[useThese]

# remove isoforms
keepThese <- which(!duplicated(names(dovetailProteins)))

dovetailProteins <- dovetailProteins[keepThese]

# very very long protein sequences can cause the API to fail,
# so they must be excluded
notTooLong <- which(width(dovetailProteins) < 10000)

dovetailProteins <- dovetailProteins[notTooLong]

# remove genes that don't appear in dropseq
inDS <- which(names(dovetailProteins) %in% gsub("[.]t.*", "",
    ds.ds@data@Dimnames[[1]]))

dovetailProteins <- dovetailProteins[inDS]

# initialize empty results list
results <- vector("list", length(dovetailProteins))

# pull putative binding motifs using jaspar RESTful API and
# put results into list
for (i in 1:length(dovetailProteins)) {
    print(i)
    link <- paste0("http://jaspar.genereg.net/api/v1/infer/",
        as.character(dovetailProteins[i]), "/")
    result <- fromJSON(url(link))
    result <- result[["results"]]
    if (length(result != 0)) {
        results[[i]] <- result
    } else {
        print("Empty")
    }
}

# pull names for each result
names(results) <- names(dovetailProteins)

results.hits <- results

# remove empty entries
results.hits[sapply(results.hits, is.null)] <- NULL

# remove hits above evalue threshold
results.hits <- lapply(results.hits, function(x) x[x$evalue <
    1e-05, ])

# remove genes with no hits above evalue threshold
```

```
results.hits[sapply(results.hits, function(x) length(x$evalue) ==
    0)] <- NULL

# make gene lookup table
genes.motifs.df <- data.frame(ID = as.character(names(results.hits)),
    motifs = vapply(results.hits, function(x) paste(x$matrix_id,
        collapse = ","), ""))

# save the motif hits so we don't need to run this more than
# once
save(genes.motifs.df, file = paste0(writePath, "/", "genes.motifs.Rdata"))
```

Generate HOMER Motif Database Using JASPAR Profile Inference Tool

Once we have the motifs for each TF, we can generate the full list of motifs that we provide to HOMER. Then we will pull those motifs from the non-redundant CORE jaspar database (downloaded **here**) and put them in a folder to be converted into HOMER motif files.

```
# pull all (redundant) motif IDs from the motif lookup table
genes.motifs <- paste0(genes.motifs.df$motifs, collapse = ",")

genes.motifs <- strsplit(genes.motifs, ",")

genes.motifs <- genes.motifs[[1]]

# drop duplicated motifs
motif.IDs <- unique(genes.motifs)

# collapse and format in a way that can be used with grep
motif.IDs <- paste0(motif.IDs, collapse = "|")

rm(genes.motifs)

# copy the JASPAR motif files we need into the Hydra_PFMs
# folder
fileList <- list.files("./JASPAR2018_CORE_redundant_pfms_jaspar",
    full.names = T)

# find files with match to hydra hit
fileList <- fileList[grep(motif.IDs, fileList)]

# clear out previous results if they exist
unlink("./Hydra_PFMs", recursive = T)

dir.create("Hydra_PFMs")

# copy motifs into new folder
invisible(lapply(fileList, function(x) file.copy(from = x, to = "./Hydra_PFMs/")))
```

We can then take these JASPAR motif files and convert them to HOMER motif files using a combination of BASH and R. HOMER comes with a function to reformat JASPAR motifs (parseJasparMatrix.pl), but this does not add a detection threshold – instead just setting it to zero for all entries. The detection threshold determines the degree of degeneracy that is allowed for a sequence to be considered a TF binding site. Because

motifs will differ in length and degree of sequence preference, these scores need to be tailored to each motif. The rscript PWM_convert.R (which is called as part of the shell script jaspar2homer.sh) will normalize JASPAR motifs such that each row sums to 1 and will report the threshold for that motif (set to be some percentage below the maximum possible score). The shell script then combines the modified header and matrix into a new HOMER motif file.

Note: See how a binding score is calculated **here**.

