

T. brucei co-expression network analysis

Kennedy Mwangi

June 27, 2019

Introduction

This document contains the workflow used in the analysis of *T. brucei* gene co-expression network analysis. It contains code used in each step of the analysis.

Setting up R for the analysis

```
# set working directory
setwd("analysis/")

# ensure results are reproducible
set.seed(1)

# other settings
options(digits = 4)
options(stringsAsFactors = FALSE)

# loading required R packages
library("dupRadar")
library("Rsubread")
library("limma")
library("edgeR")
library("sva")
library("RColorBrewer")
library("ggplot2")
library("gplots")
library("reshape2")
library("ggfortify")
library("xlsx")
library("WGCNA")
library("flashClust")
library("tidyverse")
library("igraph")
library('foreach')
library('doParallel')
library('goseq')
library('Trypanosoma.brucei.TREU927', character.only = TRUE)
library('org.Tb927.tritryp.db')
library('TxDb.TbruceiTREU927.tritryp43.genes')
library('dplyr')
library('rtracklayer')
library('tibble')

# load helper functions
source("~/manuscript-shared-rnaseq/R/enrichment_analysis.R")
```

```

source("~/manuscript-shared-rnaseq/R/annotations.R")
source("~/manuscript-shared-rnaseq/R/wgcna.R")
source("~/manuscript-shared-rnaseq/R/util.R")

# set number of threads to allow in WGCNA analysis
allowWGCNAThreads(nThreads=20)

```

Data acquisition

Data used in this study is obtained from European Nucleotide Archive under accession number SRP002243 and SRR965341.

First, metadata for the data is obtained from EBI as follows:

```

#Obtain metadata information for the data used in this study from ENA and SRA databases.

# ENA metadata
# code adapted from: https://wiki.bits.vib.be/index.php/Download_read_information_and_FASTQ_data_from_the_EBI_Web_Services

accession <- "SRP002243" # similarly, obtain data for SRR965341
ena.url <- paste("http://www.ebi.ac.uk/ena/data/warehouse/filereport?accession=",
                  accession,
                  "&result=read_run",
                  "&fields=run_accession,library_name,",
                  "read_count,fastq_ftp,fastq_aspera,",
                  "fastq_galaxy,sra_ftp,sra_aspera,sra_galaxy,",
                  "&download=text",
                  sep="")
ENA.metadata <- read.table(url(ena.url), header=TRUE, sep="\t")

# SRA metadata
SRA.metadata <- read.table("../data/SraRunTable.metadata.txt", header = TRUE, sep = "\t")

# create a text file with urls to fastq files in ENA database
fastq.urls <- ENA.metadata[grep1("fastq_ftp", names(ENA.metadata))]
write.csv(fastq.urls, file="../data/fastq.urls.txt", eol = "\r\n", quote = FALSE, row.names = FALSE)

# obtain sample metadata to be used later in analysis in R.
matches <- c("Run","Library_Name","Sample_Name")
sample.metadata <- SRA.metadata[grep1(paste(matches, collapse="|"), names(SRA.metadata))]

# create grouping factor that will place each sample in the one of three tissues i.e.
# midgut (MG), proventriculus (PV) and salivary glands (SG)
tissue <- factor(c("MG", "MG", "MG", "MG", "MG", "PV", "PV", "SG", "SG", "SG", "SG",
                  "MG", "MG", "PV", "SG", "SG", "PV"))

# append factor to sample.metadata to group samples
sample.metadata["Tissue"] <- tissue

# The sample below was analysed separately as the reads are paired-end while
# the other samples are single-end.
#
# Add sample from Telleria et al 2014 study (SRR965341) to sample metadata.

```

```

sample.metadata$Run <- as.character(sample.metadata$Run)

sample.metadata <- rbind(sample.metadata, "18" = c("SA1", "SRR965341", "SA1", "SG"))

#####
# Exclusion of the following samples was done after analysis showed they
# were below required standards

# remove a sample with less than 10M reads from the analysis (SRR039951)
sample.metadata <- sample.metadata[-15,]

# remove 2 samples that have technical duplicates (SRR039937 and SRR039938)
sample.metadata <- sample.metadata[-9,]
sample.metadata <- sample.metadata[-8,]
#####

# include batch information for the samples
batch <- factor(c(1, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2))

sample.metadata["Batch"] <- batch

# print out the sample metadata table
sample.metadata

```

	Library_Name	Run	Sample_Name	Tissue	Batch
## 1	mg1	SRR039378	MG1	MG	1
## 2	mg1	SRR039381	MG1	MG	1
## 3	mg1	SRR039453	MG1	MG	1
## 4	MG2_SL	SRR039454	MG2	MG	2
## 5	MG2_SL	SRR039455	MG2	MG	2
## 6	PV2_SL	SRR039456	PV2	PV	2
## 7	PV2_SL	SRR039457	PV2	PV	2
## 10	SA2_SL	SRR039939	SA2	SG	2
## 11	SA2_SL	SRR039940	SA2	SG	2
## 12	mg1	SRR039948	MG1	MG	1
## 13	MG2_SL	SRR039949	MG2	MG	2
## 14	PV2_SL	SRR039950	PV2	PV	2
## 16	SA2_SL	SRR039952	SA2	SG	2
## 17	PV2_SL	SRR042429	PV2	PV	2
## 18	SA1	SRR965341	SA1	SG	2

Next, RNASeq data is downloaded from EBI database's FTP site.

```
cat ./scripts/fastq_download.sh
```

```

## #!/bin/bash
## #
## #Script to download fastq files from European Nucleotide Archive
## #
## FILE=$1 #File containing fastq url links to EBI FTP site
##
## OUT_DIR=../data/raw_data/
##
## cat ${FILE} | xargs -n1 wget$2 -P ${OUT_DIR}

```

Some of the downstream tools require that FASTQ files that were downloaded in zipped form are unzipped.

```
cat ../scripts/unzip.sh

## #!/bin/bash
## #
## #Script to decompress fastq.gz files
## #
## FASTQ_FILES=../data/raw_data/*.fastq.gz
##
## for file in ${FASTQ_FILES}; do
##     gunzip ${file}
## done
```

Data quality assessment

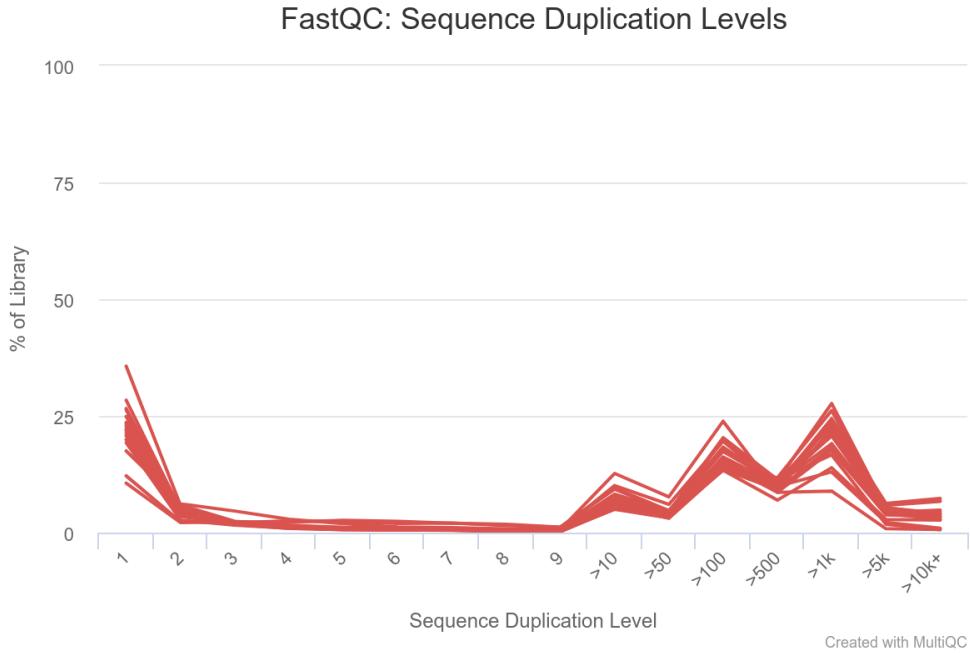
After downloading the RNASeq data, its quality is checked through the FASTQC tool whose output is a report in HTML format.

```
cat ../scripts/fastqc_reports.sh
```

```
## #!/bin/bash
## #
## #Script to run FastQC reports using the FastQC tool
## #
## #load fastqc module
## module load fastqc/0.11.4
##
## FASTQ_DIR=../data/raw_data/*.fastq
##
## #create output directory if it doesn't exist.
## mkdir -p ../results/fastqc_reports
##
## REPORTS_DIR=../results/fastqc_reports/
##
## for file in ${FASTQ_DIR}; do
##     fastqc -f fastq -o ${REPORTS_DIR} ${file}
## done
```

Following the high rate of duplicate reads after FASTQC analysis, further analysis is done to ascertain their cause. Duplicate reads are assessed whether they arise from artifacts in PCR (PCR duplicates) or from biological causes (highly expressed genes). This is done later in the analysis after read mapping. Below is the sequence duplication level plot.

See more on duplication here.



Downloading *T. brucei* and *G. morsitans* genome and annotation files

Genomes are obtained from their respective databases before alignment. The steps to obtain *Glossina morsitans* genome and its annotation file is indicated here as part of documenting the process. It is used in the analysis for reasons stated further in the reads mapping section. The genome and annotation files are downloaded from the TriTrypDB and vectorbase databases as follows:

```
# Downloading T. brucei genome
```

```
wget https://tritrypdb.org/common/downloads/release-43/TbruceiTREU927.fasta/data/TriTrypDB-43_TbruceiTREU927.fasta
-P ../data/tbrucei_genome/
```

```
# Downloading the GFF file
```

```
wget https://tritrypdb.org/common/downloads/release-43/TbruceiTREU927/gff/data/TriTrypDB-43_TbruceiTREU927.gff
-P ../data/tbrucei_genome_annotations_GFF/
```

```
# convert the tbrucei gene annotation from GFF format to GTF (required by some downstream tools)
# uses gffread from cufflinks
```

```
mkdir -p ../data/tbrucei_genome_annotations_GTF
```

```
gffread ../data/tbrucei_genome_annotations_GFF/TriTrypDB-43_TbruceiTREU927.gff \
-T -o ../data/tbrucei_genome_annotations_GTF/TriTrypDB-43_TbruceiTREU927.gtf
```

```
# Download T. brucei annotated transcripts (for use in UTR motif discovery)
```

```
wget https://tritrypdb.org/common/downloads/release-43/TbruceiTREU927.fasta/data/TriTrypDB-43_TbruceiTREU927.fasta
-P ../data/tbrucei_annotated_transcripts/
```

```
# Downloading Glossina genome
```

```
wget https://www.vectorbase.org/download/glossina-morsitans-yalescaffolds_sgmr1fagz \
-P ../data/glossina_genome_scaffolds/
```

```
# Downloading GTF file
```

```
wget https://www.vectorbase.org/download/glossina-morsitans-yalebasefeaturesgmory19gtfgz \
-P ../data/glossina_genome_annotation_GTF/
```

Alignment of reads on the genome (Read Mapping)

The first step is indexing the genome using HISAT2 followed by alignment of the reads. The output is SAM files.

