Motif analysis methods

Warren Anderson, Mete Civelek, Michael Guertin

September 18, 2019

1 Overview

Here we describe the basic methods for identifying transcription factor motifs in early adipogenesis. The methods for alignment and initial processing of ATAC-seq and PRO-sec data can be found here:

```
https://github.com/WarrenDavidAnderson/genomeAnalysis/tree/master/ATACseq
https://github.com/WarrenDavidAnderson/genomeAnalysis/tree/master/PROseq
```

2 ATAC-seq peak calling with `MACS2`

For peak calling, we used a SLURM system to run the analysis over a range of 54 parameter variations. In particular, we varied the FDR and fold change parameters as follows for both pre-adipogenesis times and 6d replicates. For peak calling at the preadipogenesis timepoints, we aggregated all .bam files for all preadipogenesis timepoint replicates. For peak calling at the mature adipocyte timepoint, we aggregated the 6 day replicate .bam files. We called open chromatin peaks using `MACS2` . We varied the FDR threshold for defining peak significance (-q flag set to 0.1, 0.05, and 0.01). We also varied the range of fold changes above background over which peaks are identified (-m flag set to 1, 3, 5, 10, and 20 for the lower limit and 10, 50, 100, and 200 for the upper limit). Here was also describe our methods for empirically defining ATAC peak summits. The code for these analyses can be found in *call_4h_peaks.slurm*, *call_6d_peaks.slurm*, *filter_sort_peaks.sh*, *integrateReads.sh*, and *empiricalSummits.R*.

Here we generate the parameters values for the4 analyses.

```
##############################
## generate params in R
##############################

module load gcc/7.1.0
module load R/3.5.1
library(dplyr)
fdr = c(0.1, 0.05, 0.01)
m1 = c(1,3,5,10,20)
m2 = c(10,50,100,200)
vars = expand.grid(fdr,m1,m2)
names(vars) = c("fdr","m1","m2")
vars = vars %>% filter(m2 > m1)
for(ii in 1:nrow(vars)){
fname = paste0("vars_",ii,".txt")
out = vars[ii,]
write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\")
}
```

The SLURM submission included two codes, the first of which is the central submission of the job to the server for parallel implementation. This code is identical for pre-adipocytes and terminal adipocytes. For the pre-adipogenesis analysis, the code is run from a directory containing the associated .bam files (e.g., 3T3_t0_rep1_sample_atac.bam, ..., 3T3_4hr_rep3_sample_atac.bam). For the 6d timepoint, the code is run from a directory with the 6d replicate .bam files (e.g., 3T3_6d_rep1_sample_atac.bam).

```
#############################
## main array script, main_macs2.sh
#############################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

# load macs2
module load macs2

dir=/nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/preadip_bams
cd $dir

# sbatch --array=1-54 main_macs2.sh
cnt=$SLURM_ARRAY_TASK_ID
$dir/macs2_run.sh $cnt
```

The analysis scripts (macs2_run.sh) differ slightly for the pre- and post-adipogenesis conditions. Here is the script for the pre-adipogenesis time points.

```
##############################################################
## macs2 run script, macs2_run.sh
##############################################################

#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=/nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/preadip_bams
cd ${dir}

# subdirectory for specific analyses
mkdir params${cnt}
cp 3T3*.bam* params${cnt}
cp vars_${cnt}.txt params${cnt}
cd params${cnt}

# isolate all motif identifiers
params=$(cat vars_${cnt}.txt)

# load macs2
module load macs2

# params
files=$(ls 3T3*.bam*)
name=3T3_atac
species=mm
fdr=$(echo ${params} | awk -F" " '{print $1}')
m1=$(echo ${params} | awk -F" " '{print $2}')
m2=$(echo ${params} | awk -F" " '{print $3}')
ndups=50
output=atac6d_$fdr_$m1_$m2
```

```
# initialize log file
exec &> log_${fdr}_${m1}_${m2}_3t3atac.txt
echo ${name}
echo fdr ${fdr}
echo m1 ${m1}
echo m2 ${m2}
echo ""

# run for all 3T3 files
macs2 callpeak -t ${files} -f BAMPE -n ${name} --outdir ${output} \
-g ${species} -B --call-summits --keep-dup ${ndups} -q ${fdr} -m ${m1} ${m2}

rm *.bam
```