```
cd enrichment_resources/Hydra_PFMs

../jaspar2homer.sh *.jaspar 2> /dev/null

cat *.motif > Hydra.motifs

rm *.final.motif
```

Link JASPAR Motif to Pfam IDs Using Swissprot (B3)

There are many instances in which TFs do not recieve a match through the JASPAR profile inference tool; however, in many cases these TFs will have an annotation for a DNA binding domain that has a highly conserved core binding motif (such as bZIPs or bHLHs). By identifying the Pfam domains associated with particular motifs (via JASPAR and Swissprot), we can infer matches to enriched motifs based on domain composition of putative TFs.

The annotations for each JASPAR motif includes the swissprot entry for the TF in question. The swissprot entry in turn has the Pfam domains associated with that TF. Swissprot and JASPAR entries are accessible via APIs, so we can move through a list of motifs and then identify their associated Pfam domains via swissprot/JASPAR.

```
# get list of JASPAR domains that we used for enrichment
# analysis
genes.motifs <- paste0(genes.motifs.df$motifs, collapse = ",")

genes.motifs <- strsplit(genes.motifs, ",")

genes.motifs <- genes.motifs[[1]]

motif.IDs <- unique(genes.motifs)

# initialize empty results list
results <- vector("list", length(motif.IDs))

# get uniprot IDs for each motif
for (i in 1:length(motif.IDs)) {
    print(i)
    link <- paste0("http://jaspar.genereg.net/api/v1/matrix/",
        as.character(motif.IDs[i]), "/")
    result <- fromJSON(url(link))
    result <- result[["uniprot_ids"]]
    if (length(result != 0)) {
        results[[i]] <- result
    } else {
        print("Empty")
    }
}
```

```r
# pull names for each result
names(results) <- motif.IDs

results <- unlist(results)

SPresults <- vector("list", length(results))

# Pull the pfam domains assocaited with each swissprot entry
for (i in 1:length(results)) {
    print(i)
    link <- paste0("https://www.uniprot.org/uniprot/", results[i],
        ".txt")
    result <- GET(link)
    result <- rawToChar(result$content)
    result <- strsplit(result, "\n")
    result <- result[[1]]
    result <- result[grepl("Pfam", result)]
    if (length(result != 0)) {
        result <- strsplit(result, ";")
        result <- lapply(result, function(x) x[2])
        result <- lapply(result, function(x) substring(x, 2))
        result <- unlist(result)
        SPresults[[i]] <- result
    } else {
        print("Empty")
    }
}

names(SPresults) <- results

# get rid of any pfam domains that aren't on our dna binding
# domain list
SPresults <- lapply(SPresults, function(x) x[x %in% pfam.TFdomains])

# get rid of empty results
SPresults[sapply(SPresults, function(x) length(x) == 0)] <- NA

SPresults <- lapply(SPresults, function(x) paste0(x, collapse = ","))

SPresults <- unlist(SPresults)

# generate table linking motif to pfam domain

pfam.lookup <- data.frame(JASPAR = names(results), Swissprot = results,
    PFAM = SPresults)
pfam.lookup <- pfam.lookup[pfam.lookup$PFAM != "NA", ]
pfam.lookup <- apply(pfam.lookup, 2, function(x) as.character(x))
pfam.lookup <- as.data.frame(pfam.lookup, stringsAsFactors = FALSE)


# for pax genes, the swissprot entries give annotation for
# both homeobox and pax pfam annotations this is redundant
# because the pax domain includes at least a partial
```

```
# homeodomain, so these entries can be collapsed into just
# the pax domain
pfam.lookup$PFAM[pfam.lookup$PFAM == "PF00046,PF00292"] <- "PF00292"

# a similar case exists for pou domains
pfam.lookup$PFAM[pfam.lookup$PFAM == "PF00046,PF00157"] <- "PF00157"

s <- strsplit(pfam.lookup$PFAM, split = ",")