Indexing the genome

```
cat ../scripts/hisat2_index.sh

## #!/bin/bash
## #
## #Script to index genome using HISAT2
## #
## GENOME_FILE=$1
##
## hisat2-build ${GENOME_FILE} bru-mor_genome_index_hisat2
```

Aligning the reads to the genome

```
cat ../scripts/hisat2_align.sh

## #!/bin/bash
## #
## #Script to align reads to the indexed genome using HISAT2
## #
## for fastq in ../data/raw_data/*.fastq; do
##     fqname=$(echo $fastq | cut -f1 -d '.')
##
##     hisat2 \
##         -x bru-mor_genome_index_hisat2 \
##         -U ${fastq} \
##         -S ${fqname}.sam \
##         -p 8 \
##         --summary-file ${fqname}.txt \
##         --new-summary
## done
##
## #move the output sam files to a new directory
##
## #create directory if not exists
## mkdir -p ../data/processed_data/bru-mor_sam
##
## #move the sam files
## mv ../data/raw_data/*.sam ../data/processed_data/bru-mor_sam/
```

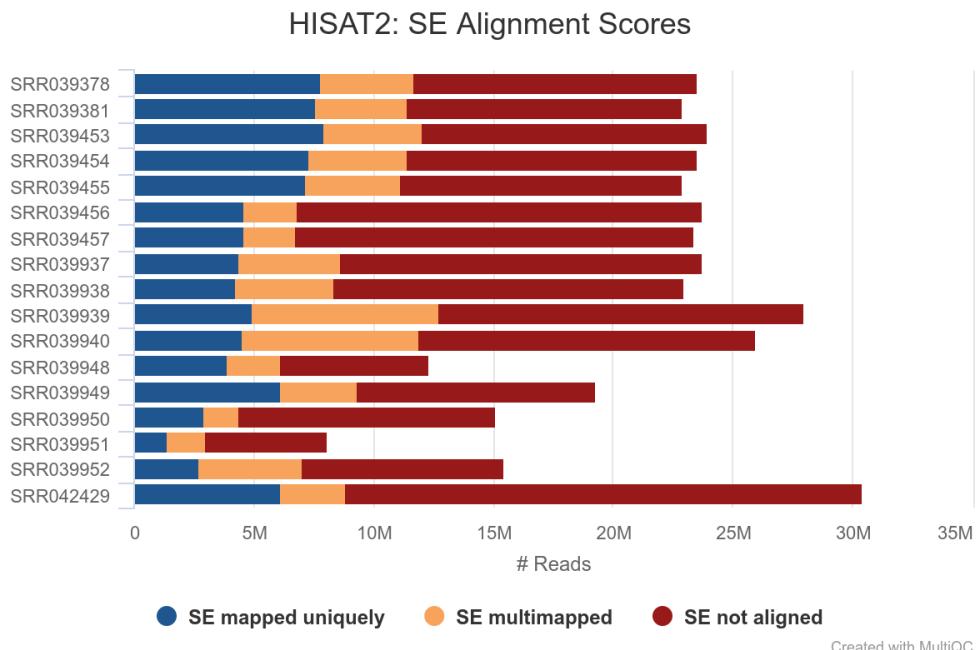
Following the high rate of unaligned reads, further analysis is done to ascertain their cause. The reads are aligned to *Glossina morsitans* genome to determine whether reads from the vector were also present in the sample during sequencing (Dual RNA-Seq). *T. brucei* and *G. morsitans* genome files are concatenated into a

single fasta file which is used during the alignment of the reads. This also ensures no cross-mapping of reads take place.

```
# make a directory to store the concatenated genomes
mkdir -p ../data/brucei-morsitans

# copy the genome files to the created directory and concatenate them
cp ../data/glossina_genome_scaffolds/glossina-* ../data/brucei-morsitans/
cp ../data/tbrucei_genome/*.fasta ../data/brucei-morsitans/
cat ../data/brucei-morsitans/*.fa* > ../data/brucei-morsitans/brucei-morsitans_genomes.fasta
```

HISAT2 is used to re-align reads on the *T. brucei* and *G. morsitans* genomes, beginning with indexing the genomes. Below is the alignment results from HISAT2 alignment tool.



Assessment of the duplication rate

At this point, quality control to assess the duplication rate can be performed. First, the SAM files are converted to sorted BAM files required by dupRadar tool.

```
cat ../scripts/sam-to-bam.sh

## #!/bin/bash
## #
## #Script to convert sam files to bam files
## #
## for sam_file in ../data/processed_data/bru-mor.sam/*sam; do
##   sam_file_name=$(echo $sam_file | cut -f1 -d '.')
##   samtools view -S -b $sam_file > ${sam_file_name}.bam
## done
##
## # move the created bam files to a new directory
## mkdir -p ../data/processed_data/bru-mor.bam
##
```

```
## mv ../data/processed_data/bru-mor.sam/*.bam ../data/processed_data/bru-mor.bam/
The BAM files are then sorted using samtools
cat ../scripts/sort_bam.sh
```

```
## #!/bin/bash
## #
## #Script to sort BAM file
## #
## for bam_file in ../data/processed_data/bru-mor.bam/*.bam; do
##     bam_file_name=$(echo $bam_file | cut -f1 -d '.')
##     samtools sort $bam_file -o ${bam_file_name}.sorted.bam
## done
```

Next, duplicates are marked in the BAM files using Picard.

```
cat ../scripts/mark_dupes.sh
```

```
## #!/bin/bash
## #
## #Script to mark duplicates in BAM files using picard
## #
## for bam_file in ../data/processed_data/bru-mor.bam/*.sorted.bam; do
##     bam_file_name=$(echo $bam_file | cut -f1 -d '.')
##     /opt/apps/picard-tools/1.119/bin/MarkDuplicates \
##         I=$bam_file \
##         O=${bam_file_name}.dupMarked.bam \
##         M=${bam_file_name}.dupMetrics.txt
## done
```

At this point, dupRadar tool is used to perform quality control in R. Before this quality control can be performed, we need to verify whether the reads are stranded or not as this is a required parameter for dupRadar as well as HTSeq tool later in the analysis. This can be done using RSeQC package - An RNA-seq Quality Control Package. `infer_experiment.py` module is used in this case. RSeQC documentation and tutorial can be found here.

First we convert *T. brucei* genome annotation GTF file into `bed` format required by RSeQC package. Then we use `infer_experiment.py` to verify strandedness using a few samples.

```
# convert GTF genome annotation to BED format using a custom script from:
#https://github.com/ExpressionAnalysis/ea-utils/tree/master/clipper
```

```
../scripts/gtf2bed.pl ../data/tbrucei_genome_annotations_GTF/TriTrypDB-43_TbruceiTREU927.gtf > \
../data/tbrucei_genome_annotations_GTF/TriTrypDB-43_TbruceiTREU927.bed

infer_experiment.py -i ../data/processed_data/bru-mor.bam/SRR039381.bam \
-r ../data/tbrucei_genome_annotations_GTF/TriTrypDB-43_TbruceiTREU927.bed

# output
#This is SingleEnd Data
#Fraction of reads failed to determine: 0.0008
#Fraction of reads explained by "++,--": 0.3832
#Fraction of reads explained by "+-,+-": 0.6161
```

The next step is to run the dupRadar quality control analysis setting the `stranded` parameter as FALSE as the reads are not strand specific. Tutorial for the tool can be found here

```

# Parameters
bam_file <- "../data/processed_data/bru-mor_bam/SRR039951.dupMarked.bam"
gtf_file <- "../data/brucei-morsitans/brucei-morsitans_annotations.gtf"
stranded <- 0
paired <- FALSE
threads <- 12

# Duplication rate analysis
dm <- analyzeDuprates(bam_file, gtf_file, stranded, paired, threads)

#Plots
png(filename = "../figures/duplication_rate/SRR039951.png")
duprateExpDensPlot(DupMat=dm)
title("SRR039951")
dev.off()

# Boxplot
duprateExpBoxplot(DupMat=dm)

```

Below are representative plots of samples with PCR duplicates and without PCR duplicates. Samples with PCR duplicates are removed from the analysis before counts data are read into R.