Here is the script for the 6d time point.

```
################################################################
## macs2 run script, macs2_run.sh
################################################################

#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=/nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/6d_bams
cd ${dir}

# subdirectory for specific analyses
mkdir params${cnt}
cp *.bam params${cnt}
cp vars_${cnt}.txt params${cnt}
cd params${cnt}

# isolate all motif identifiers
params=$(cat vars_${cnt}.txt)

# load macs2
module load macs2

# params
files=$(ls *.bam*)
name=3T3_atac
species=mm
fdr=$(echo ${params} | awk -F" " '{print $1}')
m1=$(echo ${params} | awk -F" " '{print $2}')
m2=$(echo ${params} | awk -F" " '{print $3}')
ndups=50
output=atac6d_${fdr}_${m1}_${m2}

# initialize log file
exec &> log_${fdr}_${m1}_${m2}_3t3atac.txt
echo ${name}
echo fdr ${fdr}
echo m1 ${m1}
echo m2 ${m2}
echo ""
```

```
# run for all 3T3 files
macs2 callpeak -t ${files} -f BAMPE -n ${name} --outdir ${output} \
-g ${species} -B --call-summits --keep-dup ${ndups} -q ${fdr} -m ${m1} ${m2}

rm *.bam
```

We use the following shell code and R code to determine the number of peaks identified for each parameter combination. This unix code will loop through the results folders and determine the number of lines/peaks in each .narrowPeak file.

```
# get peak counts for each param set
# nohup sh getNpeaks.sh &
folds=$(ls -d *params*)
exec &> Npeaks.txt
for ii in ${folds}
do
cd ${dir}
cd ${ii}
d=$(ls -d */)
cd ${d}
echo ${d}
wc -l *.narrowPeak
echo " "
done
```

We then format the results using R as follows.

```
# R code to parse the results

# folder - param mapping
module load gcc/7.1.0  openmpi/2.1.5 R/3.6.0
library(dplyr)
fdr = c(0.1, 0.05, 0.01)
m1 = c(1,3,5,10,20)
m2 = c(10,50,100,200)
vars = expand.grid(fdr,m1,m2)
names(vars) = c("fdr","m1","m2")
vars = vars %>% filter(m2 > m1)
vars = vars %>% mutate(fold=paste0("params",1:nrow(vars)))
vars = vars %>% mutate(cond=paste0(vars$fdr,"_",vars$m1,"_",vars$m2))

# parse the results
library(dplyr)
dat0 = read.table("Npeaks.txt",header=F,stringsAsFactors=F,fill=T)
ind.param = grep("atac6d_",dat0[,1])
ind.peaks = grep("3T3_atac_peaks",dat0[,2])
out = matrix(NA,length(ind.param),4)
for(ii in 1:nrow(out)){
p = strsplit(dat0[ind.param[ii],1],"_")[[1]]
c = strsplit(dat0[ind.peaks[ii],1],"_")[[1]]
out[ii,1:2] = p[2:3]
out[ii,3] = strsplit(p[4],"/")[[1]][1]
out[ii,4] = c
}
out = apply(out,2,function(x){data.matrix(x) %>% as.numeric})
res = data.frame(fdr=out[,1],m1=out[,2],m2=out[,3],pkcnt=out[,4],
     stringsAsFactors=F)
min(res$pkcnt)
```

```
max(res$pkcnt)

# aggregate information, output
pkdat = merge(res, vars, by=c("fdr","m1","m2"))
fname = "peaks6d_paramvar.txt"
write.table(pkdat,fname,col.names=T,row.names=F,sep="\t",quote=F)
```