# compile results into table
pfam.lookup <- data.frame(JASPAR = rep(pfam.lookup$JASPAR, sapply(s,
    length)), Swissprot = rep(pfam.lookup$Swissprot, sapply(s,
    length)), PFAM = unlist(s))

pfam.lookup$PFAM <- as.factor(pfam.lookup$PFAM)

# collapse table such that there is only one row per pfam
# domain
pfam.lookup <- pfam.lookup %>% group_by(PFAM) %>% summarise(motifs = paste(JASPAR,
    collapse = ","))

pfam.lookup <- as.data.frame(pfam.lookup, stringsAsFactors = FALSE)

# for each pfam domain listed, we want to list all hydra
# proteins that were annotated with that domain
hydra_pfam_match <- lapply(pfam.lookup$PFAM, function(x) gsub("[.]t.*",
    "", pfam.hits[pfam.hits$V2 %in% x, 4]))

hydra_pfam_match <- lapply(hydra_pfam_match, unique)

names(hydra_pfam_match) <- pfam.lookup$PFAM

hydra_pfam_match <- lapply(hydra_pfam_match, function(x) paste0(x,
    collapse = ","))

pfam.lookup$hydra_match <- unlist(hydra_pfam_match)
```

HOMER Motif Enrichment Analysis (A3)

Next we want to execute a shell script to perform a HOMER enrichment analysis for each of the bed files we generated above (one for each metagene). The parameters for the HOMER analysis are specified in the shell script, we just specify the bed files to be used.

Note: if you want to use a pre-specified set of control sequences instead of the HOMER default (generate GC% matched control sequences from random places in the genome), you will need to modify the shell script.

```
#This should be the path to your HOMER install
export PATH=$PATH:$HOME/homer/bin/

cd $WRITEPATH

../enrichment_resources/findMotifs_homer.sh *_peaks.bed
#2> /dev/null
```

We can then import the plain text results from HOMER and identify the significantly enriched (qval <= 0.05) motifs for each metagene.

```
# list all files in motif enrichment directory
fileIDs <- list.files(writePath, recursive = T, full.names = T)

# get only the text files with homer results
fileIDs <- fileIDs[grep("knownResults.txt", fileIDs)]

# generate list containing all enrichment results
enrich.results <- lapply(fileIDs, function(x) read.table(x, header = F,
    stringsAsFactors = F, row.names = NULL, skip = 1, sep = "\t"))

# give correct column names
correctCol <- c("Name", "Consensus", "pval", "log-pval", "qval",
    "targetHits", "targetPercent", "backgroundHits", "backgroundPercent")

enrich.results <- lapply(enrich.results, setNames, correctCol)

# assign names for each list entry based on it's metagene
names(enrich.results) <- gsub("_peaks.*", "", fileIDs)

# keep only enrichment results with qval below 0.05
enrich.results <- lapply(enrich.results, function(x) x[x$qval <=
    0.05, ])

# remove empty slots
enrich.results[sapply(enrich.results, function(x) nrow(x) ==
    0)] <- NULL

# save results
save(enrich.results, file = paste0(writePath, "/", "enrich.results.Rdata"))

# pull just the IDs of enriched motifs
motifs.results <- lapply(enrich.results, function(x) strsplit(x$Name,
    split = "/", fixed = T))

motifs.results <- lapply(motifs.results, function(x) vapply(x,
    function(y) y[2], ""))

motifs.results <- lapply(motifs.results, function(x) as.character(x))

names(motifs.results) <- gsub(paste0(writePath, "/"), "", names(motifs.results))
```

Determine Plausible TF-Motif Interactions and Identify Potential Regulators (B4)

Next, we want to match the enriched motifs to TFs that are expressed in a particular metagene. To do this we will go through each metagene and check if there is a TF predicted to bind an enriched motif that also has a correlation score above 0.3. We compile a list of all positive matches that summarizes the enrichment results.