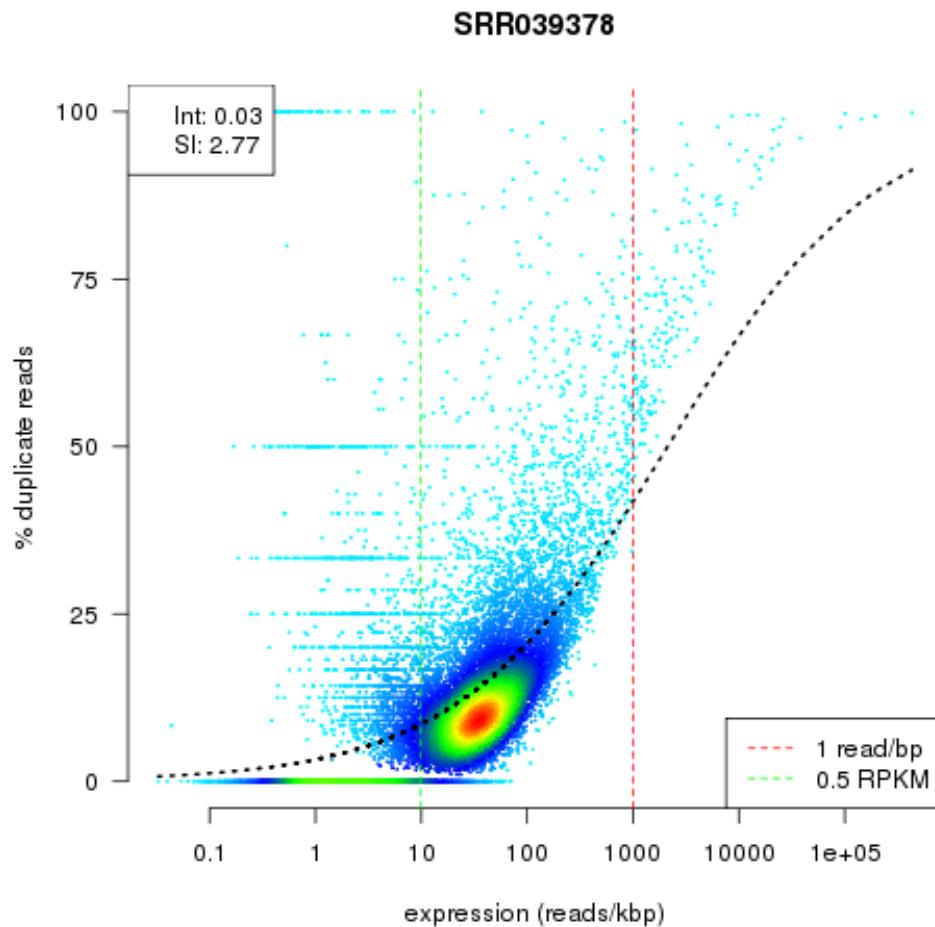


Figure 1: Sample without PCR duplicates

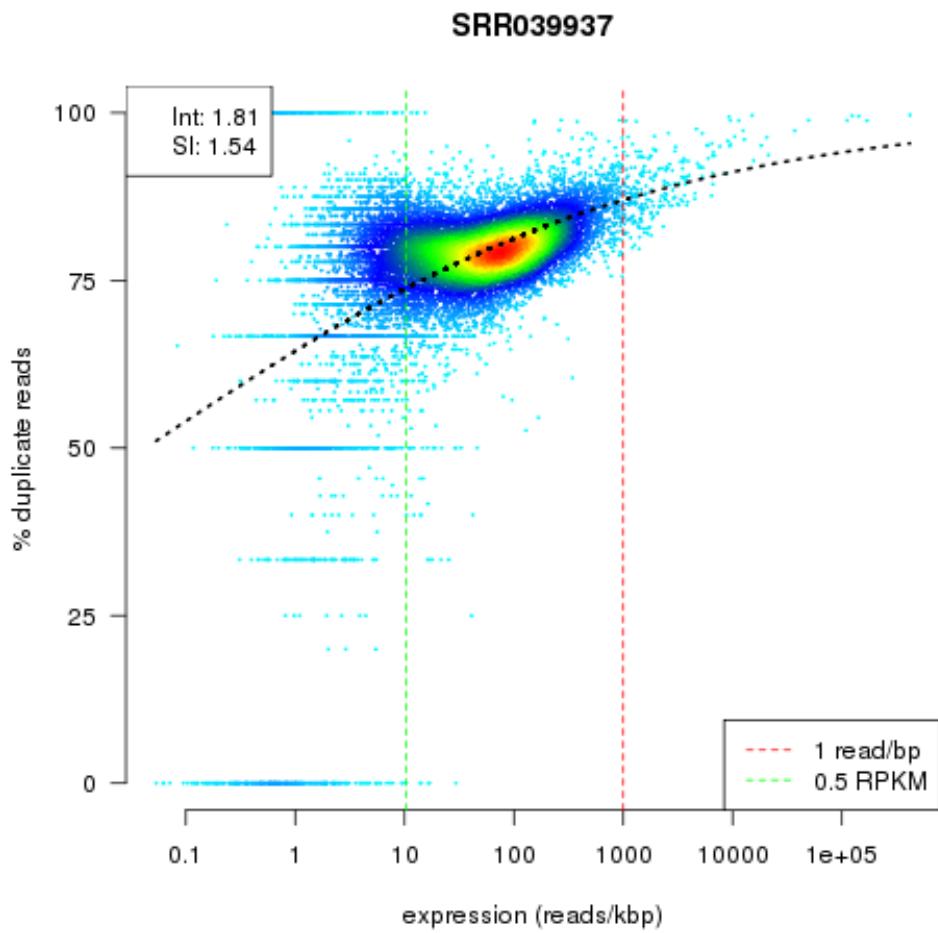


Figure 2: Sample with PCR duplicates

Reads quantification

HTSeq tool is used to count reads that aligned to the *T. brucei* genome. *T. brucei* annotation file is used and therefore HTSeq excludes counting *G. morsitans* reads that aligned to *Glossina* genome. The output is a text file for each sample that contains the number of reads that were counted for each gene.

```
cat ../scripts/htseq_counts.sh

## ##!/bin/bash
## #
## #Script to counts the number of reads aligned to T. brucei genome using HTSeq.
## #Resource: HTSeq documentation https://htseq.readthedocs.io/en/latest/count.html
## #
## module load htseq/0.11.2
##
## #create output directory if it doesn't exist
## mkdir -p ../results/brucei-HTSeq_count_results
##
## GFF_FILE=$1
##
## for bam_file in ../data/processed_data/bru-mor_bam/*.bam; do
##     bam_file_name=$(echo $bam_file | cut -f1 -d '.')
##     ##
##     python /opt/apps/htseq/0.11.2/bin/htseq-count \
##         -f bam \
##         -s no \
##         -t exon \
##         -i Parent \
##         $bam_file \
##         $GFF_FILE \
##         > ../results/brucei-HTSeq_count_results/${bam_file_name}.counts.txt
## done
```

Below is a graphical visualization of reads assignment by HTSeq.

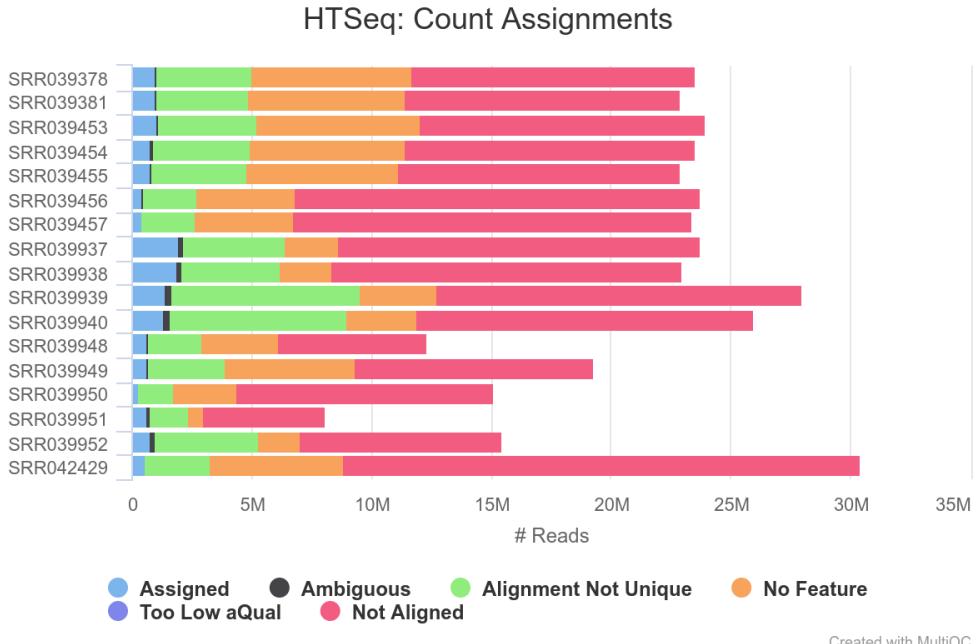


Figure 3: HTSeq reads assignment

Generating MultiQC report

MultiQC aggregates results from FASTQC, HISAT2 and HTSeq analysis into an HTML formatted single report for better visualization.

```
#change directory to results
cd ../results

#Run multiqc
multiqc .

# create a directory for the multiQC report and move the output there.
mkdir -p brucei_multiqc_report
mv multiqc* brucei_multiqc_report/
```

Filtering out non-protein coding genes

Before loading the data into R, filter out the non-protein coding genes which include ncRNA, snRNA, snoRNA, pseudogenic transcripts, rRNA and tRNA.

```
cat ../scripts/exclude_features.sh

## #!/bin/bash
## # script to exclude gene features from reads counts
##
## # create directory for the filtered counts
## mkdir -p ../results/brucei_HTSeq_count_results_mRNA
##
## for file in ../results/brucei_HTSeq_count_results/*.counts.txt; do
##   counts_file=$(basename "$file" .counts.txt)
```

```

##      grep -v -f ../results/excluded_features.txt ${file} > \
##          ../results/brucei_HTS defense_count_results_mRNA/"${counts_file}" .counts.txt
## done

```

Analysis in R

Importing samples count data into R

For further analysis, samples read counts are read into R. To read the sample counts data into R using the script below, simply type `source("../scripts/htseq-combine_all.R")` on the R console and hit enter. Here, ensure the samples to be excluded in the analysis (SRR039951, SRR039937 and SRR039938) are not among the input files.

```

cat ../scripts/htseq-combine_all.R

## #!/usr/bin/Rscript
##
## # Take 'all' htseq-count results and melt them in to one big dataframe
##
## #Adapted from: https://wiki.bits.vib.be/index.php/NGS_RNASeq_DE_Exercise.4
##
## # where are we?
## basedir <- "../results"
## setwd(basedir)
##
## cntdir <- paste(basedir, "brucei_HTS defense_count_results_mRNA", sep="/")
## pat <- ".counts.txt"
## hisat2.all <- list.files(path = cntdir,
##                           pattern = pat,
##                           all.files = TRUE,
##                           recursive = FALSE,
##                           ignore.case = FALSE,
##                           include.dirs = FALSE)
##
## # we choose the 'all' series
## myfiles <- hisat2.all
## DT <- list()
##
## # read each file as array element of DT and rename the last 2 cols
## # we created a list of single sample tables
## for (i in 1:length(myfiles) ) {
##   infile = paste(cntdir, myfiles[i], sep = "/")
##   DT[[myfiles[i]]] <- read.table(infile, header = F, stringsAsFactors = FALSE)
##   cnts <- gsub("(.*).counts.txt", "\\\1", myfiles[i])
##   colnames(DT[[myfiles[i]]]) <- c("ID", cnts)
## }
##
## # merge all elements based on first ID columns
## data <- DT[[myfiles[1]]]
##
## # inspect
## #head(data)
##
## # we now add each other table with the ID column as key

```

```

## for (i in 2:length(myfiles)) {
##   y <- DT[[myfiles[i]]]
##   z <- merge(data, y, by = c("ID"))
##   data <- z
## }
##
## # ID column becomes rownames
## rownames(data) <- data$ID
## data <- data[,-1]
##
## ## add total counts per sample
## data <- rbind(data, tot.counts=colSums(data))
##
## # inspect and look at the top row names!
## #head(data)
##
## #tail(data)
##
######
## # take summary rows to a new table
## # ( not starting with Tb and tmp with invert=TRUE )
##
## # transpose table for readability
## data.all.summary <- data[grep("^Tb|^tmp", rownames(data), perl=TRUE, invert=TRUE), ]
##
## # review
## #data.all.summary
##
## # transpose table
## t(data.all.summary)
##
## # write summary to file
## write.csv(data.all.summary, file = "brucei_htseq_counts_all-summary.csv")
##
######
## # take all data rows to a new table
##
## data.all <- data[grep("^Tb|^tmp", rownames(data), perl=TRUE, invert=FALSE), ]
##
## # inspect final merged table
## #head(data.all, 3)
##
## # write data to file
## write.table(data.all, file = "brucei_htseq_counts_all.txt", quote = FALSE, sep = "\t")
##
## # cleanup intermediate objects
## rm(y, z, i, DT)
##
## #return to analysis directory
## setwd("../analysis")

```

Gene IDs and Transcript IDs

```
# for changing transcript ids to corresponding gene ids
gtf_file <- import("../data/TriTrypDB-43_TbruceiTREU927.gtf")

gene_and_transcript_id <- mcols(gtf_file)[,c("gene_id","transcript_id")]

gene_and_transcript_id <- unique(gene_and_transcript_id)
```

Sample quality check

The quality of the samples is checked before further analysis to check for outlier and batch effects.

```
# Remove extra column from sample SRR965341 added while reading data into R
data.all["Var.2"] <- NULL

# Create a DGEList object
counts <- DGEList(data.all, group = sample.metadata$Tissue)

# check the number of genes with no expression in all samples
table(rowSums(counts$counts==0)==15)
#FALSE TRUE
# 9184 792
#####
#used the alternative function below instead of this
#
# filter out Non-expressed genes and those with low expression values.
# keep genes with at least 1 read per million in n samples, where n
# is minimum number of replicates which is 3 in this case
keep <- rowSums(cpm(counts)>1) >= 3

# retain genes above the cpm value
#filtered.counts <- counts[keep, , keep.lib.sizes=FALSE]
#####

# Filtering non-expressed and lowly-expressed genes.
#
# Alternative filtering function that filters better (Law et al 2016)
# retain genes above a calculated cpm value
keep.exprs <- filterByExpr(counts, group=sample.metadata$Sample_Name)
filtered.counts <- counts[keep.exprs,, keep.lib.sizes=FALSE]

# replace transcript ids with gene ids as rownames
filtered.counts.tmp <- tibble::rownames_to_column(as.data.frame(filtered.counts$counts),
                                                 "transcript_id")

filtered.counts.tmp$gene_id <- gene_and_transcript_id$gene_id[match(filtered.counts.tmp$transcript_id,
                                                               gene_and_transcript_id$transcript_id)]

filtered.counts.tmp <- as.data.frame(filtered.counts.tmp) %>% remove_rownames %>%
  column_to_rownames(var = "gene_id")

filtered.counts.tmp$transcript_id <- NULL
```

```

filtered.counts$counts <- as.matrix(filtered.counts.tmp)

# obtain logCPM unnormalized for plotting purposes.
# Here, the norm.factors value is 1 for all samples
logcpm.unnorm.counts <- cpm(filtered.counts, log = TRUE, prior.count = 2, normalized.lib.sizes = TRUE)

# Normalize for composition bias using TMM
filtered.counts <- calcNormFactors(filtered.counts, method = 'TMM')

# Convert counts per million per gene to log counts per million for further downstream analysis.
logcpm.norm.counts <- cpm(filtered.counts, log = TRUE, prior.count = 2, normalized.lib.sizes = TRUE)

# use ComBat to remove batch effects
modcombat <- model.matrix(~Tissue, data=sample.metadata)
logcpm.norm.counts.combat <- ComBat(dat=logcpm.norm.counts, batch = sample.metadata$Batch,
                                     mod = modcombat)

```

Various plots are made for the samples before and after normalization.