The codes for implementing the above analyses can be found in call_4h_peaks.slurm and call_6d_peaks.slurm. Next we merge any overlapping peaks and remove documented 'blacklist' regions them from the resulting peak sets using `Bedtools` (see *filter_sort_peaks.sh*).

```
# get blacklisted regions
wget http://mitra.stanford.edu/kundaje/akundaje/release/blacklists/
mm10-mouse/mm10.blacklist.bed.gz
gunzip mm10.blacklist.bed.gz

# bedtools
PATH=$PATH:/media/wa3j/Seagate2/Documents/software/bedtools2/bin/

# document summit coordinate
awk '{OFS="\t";} {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$2+$10}' \
3T3_atac_peaks.narrowPeak > allpeaks.txt

# sort and filter, pre-adipogenesis
sort -k1,1 -k2,2n allpeaks.txt | \
bedtools merge -i stdin -c 10,11 -o collapse | \
bedtools subtract -a stdin -b mm10.blacklist.bed | \
awk ' $2 >= 0 ' > 3T3_atac_4hpeaks_sorted.bed


# sort and filter, adipocyte data
awk '{OFS="\t";} {print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$2+$10}' | \
3T3_atac_peaks.narrowPeak > allpeaks.txt
sort -k1,1 -k2,2n allpeaks.txt | \
bedtools merge -i stdin -c 10,11 -o collapse | \
bedtools subtract -a stdin -b mm10.blacklist.bed | \
awk ' $2 >= 0 ' > 3T3_atac_6dpeaks_sorted.bed
```

We empirically define ATAC peak summits as follows. First, we reformat the ATAC data by integrating over entire reads (see *integrateReads.sh*). First we integrate over reads combined across all pre-adipogenesis time points.

```
#####################################################
## integrated reads for all preadip time points
## used in empiricalSummits.R to get summits
#####################################################

cd /nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/preadip_bams

# morege all preadipogenesis bams
module load samtools
samtools merge preadipMerged.bam *.bam

# isolate only concordant alignments
mv preadipMerged.bam pre.bam
samtools view -b -f 0x2 pre.bam -o preadipMerged.bam
samtools sort -n preadipMerged.bam -o sorted.bam
samtools fixmate sorted.bam fixed.bam
```

```
# produce a bed file
module load bedtools
bedtools bamtobed -i fixed.bam -bedpe > preadip0.bed
rm fixed.bam sorted.bam

# filter and sort the bed file
awk '{OFS="\t";} {print $1,$2,$6,$7,$8,$9}' preadip0.bed > bed
sort -k 1,1 bed > sorted.bed
rm bed preadip0.bed

# from bed to bedgraph
bedtools genomecov -bg -trackline -trackopts name=preadip -i sorted.bed | \
 -g mm10.chrom.sizes > preadip.bedGraph

# generate the bigwig
module load ucsc-tools
bedGraphToBigWig preadip.bedGraph mm10.chrom.sizes preadip.bigWig
```

Next we integrate over reads for individual time points.

```
####################################################
## integrated reads for all preadip time points
## individual bigwigs
## used in fimo peak enrichment analyses
####################################################

# set id files for the array job (R)
cnt=1
for ii in *.bam*
do
echo ${ii} > array${cnt}
cnt=$(expr ${cnt} + 1)
done

####### org_main.sh #######
#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=4:00:00
#SBATCH --partition=standard

dir=~/preadip_bams/integrateIndiv
cd ${dir}

# sbatch --array=1-21 org_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/org_run.sh ${cnt}

##########################
####### org_run.sh #######
#!/bin/bash

# chmod u+x *.sh

dir=~/preadip_bams/integrateIndiv
cd ${dir}

module load bedtools
module load samtools
module load ucsc-tools

cnt="$1"

# data
dat=$(cat array${cnt})
condit=$(echo ${dat} | awk -F"_sample_atac.bam" '{print $1}')

# isolate only concordant alignments
samtools view -b -f 0x2 ${dat} -o ${condit}.bam
samtools sort -n ${condit}.bam -o sorted_${condit}.bam
samtools fixmate sorted_${condit}.bam fixed_${condit}.bam

# produce a bed file
bedtools bamtobed -i fixed_${condit}.bam -bedpe > ${condit}.bed
rm ${condit}.bam sorted_${condit}.bam fixed_${condit}.bam

# filter and sort the bed file
awk '{OFS="\t";} {print $1,$2,$6,$7,$8,$9}' ${condit}.bed |  \
sort -k 1,1 > sorted_${condit}.bed
rm ${condit}.bed

# from bed to bedgraph
bedtools genomecov -bg -trackline | \
-trackopts name=preadip -i sorted_${condit}.bed | \
 -g mm10.chrom.sizes > ${condit}.bedGraph

# generate the bigwig
bedGraphToBigWig ${condit}.bedGraph mm10.chrom.sizes ${condit}.bigWig
rm sorted_${condit}.bed
```