```
corResults.df$ID <- NULL
```

```
genes.motifs.df$ID <- as.character(genes.motifs.df$ID)
genes.motifs.df$motifs <- as.character(genes.motifs.df$motifs)

# loosen cutoff to catch marginal TFs
corResults.keep <- apply(corResults.df, 2, function(x) x[x >
    0.3])


# identify those TFs that likely bind to a particular motif
# and that are correlated with the metagene of interest

# initialize empty results object
TF.match <- vector("list", length = length(motifs.results))

# move through the list of results for all the metagenes with
# significant hits
for (i in 1:length(motifs.results)) {

    # identify the list of TFs associated with the motifs (via
    # JASPAR) found for the metagene in question
    geneList <- lapply(motifs.results[[i]], function(x) genes.motifs.df[grep(x,
        genes.motifs.df$motifs), "ID"])

    # do the same but using the PFAM results
    pfamList <- lapply(motifs.results[[i]], function(x) {
        if (length(pfam.lookup[grep(x, pfam.lookup$motifs), "hydra_match"]) !=
            0) {
            strsplit(pfam.lookup[grep(x, pfam.lookup$motifs),
                "hydra_match"], ",")[[1]]
        } else {
            return("")
        }
    })

    # keep only those TFs that are correlated with the metagene
    # in question
    geneList <- lapply(geneList, function(x) x[x %in% gsub("[.]t.*",
        "", names(corResults.keep[[names(motifs.results)[i]]]))])

    pfamList <- lapply(pfamList, function(x) x[x %in% gsub("[.]t.*",
        "", names(corResults.keep[[names(motifs.results)[i]]]))])

    # pull the correlation scores for each TF that was included
    scores <- lapply(geneList, function(x) corResults.df[match(x,
        gsub("[.]t.*", "", rownames(corResults.df))), names(motifs.results)[i]])

    PfamScores <- lapply(pfamList, function(x) corResults.df[match(x,
        gsub("[.]t.*", "", rownames(corResults.df))), names(motifs.results)[i]])

    # collapse the list of TFs into a single character string
    geneList <- unlist(lapply(geneList, function(x) paste(x,
        collapse = ",")))
```

```
    # do the same for the scores
    scores <- unlist(lapply(scores, function(x) paste(x, collapse = ",")))

    # do the same for the pfam results
    pfamList <- unlist(lapply(pfamList, function(x) paste(x,
        collapse = ",")))

    PfamScores <- unlist(lapply(PfamScores, function(x) paste(x,
        collapse = ",")))

    # combine everything into a data frame with the motif, it's
    # associated TFs, and the metagene correlation score
    TF.match[[i]] <- data.frame(motif_id = motifs.results[[i]],
        TF = geneList, CorScore = scores, pfamTF = pfamList,
        pfamCorScore = PfamScores)
}

# fix the names to keep metagene ID
names(TF.match) <- names(motifs.results)

# remove empty entries
for (i in 1:length(TF.match)) {
    DF <- TF.match[[i]]
    DF <- DF[(DF$TF != "") | (DF$pfamTF != ""), ]
    TF.match[[i]] <- DF
}

TF.match[sapply(TF.match, function(x) nrow(x) == 0)] <- NULL

# collapse list to DF
TF.match <- ldply(TF.match)

# export table
write.csv(TF.match, file = paste0(writePath, "/TF.match.csv"))
```

Another way to visualize the enrichment results is to focus on the transcription factors predicted to be key metagene regulators. Below, we move through a list of all TFs that were predicted to regulate at least one metagene and generate tSNE plots both for the TF in question as well as the cell scores for all metagenes predicted to be regulated by that TF.