Samples heatmap

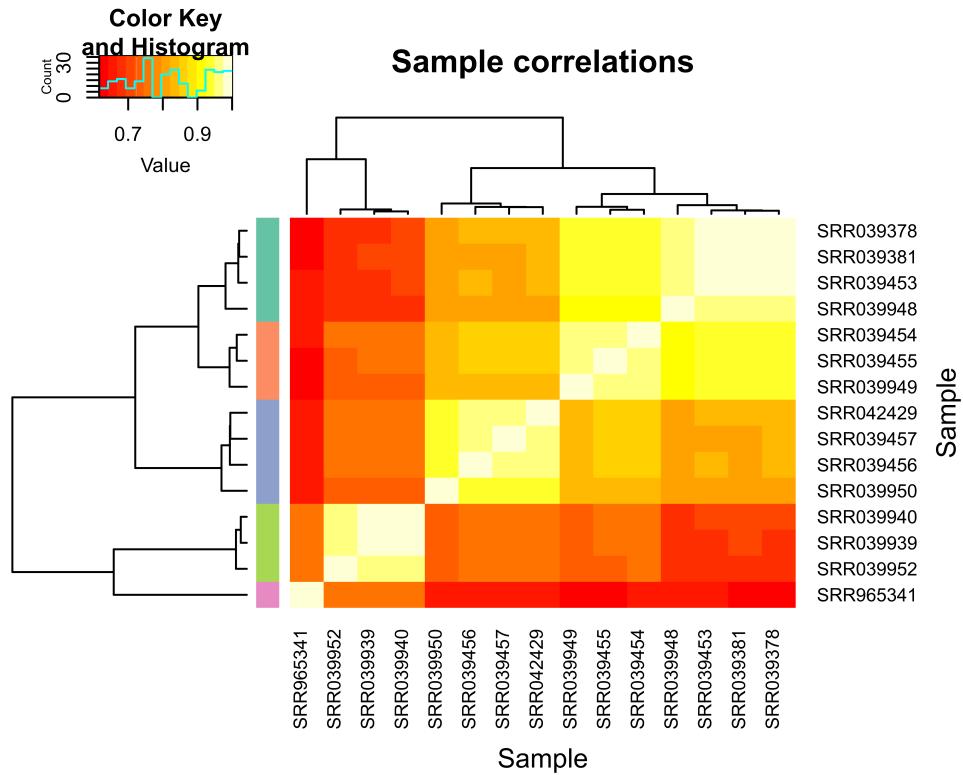


Figure 4: Normalized sample correlation heatmap

Samples density plot

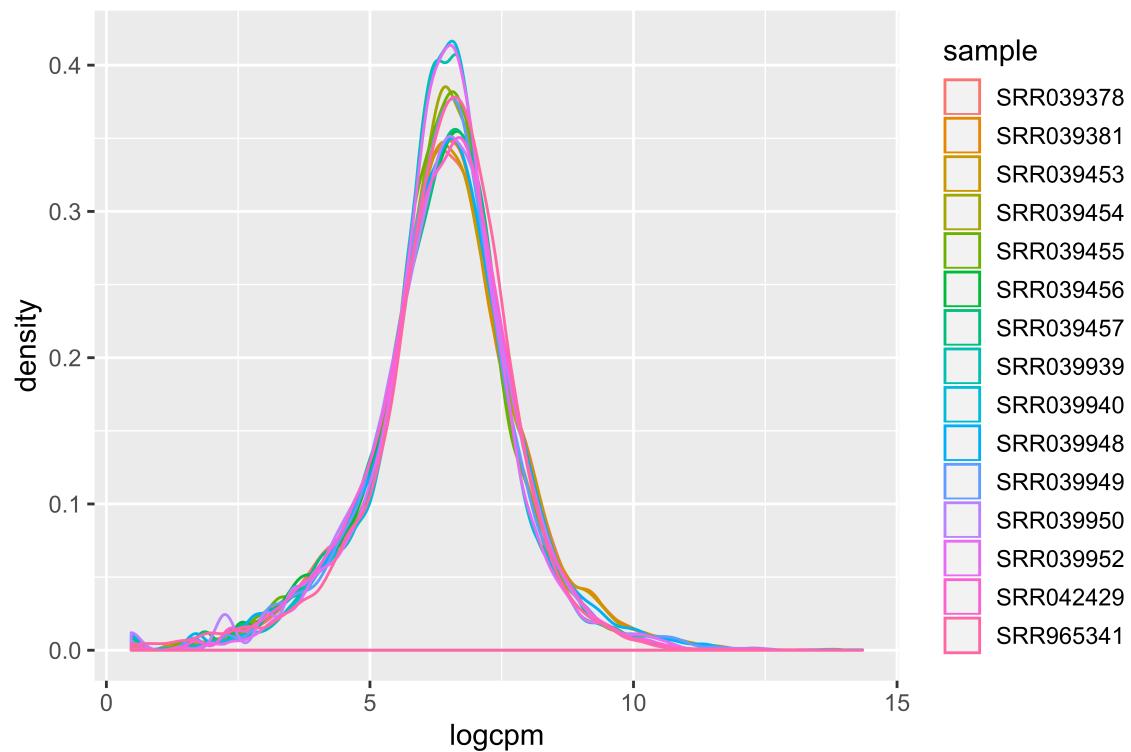


Figure 5: Normalized sample density plot

Principal component analysis

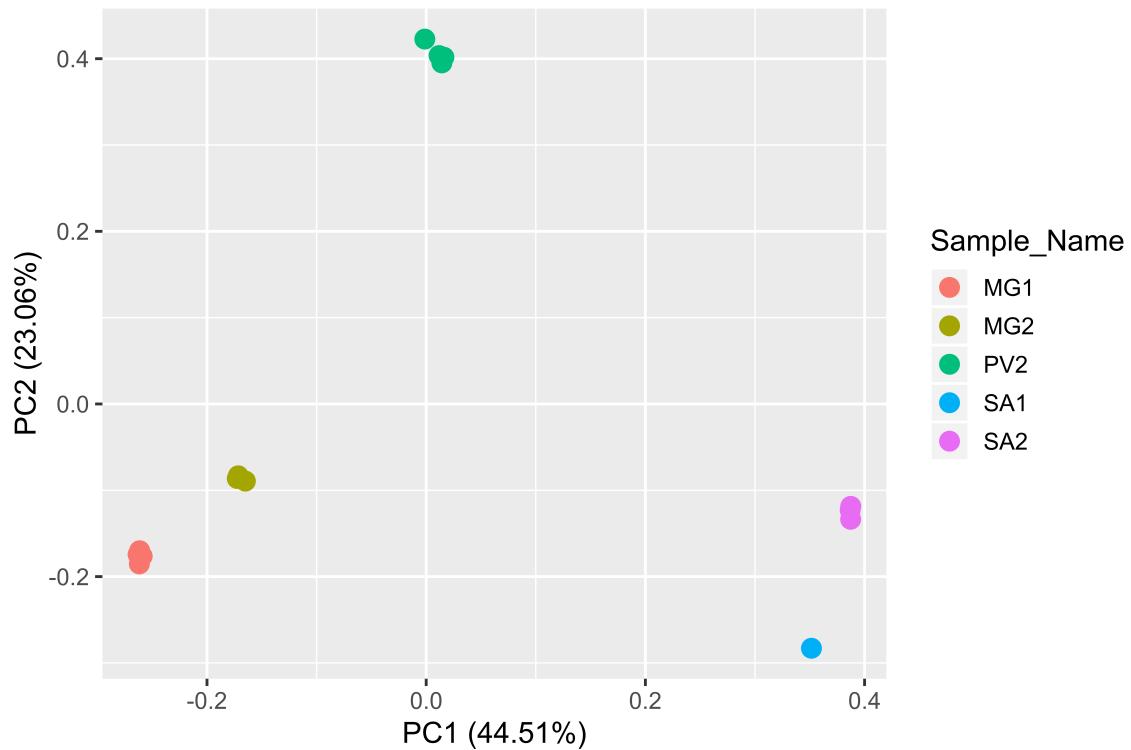


Figure 6: Normalized sample PCA

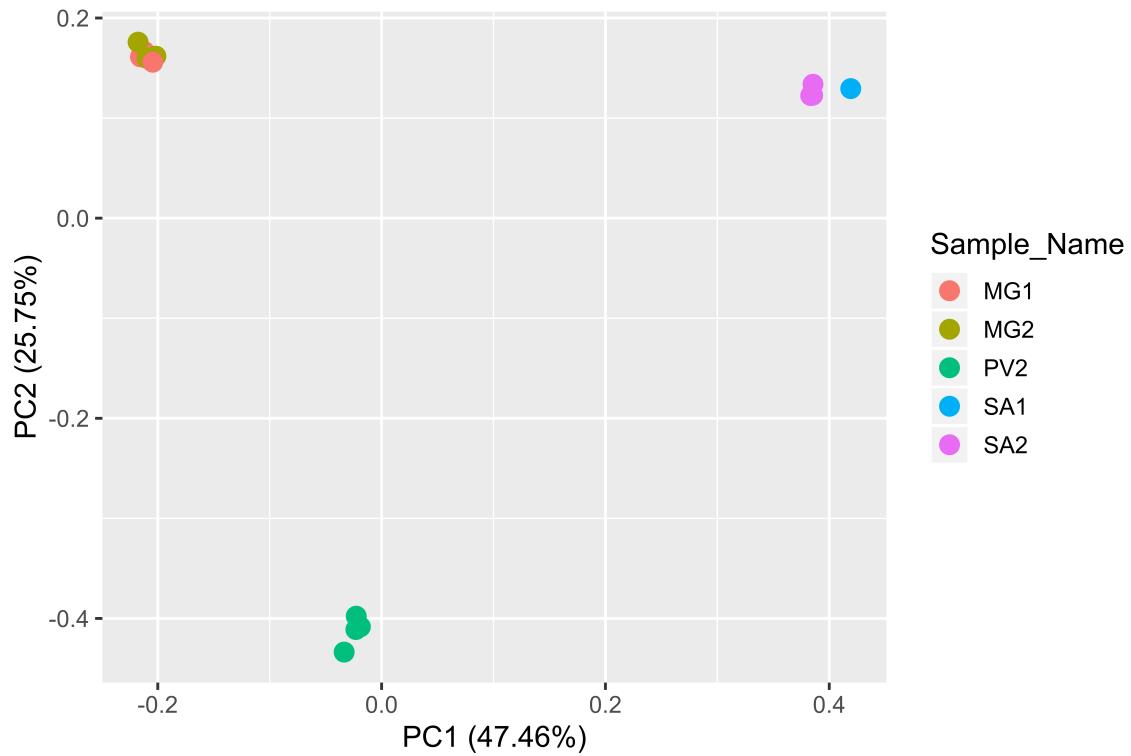


Figure 7: Normalized and batch effect removed sample PCA

Boxplot

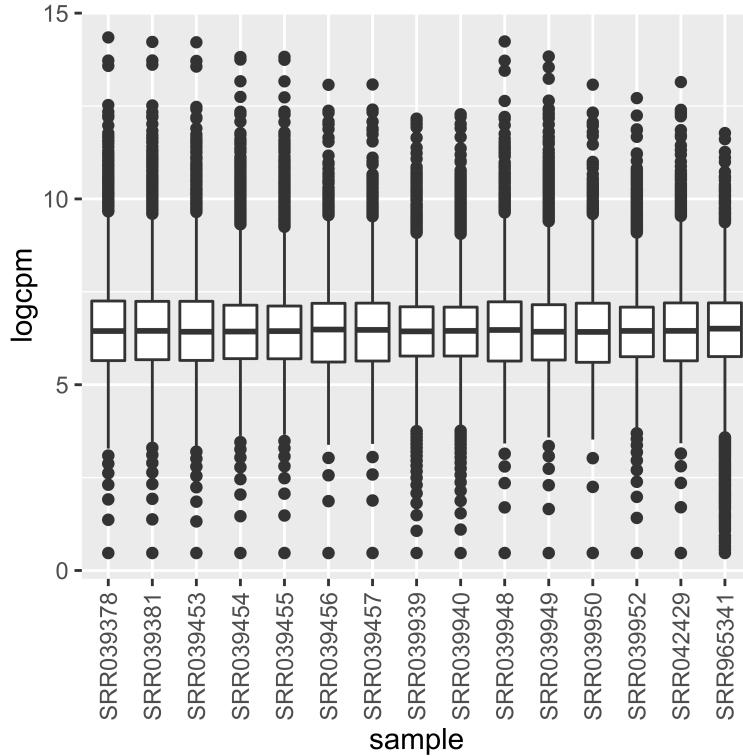


Figure 8: Normalized sample boxplot

Identify differentially expressed genes

```

# Apply sample grouping based on Tissue from which the sample was derived
design <- model.matrix(~0+sample.metadata$Tissue+sample.metadata$Batch)
#colnames(design) <- levels(sample.metadata$Tissue)
colnames(design) <- c("MG", "PV", "SG", "Batch")

# Estimate dispersions for tags
filtered.counts.dge <- estimateDisp(filtered.counts, design, robust = TRUE)

# Fit a generalized likelihood model to the DGELIST using sample grouping
fit <- glmFit(filtered.counts.dge, design)

#####
# code in this section adapted from https://github.com/iscb-dc-rsg/2016-summer-workshop
# generate a list of all possible pairwise contrasts
condition_pairs <- t(combn(levels(sample.metadata$Tissue), 2))

comparisons <- list()
for (i in 1:nrow(condition_pairs)) {
  comparisons[[i]] <- as.character(condition_pairs[i,])
}

# remove MG to SG comparison
comparisons[[2]] <- NULL