We empirically define peaks using the reads integrated over all pre-adipogenesis time points by sliding a 50 bp window across each peak in 5 bp increments. We selected the summit of a given peak as the center of the 50 bp window with the maximal number of reads (*empiricalSummits.R*). First we load data from previous analyses.

```
############################################################
## key analysis parameters
############################################################

window = 50 # window = sliding window size (bp)
bin = 5 # bin = increment by which the sliding window will move (bp)

############################################################
## import macs2 peak data and ATAC bigWigs
############################################################

# peak coordinates
# https://github.com/taoliu/MACS
fname = "3T3_atac_4hpeaks_sorted.bed"
bed0 = read.table(fname,stringsAsFactors=F,header=F)
names(bed0) = c("chr","start","end","rel","summit")
all.peaks = paste0(bed0[,1],":",bed0[,2],"-",bed0[,3])

# aggregated/integrated preadipogenic ATAC bigWig
bw.file = paste0("/media/wa3j/Seagate2/Documents/PRO/",
                 "adipogenesis/July2018/integratedBW/preadip.bigWig")
```

```
loaded.bw = load.bigWig(bw.file)
```

Here are the functions for this analysis.

```
#############################################################
# function to get coordinates for all peaks
#############################################################

# function to find peak info from rownames of the results frame
get.map.from.res = function(res){
  namen = res
  chrs = sapply(namen,function(x){strsplit(x,":")[[1]][1]})
  ends = sapply(namen,function(x){strsplit(x,"-")[[1]][2]})
  strs = sapply(namen,function(x){y=strsplit(x,"-")[[1]][1];
  return(strsplit(y,":")[[1]][2]) })
  coords = cbind(chrs,strs,ends) %>% as.data.frame(stringsAsFactors=F)
  return( coords )
} # get.map.from.res

coords0 = get.map.from.res(all.peaks)
coords0[,2:3] = apply(coords0[,2:3],2,function(x){
     data.matrix(x) %>% as.numeric})

#############################################################
#  functions to loop through peaks and empirically find summits
#############################################################

# function to get coordinates for mapping reads to identify a peak window
# input: coords = coordinates in a single line with bed format
# input: win = window size (bp)
# input: del = delta for sliding the window over the coordinates (bp)
# output: mat = dataframe with bed coordinates
get.coords = function(coords=NULL, win=NULL, del=NULL){
  chr = coords[1] %>% as.character
  range = (coords[3] - coords[2]) %>% data.matrix %>% as.numeric
  n.win = ceiling( (range-win)/del )
  len = n.win * del + win
  diff = len - range
  addL = floor(diff/2)
  addR = ceiling(diff/2)
  str = (coords[2] - addL) %>% data.matrix %>% as.numeric
  end = (coords[3] + addR) %>% data.matrix %>% as.numeric
  mat = c()
  for(ii in 1:(n.win+1)){
    st = str + (ii-1) * del
    ed = st + win
    mat = rbind(mat, c(chr, st, ed))
  }
  mat = as.data.frame(mat,stringsAsFactors=FALSE)
  names(mat) = c("chr","start","end")
  mat[,2:3] = apply(mat[,2:3],2,function(x){data.matrix(x)%>%as.numeric})
  return(mat)
} # get.coords

# function to empirically find summits for a set of peaks
# we slide a window along the peak and identify the center
# of the window with the highest reads - that is the summit
# input: peaks = bed file with ATAC peaks
# input: window = sliding window size (bp)
# input: bin = increment by which the sliding window will move (bp)
# input: bw.file = bigWig file with path included
# output = bed with peaks and summits
find.summits = function(peaks=NULL, window=NULL, bin=NULL, bw.file=NULL){

  # loop through all peaks
  summits = rep(0,nrow(peaks))
  for(ii in 1:nrow(peaks)){

    # set the summit to the middle for peak < window
    dpeak = peaks[ii,3] - peaks[ii,2]
    if(dpeak < window){
      summits[ii] = floor(peaks[ii,2] + dpeak/2)
      next
    }

    # identify the summit as the center of the max peak
    windows = get.coords(coords=peaks[ii,], win=window, del=bin)
    counts = bed.region.bpQuery.bigWig(bw.file, windows)
    ind = which(counts == max(counts))[1]
    summits[ii] = floor(windows[ind,2] + window/2)

    # progress tracking
    if((100*ii/nrow(peaks)) %% 1 == 0){
      Sys.sleep(0.5)
      print(paste0(100*ii/nrow(peaks),"% completed"))
      flush.console()
    }

  } ## ii, peak loop

  # return bed with peaks and summits
  return(cbind(peaks,summits))

} # find.summits
```