```
# pull cell scores from current NMF analysis (whole animal)
cellScores <- read.csv("wg_K84/GoodMeta_CellScores.csv", row.names = 1)

# add metagene info to the metadata slot of the seurat object
cellScores <- cellScores[match(rownames(cellScores), rownames(ds.ds@meta.data)),
    ]
cellScores[is.na(cellScores)] <- 0
ds.ds@meta.data <- cbind(ds.ds@meta.data, cellScores)

# create directory called keyTFs, and then for each TF, plot
# the gene itself and all metagenes for which it is a
# putative regulator in a subdirectory

dir.create(paste0(writePath, "/", "keyTFs"))
```

```
# get list of key TFs
keyTFs <- c(as.character(TF.match$TF), as.character(TF.match$pfamTF))
keyTFs <- unlist(strsplit(keyTFs, split = ",", fixed = T))
keyTFs <- unique(keyTFs)

# move through list of TFs, make a subdirectory for each TF
# and plot it's expression along with associated metagenes
for (i in 1:length(keyTFs)) {
    subPath <- paste0(writePath, "/", "keyTFs/", keyTFs[i])
    dir.create(subPath)

    # plot it's expression pattern
    pdf(paste0(subPath, "/", keyTFs[i], ".pdf"), width = 10,
        height = 10)
    print(FeaturePlot(ds.ds, hFind(ds.ds, paste0(keyTFs[i], ".t")),
        cols.use = c("grey", "blue"), pt.size = 2))
    dev.off()

    # plot all metagenes associated with TF
    linkedMeta <- TF.match[grep(keyTFs[i], paste(TF.match$pfamTF,
        TF.match$TF, sep = ",")), ".id"]

    for (y in 1:length(linkedMeta)) {
        pdf(paste0(subPath, "/", linkedMeta[y], ".pdf"), width = 10,
            height = 10)
        print(FeaturePlot(ds.ds, linkedMeta[y], cols.use = c("grey",
            "blue"), pt.size = 2))
        dev.off()
    }
}
```

As yet another way of interrogating the results, we can move through the list of all motifs found to be significantly enriched in at least one metagene and generate a plot of all the metagenes associated with that motif along with the TFs predicted to regulate the motif in those metagenes.

```
# create directory called byMotif, and then for each enriched
# Motif, plot all metagenes for which there is enrichment

dir.create(paste0(writePath, "/", "byMotif"))

# get list of key TFs
motifList <- as.character(unique(unlist(motifs.results)))

# initialize empty results list
results <- vector("character", length(motifList))

# pull additional information on the motif from JASPAR (short
# name and TF family). This helps make the output easier to
# interpret
for (i in 1:length(motifList)) {
    print(i)
    link <- paste0("http://jaspar.genereg.net/api/v1/matrix/",
        as.character(motifList[i]), "/")
    result <- fromJSON(url(link))
```

```
    result <- paste(result[["family"]], result[["name"]], sep = "_")
    results[i] <- result

}

results <- paste(results, motifList, sep = "_")

# move through list of motifs, make a subdirectory for each
# motif and the associated metagenes along with TFs predicted
# to regulate those metagenes
for (i in 1:length(motifList)) {
    subPath <- paste0(writePath, "/", "byMotif/", results[i])
    dir.create(subPath)


    # plot all metagenes associated with the motif
    linkedMeta <- TF.match[TF.match$motif_id == motifList[i],
        ".id"]

    for (y in 1:length(linkedMeta)) {
        pdf(paste0(subPath, "/", linkedMeta[y], ".pdf"), width = 10,
            height = 10)
        print(FeaturePlot(ds.ds, linkedMeta[y], cols.use = c("grey",
            "blue"), pt.size = 2))
        dev.off()
    }
    # create a subdirectory for each metagene and plot the TFs
    # that are predicted to bind that motif in those cells
    DF <- TF.match[TF.match$motif_id == motifList[i], ]

    for (y in 1:nrow(DF)) {
        subsubPath <- paste0(subPath, "/", DF$.id[y], "/")
        dir.create(subsubPath)

        metaTFs <- paste(DF$TF[y], DF$pfamTF[y], sep = ",")
        metaTFs <- strsplit(metaTFs, ",")[[1]]
        metaTFs <- metaTFs[metaTFs != ""]