```

```

# vector to store differentially expressed genes
sig_genes <- c()

# iterate over the contrasts, and perform a differential expression test for
# each pair
for (conds in comparisons) {
  # generate string contrast formula
  contrast_formula <- paste(conds, collapse=' - ')

  contrast_mat <- makeContrasts(contrasts=contrast_formula, levels=design)
  contrast_lrt <- glmLRT(fit, contrast=contrast_mat)
  topGenes <- topTags(contrast_lrt, n=Inf, p.value=0.05, adjust.method = "BH")

  # Grab highly ranked genes
  sig_genes <- union(sig_genes, rownames(topGenes$table))
}

# Filter out genes which were not differentially expressed for any contrast
de.genes <- filtered.counts.dge[rownames(filtered.counts.dge) %in% sig_genes,]
dim(de.genes$counts)
#3934    15
#####
#####

# Obtain the counts of genes expressed for each contrast individually
# This aims to obtain the number of genes differentially expressed between
# the 3 stages of development i.e. MG -> PV, PV -> SG

# Likelihood ratio test to identify DEGs
# SG compared to PV
SG_vs_PV_lrt <- glmLRT(fit, contrast=c(0,-1,1,0))

# PV compared to MG
PV_vs_MG_lrt <- glmLRT(fit, contrast = c(-1,1,0,0))

# Genes with most significant differences (using topTags)
# SG compared to PV
topGenes_SG <- topTags(SG_vs_PV_lrt, adjust.method = "BH", p.value = 0.05, n=Inf)
dim(topGenes_SG)
#3166      5

# PV compared to MG
topGenes_PV <- topTags(PV_vs_MG_lrt, adjust.method = "BH", p.value = 0.05, n=Inf)
dim(topGenes_PV)
#1908      5

#Total number of genes: 5074
#####
# DE genes at 5% FDR (using decideTestsDGE function)
#
# SG compared to PV
SG_vs_PV_de.genes <- decideTestsDGE(SG_vs_PV_lrt, adjust.method = "BH", p.value = 0.05)

```

```

# get summary
summary(SG_vs_PV_de.genes)
#      -1*PV 1*SG
#Down      1463
#NotSig    4224
#Up       1703

# PV compared to MG
PV_vs_MG_de.genes <- decideTestsDGE(PV_vs_MG_lrt, adjust.method = "BH", p.value = 0.05)

# summary
summary(PV_vs_MG_de.genes)
#      -1*MG 1*PV
#Down      987
#NotSig   5482
#Up       921

# DE genes in the PV that are common in both comparisons
de.common <- which(PV_vs_MG_de.genes[,1] != 0 & SG_vs_PV_de.genes[,1] != 0)
length(de.common)
#1140

de.common.df <- as.data.frame(de.common)
de.common.df <- tibble::rownames_to_column(de.common.df, var = "gene_id")

# create a dataframe with data on PV and SG differential gene expression
PV_data <- topGenes_PV$table
SG_data <- topGenes_SG$table

PV_data <- tibble::rownames_to_column(PV_data, var = "gene_id")
SG_data <- tibble::rownames_to_column(SG_data, var = "gene_id")

# obtain the common genes for each comparison
PV_data_common_de_genes <- PV_data %>% filter(gene_id %in% de.common.df$gene_id)
SG_data_common_de_genes <- SG_data %>% filter(gene_id %in% de.common.df$gene_id)

PV_data_common_de_genes <- PV_data_common_de_genes[order(PV_data_common_de_genes$logFC,
                                                       decreasing = TRUE),]
SG_data_common_de_genes <- SG_data_common_de_genes[order(SG_data_common_de_genes$logFC,
                                                       decreasing = TRUE),]

# write out to excel upregulated and downregulated genes and the common genes between contrasts
PV_vs_MG <- PV_data[order(PV_data$logFC, decreasing = TRUE),]
SG_vs_PV <- SG_data[order(SG_data$logFC, decreasing = TRUE),]

write.xlsx(PV_vs_MG, file = "../results/differentially_expressed_genes.xlsx",
          sheetName = "MG vs PV", row.names = FALSE)
write.xlsx(SG_vs_PV, file = "../results/differentially_expressed_genes.xlsx",
          sheetName = "PV vs SG", append = TRUE, row.names = FALSE)

write.xlsx(PV_data_common_de_genes,
          file = "../results/differentially_expressed_genes_common_in_contrasts.xlsx",
          sheetName = "MG vs PV Common genes", append = TRUE, row.names = FALSE)

```

```

write.xlsx(SG_data_common_de_genes,
          file = "../results/differentially_expressed_genes_common_in_contrasts.xlsx",
          sheetName = "PV vs SG Common genes", append = TRUE, row.names = FALSE)

PV_contrast_common_de_genes_logFC <- PV_data_common_de_genes %>% select(gene_id, logFC_MGvsPV=logFC)

SG_contrast_common_de_genes_logFC <- SG_data_common_de_genes %>% select(gene_id, logFC_PVvsSG=logFC)

contrast_common_de_genes_logFC <- merge(PV_contrast_common_de_genes_logFC,
                                         SG_contrast_common_de_genes_logFC, by="gene_id")

write.xlsx(contrast_common_de_genes_logFC,
          file = "../results/differentially_expressed_genes_common_in_contrasts_logFC_comparison.xlsx",
          sheetName = "Common DEGs LogFC Comparisons", row.names = FALSE)

```

Plotting to visually inspect differential gene expression results.

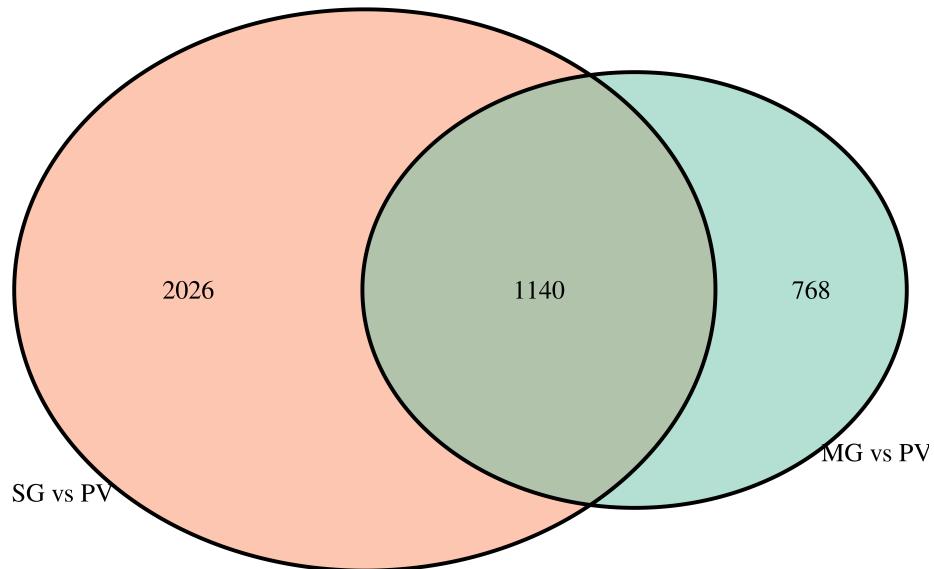


Figure 9: Venn diagram showing differentially expressed genes

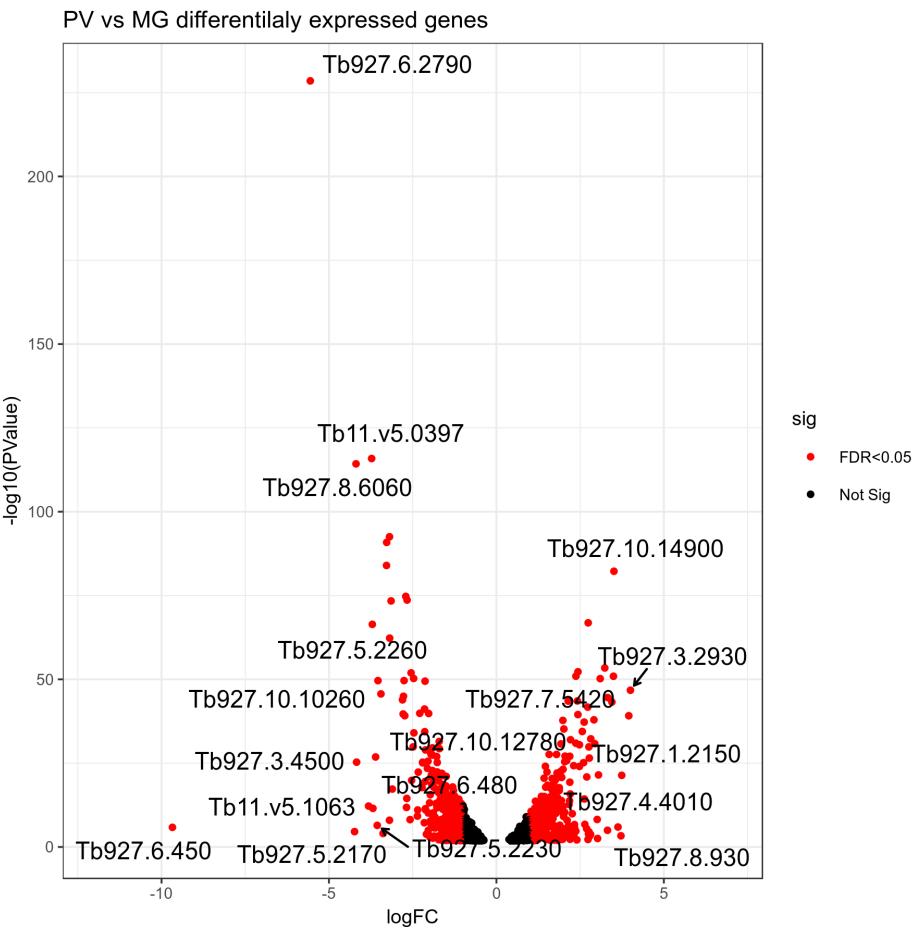


Figure 10: Differentially expressed genes in the proventriculus compared to midgut

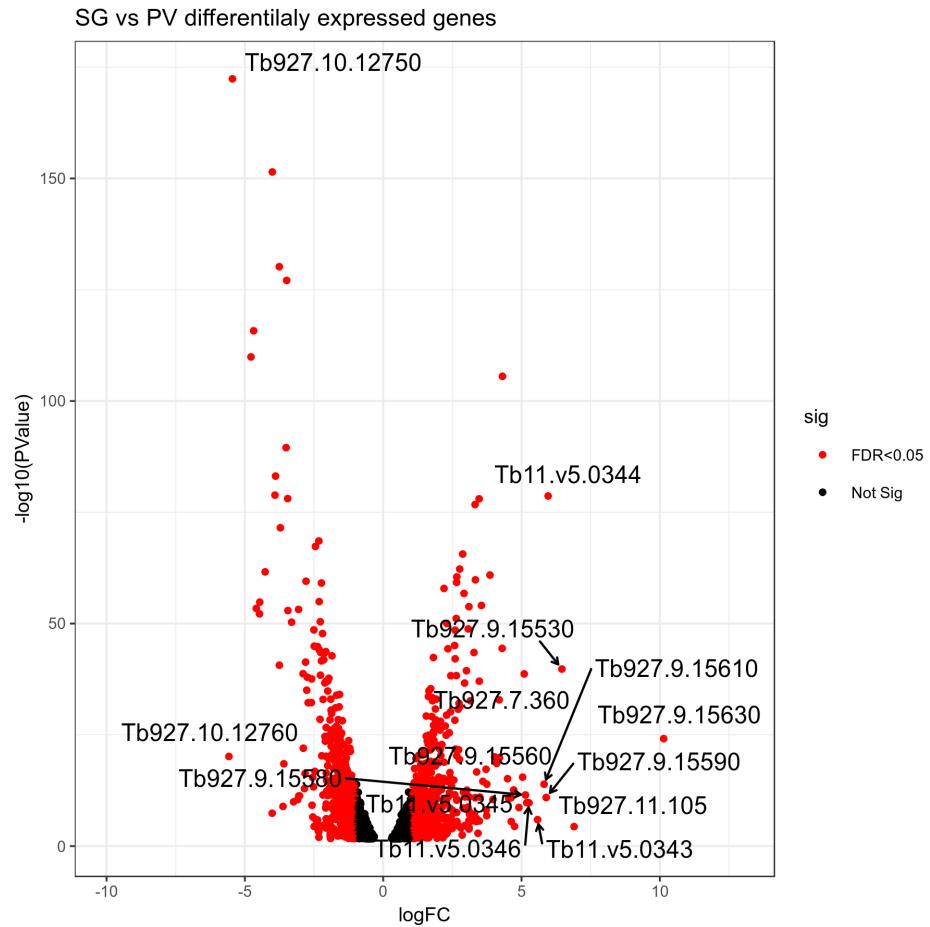


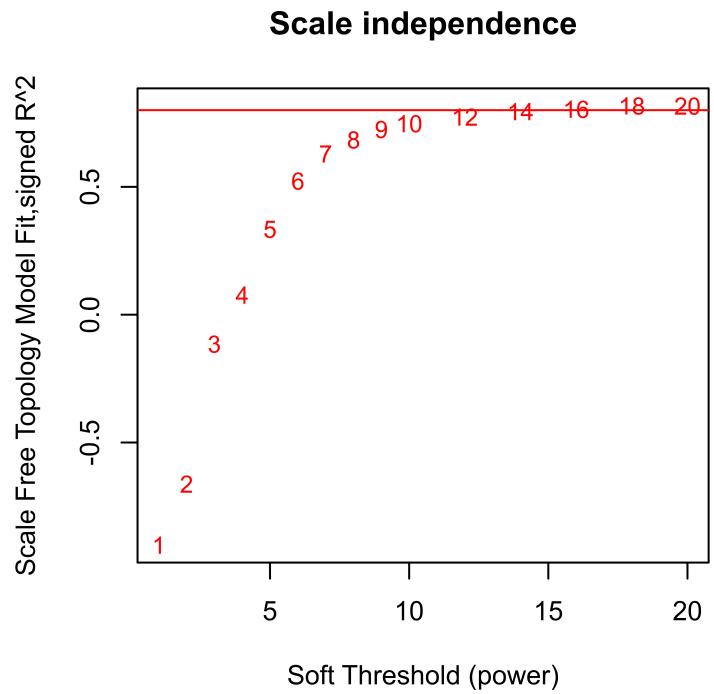
Figure 11: Differentially expressed genes in the salivary gland compared to proventriculus

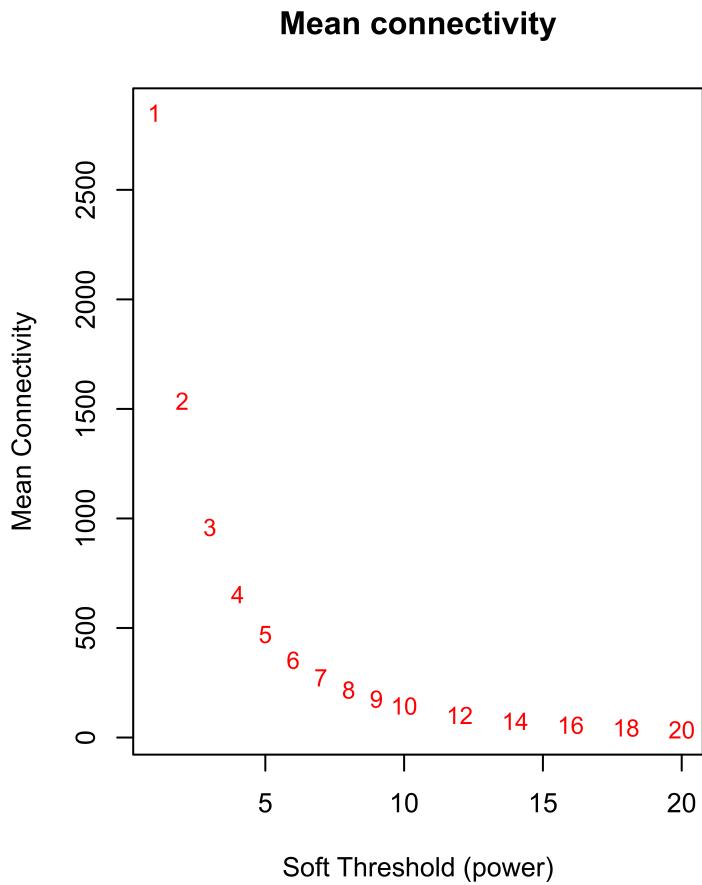
Weighted gene co-expression analysis

```
# obtain the required counts data (WGCNA input)
# WGCNA requires genes to be in columns
network.counts <- t(logcpm.norm.counts.combat)

# determine the soft-thresholding power to use
powers <- c(c(1:10), seq(from = 12, to=20, by=2))
sft <- pickSoftThreshold(network.counts, powerVector = powers, verbose = 5)
```

Plots to determine the soft thresholding power to use.





```
#####
## Constructing the network
#####

# construct adjacency matrix
softpower <- 14
adjacency.matrix <- adjacency(network.counts, power=softpower,
                               type = "signed", corFnc = "cor")

# Turn the adjacency matrix to topologicaal overlap matrix to minimize
# the effects of noise and spurious associations
TOM <- TOMsimilarity(adjacency.matrix, TOMType = "signed")
dissTOM <- 1 - TOM

#set diagonal to NA to remove uninformative correlations
diag(adjacency.matrix) <- NA

# Adjacency matrix heatmap plot / network heatmap of selected genes
heatmap_indices <- sample(nrow(adjacency.matrix), 500) # sub-sample for visualization purposes

png(filename = "../figures/adjacency_matrix_heatmap.png", res = 1200, type = "cairo", units = 'in',
    width = 5, height = 5, pointsize = 10)
```

```

heatmap.2(t(adjacency.matrix[heatmap_indices, heatmap_indices]),
  col=redgreen(75),
  labRow=NA, labCol=NA,
  trace='none', dendrogram='row',
  xlab='Gene', ylab='Gene',
  main='Adjacency matrix',
  density.info='none', revC=TRUE)
dev.off()

```

```

# remove adjacency matrix and TOM to free up memory
rm(adjacency.matrix)
gc()

```

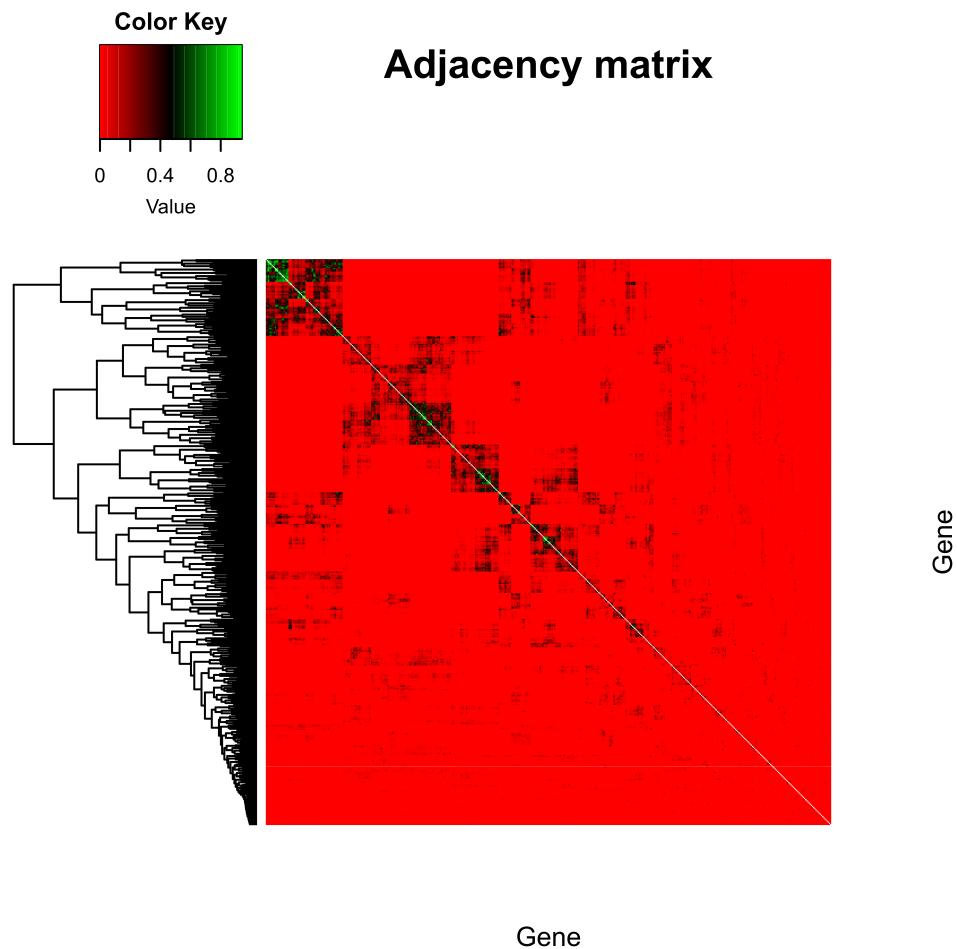


Figure 12: Adjacency matrix heatmap (500 genes)

```

#####
## Detecting co-expression modules in R
#####

# view the dendrogram based on hierarchical clustering of genes
gene.tree <- flashClust(as.dist(dissTOM), method = "average")

# plot the gene tree
png(filename = "../figures/gene_tree.png", res = 1200, type = "cairo", units = 'in',
    width = 7, height = 8, pointsize = 10)
#sizeGrWindow(12,9) #open graphical window
plot(gene.tree, xlab="", sub="", main = "Gene clustering based on TOM dissimilarity",
      labels = FALSE, hang = 0.04)
dev.off()

# identify the modules
module.labels <- cutreeDynamicTree(gene.tree, deepSplit = FALSE,
                                      minModuleSize = 30)

#view
table(module.labels)

# convert labels to colours
module.colours <- labels2colors(module.labels)

# view
table(module.colours)

# a list of 27 modules



|             |            |              |               |            |
|-------------|------------|--------------|---------------|------------|
| black       | blue       | brown        | cyan          | darkgreen  |
| 438         | 614        | 547          | 219           | 102        |
| darkgrey    | darkorange | darkred      | darkturquoise | green      |
| 93          | 80         | 106          | 100           | 528        |
| greenyellow | grey       | grey60       | lightcyan     | lightgreen |
| 251         | 59         | 191          | 193           | 164        |
| lightyellow | magenta    | midnightblue | orange        | pink       |
| 129         | 264        | 200          | 86            | 383        |
| purple      | red        | royalblue    | salmon        | tan        |
| 251         | 460        | 127          | 230           | 243        |
| turquoise   | white      | yellow       |               |            |
| 732         | 61         | 539          |               |            |



# visualize the gene tree and TOM matrix together using TOM plot
# if necessary, raise dissTOM to a power to make moderately strong connection more visible in heatmap
diag(dissTOM) <- NA

png(filename = "../figures/gene_tree_and_dissTOM.png", res = 1200, type = "cairo", units = 'in',
    width = 5, height = 6, pointsize = 10)
TOMplot(dissTOM, gene.tree, as.character(module.colours))
dev.off()

# remove matrix to free memory