        for (z in 1:length(metaTFs)) {
            pdf(paste0(subsubPath, "/", metaTFs[z], ".pdf"),
                width = 10, height = 10)
            print(FeaturePlot(ds.ds, hFind(ds.ds, paste0(metaTFs[z],
                ".t")), cols.use = c("grey", "blue"), pt.size = 2))
            dev.off()
        }
    }
}
```

Finally, we can survey all motif enrichment results for all metagenes by generating a binary enrichment heatmap, to visualize broader trends in transcriptional regulation. This figure corresponds to supplementary figure S22.

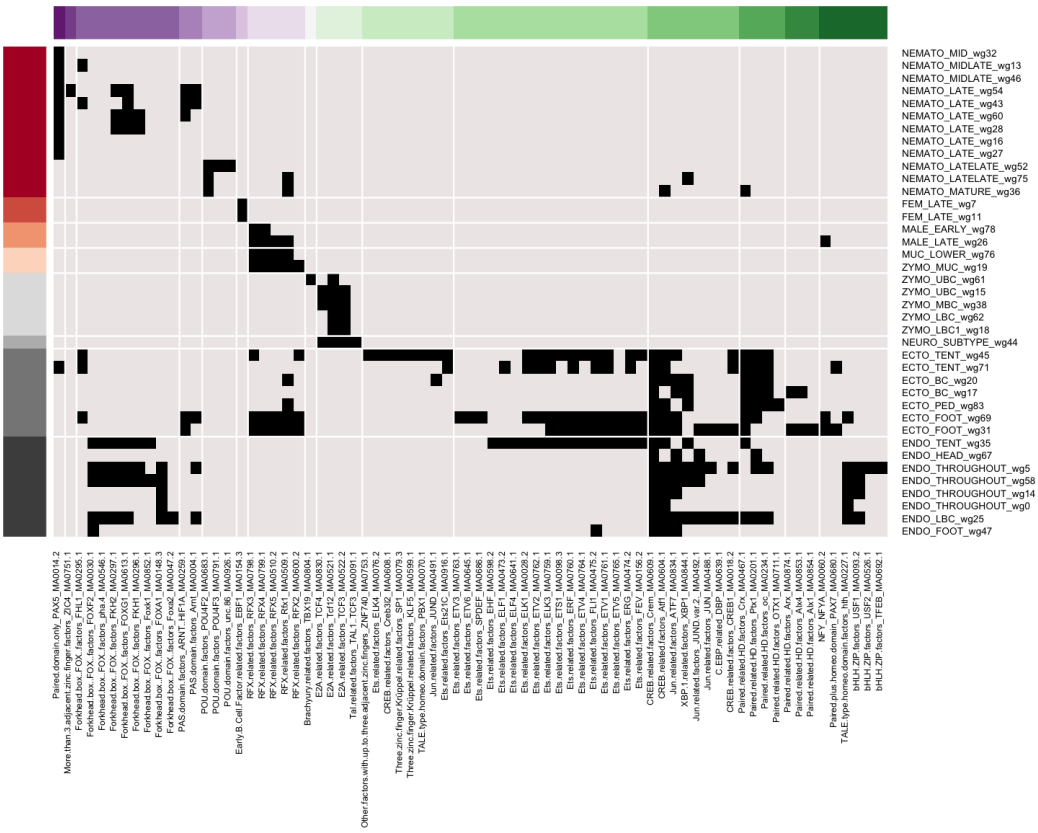

Figure 2: Motif Enrichment Matrix for wg K=84 Metagenes

Software versions

This document was computed on Tue Oct 16 14:20:31 2018 with the following R package versions.