```

```

rm(dissTOM)
gc()

# plot gene dendrogram
png(filename = "../figures/gene_tree_and_colours.png", res = 1200, type = "cairo", units = 'in',
    width = 6, height = 6, pointsize = 10)
#sizeGrWindow(8,6) #open graphical window
plotDendroAndColors(gene.tree, module.colours, "Dynamic Tree Cut", dendroLabels = FALSE,
                     hang = 0.03, addGuide = TRUE, guideHang = 0.05,
                     main = "Gene dendrogram and module colours")
dev.off()

# get hub genes
# choose power 4: https://support.bioconductor.org/p/46342/
module.hub.genes <- chooseTopHubInEachModule(network.counts, module.colours,
                                              power = 4, type = "signed")

# A list of module hub genes
  black           blue          brown         cyan
  "Tb927.7.1790"  "Tb927.8.3620"  "Tb927.11.1570"  "Tb927.2.5530"
  darkgreen       darkgrey      darkorange      darkred
  "Tb927.11.14020"  "Tb927.9.9450"  "Tb927.8.710"   "Tb927.2.5270"
  darkturquoise     green        greenyellow     grey60
  "Tb927.8.6650"  "Tb927.10.13790" "Tb927.7.920"   "Tb927.10.3820"
  lightcyan        lightgreen    lightyellow    magenta
  "Tb927.11.6440"  "Tb927.10.720"   "Tb927.10.2560" "Tb927.9.6290"
  midnightblue      orange        pink          purple
  "Tb927.11.6640"  "Tb927.9.2520"   "Tb927.10.6200" "Tb927.1.600"
  red             royalblue      salmon          tan
  "Tb927.7.6920"  "Tb927.10.15680" "Tb927.11.1450" "Tb927.3.2930"
  turquoise        white         yellow
  "Tb927.9.15630"  "Tb927.8.7980"   "Tb927.1.3550"

```

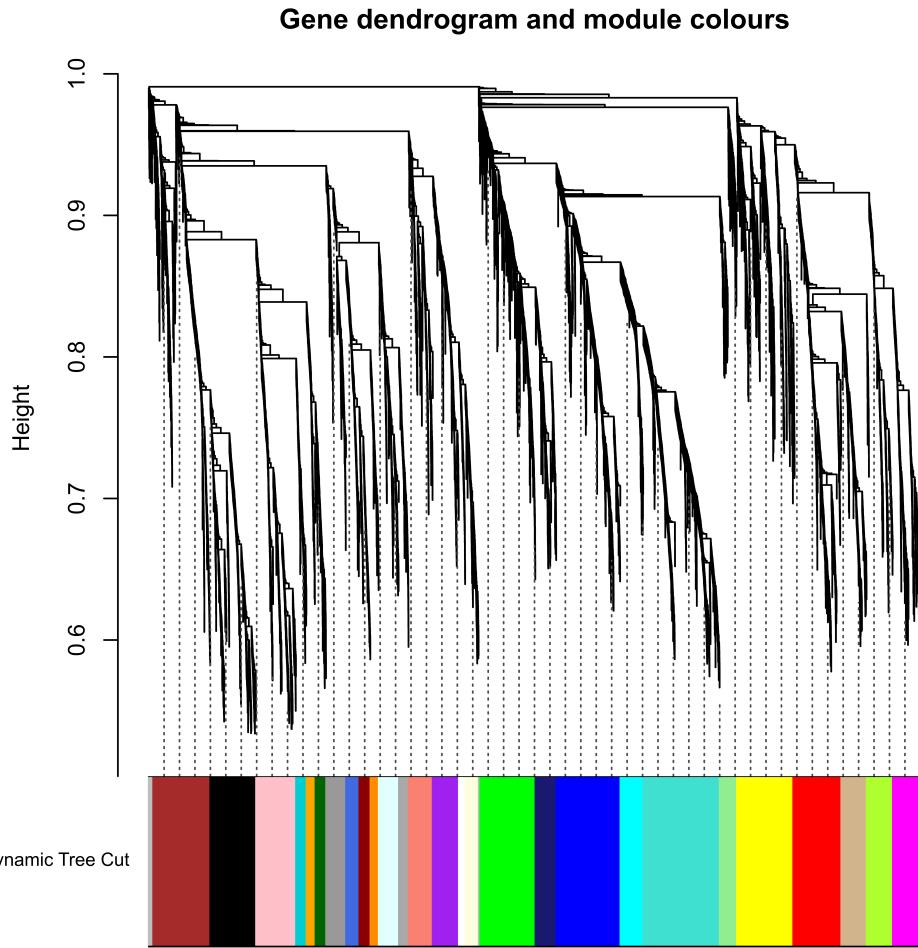


Figure 13: Gene tree and module colours

```
#####
## Network export to cytoscape
#####

# select modules of interest
interesting.modules <- c('black', 'cyan', 'grey', 'brown',
                         'midnightblue', 'blue', 'darkgreen', 'tan', 'darkgrey', 'darkorange',
                         'darkred', 'darkturquoise', 'green', 'greenyellow', 'grey60', 'lightcyan',
                         'lightgreen', 'lightyellow', 'magenta', 'orange', 'pink', 'purple', 'red',
                         'royalblue', 'salmon', 'turquoise', 'white', 'yellow') # all module colours, thus

# enriched modules
enriched.modules <- c("black", "tan", "brown", "blue", "turquoise", "magenta", "darkturquoise",
                      "green", "red", "pink", "salmon", "lightyellow", "purple", "greenyellow")

# obtain gene ids
gene.ids <- rownames(logcpm.norm.counts.combat)

# select module genes
#inModules <- is.finite(match(module.colours, interesting.modules)) # whole network modules
```

```

#inModules <- is.finite(match(module.colours, enriched.modules)) # enriched modules
inModules <- is.finite(match(module.colours, c("midnightblue", "lightcyan"))) # individual modules

modGenes <- gene.ids[inModules]

# select the corresponding dissTOM based on module genes
modTOM <- TOM[inModules, inModules]
dimnames(modTOM) <- list(modGenes, modGenes)

# Export the network into edge and node list files Cytoscape can read
exportNetworkToCytoscape(modTOM,
                         edgeFile = "../results/cytoscape_input_files/CytoscapeInput-edges_midnightblue",
                         nodeFile = "../results/cytoscape_input_files/CytoscapeInput-nodes_midnightblue",
                         weighted = TRUE,
                         threshold = 0,
                         nodeNames = modGenes,
                         nodeAttr = module.colours[inModules]);

# Also, export the network as graphml format
# use export_network_to_graphml function
source("../scripts/network_export_graphml.R")

# the whole network
entire.network <- export_network_to_graphml(TOM,
                                              filename = "../results/entire_network_thresh0.graphml",
                                              threshold = 0,
                                              nodeAttr = gene.ids,
                                              max_edge_ratio = 3)

# network modules
# create a dataframe with node attributes
enriched.module.colours <- module.colours[inModules] #get enriched module colours from module.colours
node.attributes <- cbind(modGenes, module=enriched.module.colours) # node attrr. for enriched modules

#node.attributes <- cbind(modGenes, module=module.colours) # get node attr. for whole network
node.attributes <- as.data.frame(node.attributes)

# Add RGB versions of colour modules
node.attributes$colourRGB <- col2hex(node.attributes$module)

modules.network <- export_network_to_graphml(modTOM,
                                              filename = "../results/modules_network_thresh0.02.graphml",
                                              threshold = 0.02,
                                              nodeAttrDataFrame = node.attributes)

# write out a node attributes files with hexadecimal colour names for module genes
write.table(node.attributes,
            file = "../results/cytoscape_input_files/Cytoscape_node_attributes_enriched_modules_midnightblue",
            row.names = FALSE,
            quote = FALSE, sep = "\t")

```

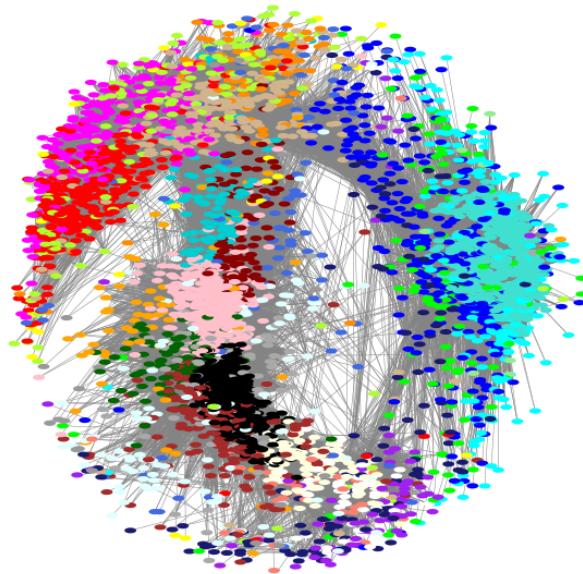


Figure 14: Co-expression subnetwork

Functional Analysis

Loading annotations from the packages

Document: manuscript-shared-rnaseq/Rmd/init/load_pathogen_annotations.Rmd

```
#####
# load gene annotations from packages
#####
orgdb <- get("Trypanosoma.brucei.TREU927")

# Fix AnnotationDbi namespace mess
assign('select', dplyr::select, envir=.GlobalEnv)
assign('get', base::get, envir=.GlobalEnv)

gene_info <- load_parasite_annotations(orgdb, rownames(logcpm.norm.counts.combat),
                                         keytype="GID")

# Get transcript lengths (sum of all exon lengths for each gene)
txdb <- orgdb@txdbSlot
transcript_lengths <- transcriptLengths(txdb)
transcript_lengths <- transcript_lengths[transcript_lengths$gene_id %in%
                                         gene_info$gene_id,]

gene_info[match(transcript_lengths$gene_id, gene_info$gene_id),
          'transcript_length'] <- transcript_lengths$tx_len
gene_info$transcript_length <- as.numeric(gene_info$transcript_length)

# A gene was excluded in the annotation database as a result of an orphan transcript. Add its placeholder
## ---not run---
gene_info <- rbind(gene_info, data.frame(gene_id='Tb927.4.4663',
                                         chromosome='4',
```

```

        description=NA, strand=NA, type=NA,
        transcript_length=NA))

# Keep only the feature information remaining genes
gene_info <- gene_info[gene_info$gene_id %in% rownames(logcpm.norm.counts.combat),] #WGCNA input counts

# For now, just grab the description for the first transcript
#gene_info <- gene_info[!duplicated(gene_info$gene_id),]

# Gene IDs
gene_ids <- rownames(logcpm.norm.counts.combat)

# gene annotations preview
kable(head(gene_info), caption='Preview of gene annotations.')

#####
# load GO terms associated with each parasite gene that were downloaded from Tritrypdb
#####
# T. brucei go term annotations file path
#go_term_mapping <- "~/tbrucei_annotation_package/build/3.6.0/TriTrypDB-43_TbruceiTREU927_go_table.txt"
#go_terms <- read.table(go_term_mapping, header = TRUE)

# load go terms from annotation package instead
go_terms <- load_go_terms(orgdb, rownames(logcpm.norm.counts.combat),
                           keytype='GID')

# Exclude genes not found in count table --not run--
#go_terms <- go_terms[go_terms$GID %in% rownames(logcpm.norm.counts.combat),]

# gene / go term mapping
gene_go_mapping <- as.data.frame(unique(go_terms %>% select(GID, GO, ONTOLOGY)))
colnames(gene_go_mapping) <- c('gene', 'category', 'ontology')

# go id / term mapping
go_term_id_mapping <- as.data.frame(unique(go_terms[c('GO', 'TERM', 'ONTOLOGY')]))
colnames(go_term_id_mapping) <- c("category", "term", "ontology")

#####
# Load KEGG annotations
#####

gene_kegg_mapping <- load_kegg_mapping(orgdb, rownames(logcpm.norm.counts.combat),
                                         keytype="GID")

kegg_pathways <- load_kegg_pathways(orgdb, rownames(logcpm.norm.counts.combat),
                                      keytype="GID")

# Rename gene/KEGG mapping columns to be consistent with GO mapping
colnames(gene_kegg_mapping) <- c('gene', 'category')
colnames(kegg_pathways)     <- c('category', 'name', 'class', 'description')