```
R version 3.5.1 (2018-07-02)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS High Sierra 10.13.4

Matrix products: default
BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] RColorBrewer_1.1-2  gplots_3.0.1        dplyr_0.7.6
 [4] plyr_1.8.4          httr_1.3.1          Biostrings_2.48.0
 [7] XVector_0.20.0      IRanges_2.14.1      S4Vectors_0.18.1
```

```
[10] BiocGenerics_0.26.0 jsonlite_1.5        kableExtra_0.9.0
[13] xtable_1.8-2        knitr_1.20          magick_1.9
[16] Seurat_2.3.4        Matrix_1.2-14       cowplot_0.9.3
[19] ggplot2_3.0.0

loaded via a namespace (and not attached):
  [1] Rtsne_0.13          colorspace_1.3-2    class_7.3-14
  [4] modeltools_0.2-22   ggridges_0.5.0      mclust_5.4.1
  [7] rprojroot_1.3-2     htmlTable_1.12      base64enc_0.1-3
 [10] rstudioapi_0.7      proxy_0.4-22        flexmix_2.3-14
 [13] bit64_0.9-7         mvtnorm_1.0-8       xml2_1.2.0
 [16] codetools_0.2-15    splines_3.5.1       R.methodsS3_1.7.1
 [19] robustbase_0.93-2   Formula_1.2-3       ica_1.0-2
 [22] cluster_2.0.7-1     kernlab_0.9-26      png_0.1-7
 [25] R.oo_1.22.0         readr_1.1.1         compiler_3.5.1
 [28] backports_1.1.2     assertthat_0.2.0    lazyeval_0.2.1
 [31] formatR_1.5         lars_1.2            acepack_1.4.1
 [34] htmltools_0.3.6     tools_3.5.1         bindrcpp_0.2.2
 [37] igraph_1.2.2        gtable_0.2.0        glue_1.3.0
 [40] RANN_2.6            reshape2_1.4.3      Rcpp_0.12.18
 [43] trimcluster_0.1-2.1 gdata_2.18.0        ape_5.1
 [46] nlme_3.1-137        iterators_1.0.10    fpc_2.1-11.1
 [49] gbRd_0.4-11         lmtest_0.9-36       stringr_1.3.1
 [52] rvest_0.3.2         irlba_2.3.2         gtools_3.8.1
 [55] DEoptimR_1.0-8      zlibbioc_1.26.0     MASS_7.3-50
 [58] zoo_1.8-3           scales_0.5.0        hms_0.4.2
 [61] doSNOW_1.0.16       yaml_2.2.0          reticulate_1.9
 [64] pbapply_1.3-4       gridExtra_2.3       rpart_4.1-13
 [67] segmented_0.5-3.0   latticeExtra_0.6-28 stringi_1.2.4
 [70] foreach_1.4.4       checkmate_1.8.5     caTools_1.17.1.1
 [73] bibtex_0.4.2        Rdpack_0.9-0        SDMTools_1.1-221
 [76] rlang_0.2.1         pkgconfig_2.0.1     dtw_1.20-1
 [79] prabclus_2.2-6      bitops_1.0-6        evaluate_0.11
 [82] lattice_0.20-35     ROCR_1.0-7          purrr_0.2.5
 [85] bindr_0.1.1         htmlwidgets_1.2     bit_1.1-14
 [88] tidyselect_0.2.4    magrittr_1.5        R6_2.2.2
 [91] snow_0.4-2          Hmisc_4.1-1         pillar_1.3.0
 [94] foreign_0.8-70      withr_2.1.2         fitdistrplus_1.0-9
 [97] mixtools_1.1.0      survival_2.42-3     nnet_7.3-12
[100] tibble_1.4.2        tsne_0.1-3          crayon_1.3.4
[103] hdf5r_1.0.0         KernSmooth_2.23-15  rmarkdown_1.10
[106] grid_3.5.1          data.table_1.11.4   metap_1.0
[109] digest_0.6.15       diptest_0.75-7      tidyr_0.8.1
[112] R.utils_2.6.0       munsell_0.5.0       viridisLite_0.3.0
```