```

```
kegg_pathways <- unique(kegg_pathways)
```

GO Enrichment

```
document: manuscript-shared-rnaseq/Rmd/results/go_enrichment_network.Rmd

# save required variables obtained earlier in analysis with variable names of the code below
module_colors <- module.colours
gene_tree <- gene.tree
wgcna_input <- logcpm.norm.counts.combat
modules_of_interest <- interesting.modules
num_modules <- length(unique(module_colors))

# save the module sizes
# Data frame of module sizes
module_counts <- c()
for (color in unique(module_colors)) {
    module_counts <- append(module_counts, sum(module_colors == color))
}

# create a mapping from module id to number of genes for later use
module_sizes <- data.frame(module_id=unique(module_colors),
                             num_genes=module_counts)

# Initialize parallelization
cl <- makeCluster(max(1, min(10, detectCores() - 2, na.rm = TRUE)))
registerDoParallel(cl)
message("Performing GO enrichment")

# Check each module for enrichment in GO terms and save result in a list
module_go_enrichment <- foreach(color=unique(module_colors), .packages=c('goseq')) %dopar% {
    set.seed(1)
    # Measure GO enrichment for module
    enriched <- tryCatch({
        # module gene ids
        in_module_geneids <- gene_ids[module_colors == color]
        message(sprintf("[GO enrichment] %s", color))

        # T. brucei GO enrichment
        enriched <- test_gene_enrichment(in_module_geneids, gene_ids,
                                         gene_go_mapping, gene_lengths)

        # Add descriptions
        enriched <- merge(enriched, go_term_id_mapping, by='category')
    }, error=function(e) {
        # goseq fails in some cases; have not been able to track down cause yet
        # to avoid errors we will just return an empty result set
        warning(sprintf("GO enrichment failed for module %s", color))
        cbind(
            get_enrichment_placeholder(),
            term=numeric(0),
            ontology=numeric(0)
        )
    })
}
```

```

    })
enriched
}
names(module_go_enrichment) <- unique(module_colors)

# remove any null/empty entries from the results
module_go_enrichment <- module_go_enrichment[!sapply(module_go_enrichment, is.null)]

# unregister cpus
stopCluster(cl)

# Print enrichment results

# temporarily repeat the gene / go term mapping to add 'term' column
gene_go_mapping_tmp <- as.data.frame(unique(go_terms %>% select(GID, GO, TERM, ONTOLOGY)))
colnames(gene_go_mapping_tmp) <- c('gene', 'category', 'term', 'ontology')

gene_info_tmp <- gene_info %>% select(-chromosome, -strand,
                                         -type, -transcript_length)
colnames(gene_info_tmp) <- c("gene", "description")

tmp <- cbind(gene_info_tmp, color=module_colors)

#tmp <- cbind(gene=gene_ids, color=module_colors)
gene_mapping <- merge(gene_go_mapping_tmp, tmp, by='gene')
cat(sprintf('Total enriched modules: %d\n',
            sum(sapply(module_go_enrichment, nrow) > 0)))

# create tables of the results in this document
print_enrichment_results(module_go_enrichment, module_sizes, 'GO terms',
                        NULL, gene_mapping, output_dir='../results/printed_enrichment_results',
                        NULL, gene_mapping,
                        enrichment_type='go',
                        include_gene_lists=FALSE)

enriched_colors_go <- get_enriched_modules(module_go_enrichment)

# Module enrichment status (used in dendrogram plots)
go_enrichment_status <- as.numeric(module_colors %in% enriched_colors_go)

```

KEGG Enrichment

document: manuscript-shared-rnaseq/Rmd/results/kegg_enrichment_network.Rmd

```

# Check each module for enrichment in KEGG terms and save result in a list
cl <- makeCluster(max(1, min(10, detectCores() - 2, na.rm = TRUE)))
registerDoParallel(cl)

message("Performing KEGG enrichment")

# Check each module for enrichment in GO terms and save result in a list
module_kegg_enrichment <- foreach(color=unique(module_colors), .packages=c('goseq')) %dopar% {
  set.seed(1)

```

```

# Measure KEGG enrichment for module
enriched <- tryCatch({
  in_module_geneids <- gene_ids[module_colors == color]
  enriched <- test_gene_enrichment(in_module_geneids, gene_ids,
                                    gene_kegg_mapping, gene_lengths)

  enriched <- unique(merge(enriched, kegg_pathways[,c('category','name')], 
                           by='category'))
}, error=function(e) {
  # goseq fails in some cases; have not been able to track down cause yet
  warning(sprintf("KEGG enrichment failed for module %s", color))
  return(get_enrichment_placeholder())
})
enriched
}

names(module_kegg_enrichment) <- unique(module_colors)

# remove any null/empty entries from the results
module_kegg_enrichment <- module_kegg_enrichment[!sapply(module_kegg_enrichment, is.null)]

# unregister cpus
stopCluster(cl)

cat(sprintf('- Total enriched modules: %d\n',
            sum(sapply(module_kegg_enrichment, nrow) > 0)))

# create tables of the results in this document
print_enrichment_results(module_kegg_enrichment, module_sizes,
                         'KEGG pathway',
                         #output_dir='../results/printed_enrichment_results',
                         enrichment_type='kegg')

enriched_colors_kegg <- get_enriched_modules(module_kegg_enrichment)

# Module enrichment status (used in dendrogram plots)
kegg_enrichment_status <- as.numeric(module_colors %in% enriched_colors_kegg)

cat('\n### Dendrogram with annotated modules\n')

unassigned_modules <- as.numeric(module_colors == 'grey')

png(filename = "../figures/gene_tree_and_module_enrichment_status.png", res =1200,
    type = "cairo", units = 'in', width = 6, height = 6, pointsize = 10)
WGCNA::plotDendroAndColors(gene_tree,
                            cbind(module_colors, go_enrichment_status,
                                  kegg_enrichment_status, unassigned_modules),
                            groupLabels=c(sprintf("Modules\n(n=%s)", num_modules),
                                         #sprintf("Red = upregulated at %s", CONFIG$de_cond2),
                                         "GO enrichment", "KEGG enrichment", "Unassigned"),
                            #cex.colorLabels=cex_color_labels, cex.main=cex_main,
                            # cex.axis=cex_axis, cex.lab=cex_lab,
                            dendroLabels=FALSE,
                            #marAll=c(4,8,6,4),

```

```
guideHang=0.05)
dev.off()
```

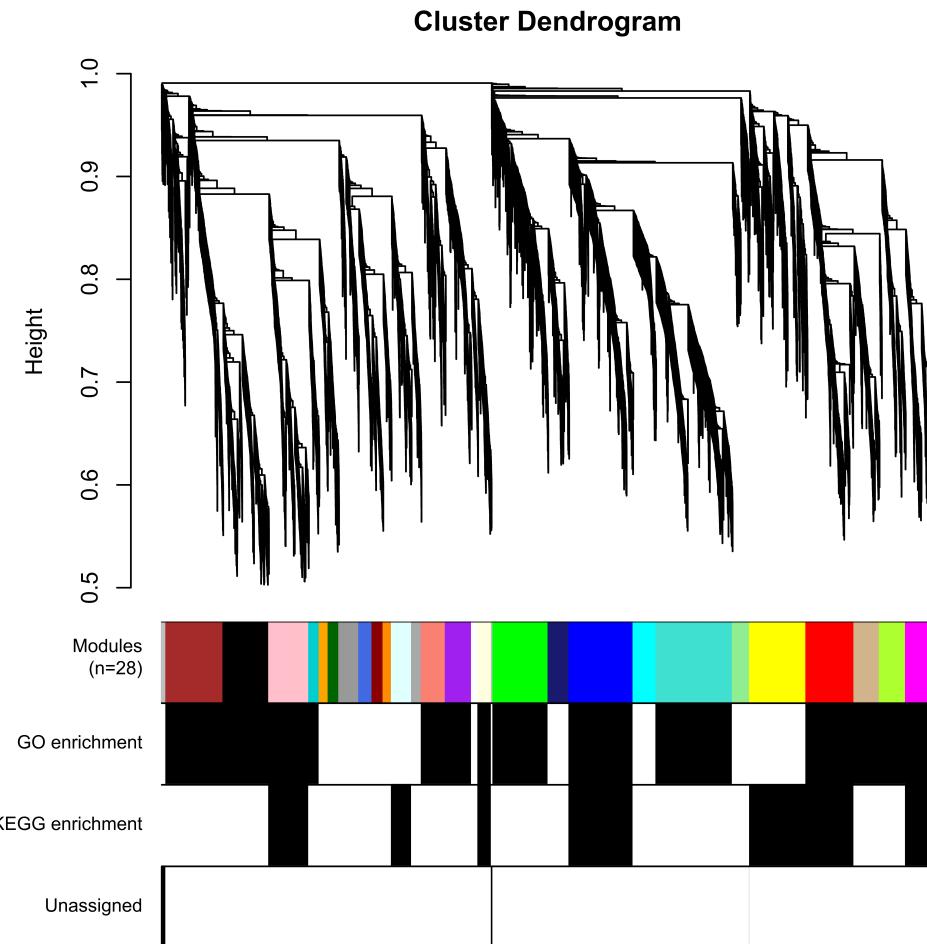


Figure 15: Gene tree and module enrichment status

Module labels mappings

```
module_label_mapping <- rbind(cbind(modules_of_interest,
                                      seq_along(modules_of_interest)),
                                cbind(remaining_modules, #remaining modules variable?
                                      seq_along(remaining_modules) +
                                      length(modules_of_interest)))

module_label_mapping <- as.data.frame(module_label_mapping)
colnames(module_label_mapping) <- c('color', 'number')
```

Save results

FIRE (Finding Informative Regulatory Elements)

Module genes and their module labels are required as input by FIRE (Finding Informative Regulatory Elements). Hence we write out text files with this input.

```
# write out cluster/module genes and their corresponding module labels for use by
# FIRE (Finding Informative Regulatory Elements)

fire.clusters <- data.frame(modGenes, module.labels[inModules])

# sort by module labels; FIRE input should start from 0 in module labels column.
fire.clusters <- fire.clusters[order(fire.clusters$module.labels.inModules.), c(1,2)]

write.table(as.matrix(fire.clusters), file = "../results/tbrucei_FIRE_expression_clusters.txt",
            quote = FALSE, row.names = FALSE, sep = "\t")

# Also, write out module genes in a text file.
write.table(data.frame(modGenes), file = "../results/tbrucei_module_genes.txt", row.names = FALSE, quote = FALSE)
```

Prepare input sequences for FIRE by obtaining module genes sequences from *T. brucei*'s entire annotated transcripts. Use seqtk package.

```
# remove unrequired info from header and retain gene id.
cut -f2 -d "|" ../data/tbrucei_annotated_transcripts/TriTrypDB-43_TbruceiTREU927_AnnotatedTranscripts.fasta

# add ">" symbol to header
awk '{ if ($0 ~ /Tb/) { printf ">"; } print $0; }' ../results/tbrucei_annotated_transcript_sequences.fasta > \
../results/tbrucei_module_genes > \
../results/tbrucei_module_genes_sequences_FIRE_input.fasta

seqkit grep -n -f ../results/tbrucei_module_genes_sorted.txt ../results/tbrucei_annotated_transcript_sequences.fasta > \
../results/tbrucei_module_genes_sequences.fasta

# FIRE commands - FIRE should be run in the directory where all scripts were installed. Therefore,
# the input files should be copied to FIRE's directory (FIRE-x.x/)
#
#check whether input files are OK
perl TOOLS/FIRE_analyse_input_files.pl \
-fastafile tbrucei_module_genes_sequences_without_delimiter.fasta \
-expfile tbrucei_FIRE_expression_clusters.txt

# remove sequences >= 10,000 base pairs as they make FIRE crash
# code - https://www.biostars.org/p/62678/
cat ../results/tbrucei_module_genes_sequences_without_delimiter.fasta | \
awk '{y= i++ % 2 ; L[y]="$0; if(y==1 && length(L[1])<=9999) {printf("%s\n%s\n",L[0],L[1]);}}' > \
../results/tbrucei_module_genes_sequences_without_delimiter_filtered.fasta

# recheck input files
perl TOOLS/FIRE_analyse_input_files.pl \
-fastafile tbrucei_module_genes_sequences_without_delimiter_filtered.fasta \
-expfile tbrucei_FIRE_expression_clusters.txt
```

```
# output
#Checking the expression file
#Expression file is OK.
#
#Checking the fasta file
#Fasta file is OK (min sequence length = 98, max length = 9917).
#
#Found fasta sequence for 7272 / 7390 identifiers in expression file.

# Analyze sequences for motifs
perl fire.pl --expfiles=tbrucei_FIRE_expression_clusters.txt --exptype=discrete \
--fastafile_dna=tbrucei_module_genes_sequences_without_delimiter_filtered.fasta --nodups=1

# create HTML output of the results
perl MORESCRIPTS/makeresultindex.pl tbrucei_FIRE_expression_clusters.txt "T. brucei cluster motifs"
```