We run the analysis and save the results as follows.

```
############################################################
#  get all summits empirically
############################################################

summits.bed = find.summits(peaks=bed0[,1:3], window=window,
     bin=bin, bw.file=loaded.bw)
save(summits.bed, file="ATACsummits_20190914.RData")
```

3 Regulatory element identification

Here we identify regulatory elements from PRO-seq data using `dREG` (`https://dreg.dnasequence.org/`) (Wang *et al.*, 2019). We aggregate the PRO data across all preadipogenesis time points, up to 4 hrs post treatment. We also convert all minus strand read counts to negative values. These processing steps are completed in the command line using the `Samtools` and `UCSC tools` (Li *et al.*, 2009; Kent *et al.*, 2010). The plus and minus strand bigWigs are then uploaded to the `dREG` web interface and the complete data can be downloaded when then analysis is finished. BedGraph files for visualization with the `UCSC genome browser` (Kent *et al.*, 2002) are prepared as well. The code for processing the aligned .bam files for `dREG` can be found in *merge_bigwig_dreg.sh*.

```
# data directories
bigwigdir=/bigWig/data/directory
dir=/output/directory
cd $dir

# strand separate primary aligned bams
# process plus and minus aligned reads separately
for i in *.bam*
do
name=$(echo $i | awk -F".bam" 'print $1')
cat > proScript$name.sh <<EOF
#!/bin/bash
samtools view -bh -F 20 $name.bam > $name_pro_plus.bam
samtools view -bh -f 0x10 $name.bam > $name_pro_minus.bam
EOF
# call a script for each core
echo calling proScript$name.sh
chmod 700 proScript$name.sh
nohup ./proScript$name.sh &
done

# aggregate bigwigs across all preadipogenesis time points
minusfiles=$(ls $bigwigdir/*minus.bigWig)
plusfiles=$(ls $bigwigdir/*plus.bigWig)
bigWigMerge $minusfiles merged_pro_preadip_minus.bg
bigWigMerge $plusfiles merged_pro_preadip_plus.bg

# multiple minus values by -1
awk ' print $1, $2, $3, $4*(-1) ' merged_pro_preadip_minus.bg \
> adipogen_minus_scaled.bg
cat merged_pro_preadip_plus.bg > adipogen_plus_scaled.bg

# generate standard bedgraphs with headers
touch minus.txt
touch plus.txt
echo "track type=bedGraph name=adipogen_minus_combined \
color=0,255,0 altColor=0,255,0 alwaysZero=on visibility=full" >> minus.txt
echo "track type=bedGraph name=adipogen_plus_combined \
color=255,0,0 altColor=255,0,0 alwaysZero=on visibility=full" >> plus.txt
cat minus.txt adipogen_minus_scaled.bg > adipogen_minus_combined.bedGraph
cat plus.txt adipogen_plus_scaled.bg > adipogen_plus_combined.bedGraph
rm minus.txt plus.txt *merged_pro* *_scaled.bg

# get chromosome sizes
wget http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes

# convert back to bigwig
mm10=mm10.chrom.sizes
bedGraphToBigWig adipogen_minus_combined.bedGraph $mm10 \
adipogen_minus_combined.bigWig
bedGraphToBigWig adipogen_plus_combined.bedGraph $mm10 \
adipogen_plus_combined.bigWig

# run dreg
https://dreg.dnasequence.org

# unpack results
tar xvzf out.tar.gz
```

```
gunzip out.dREG.peak.full.bed.gz
```

4 Transcription factor motif identification

Here we describe the methods for identifying DNA motifs using `MEME` (). The analyses are applied to data based on differential expression analysis of ATAC-seq data, timeseries clustering of ATAC-seq data, and differential expression analysis of regulatory emelents identified from PRO-seq data using `dREG` (). The methods for generating sets of 100 bp regions for motifs analysis can be found here: . The general analysis workflow is as follows. For each of these three regulatory region classes, we implement the motif analysis a number of different ways, as described below. After identifying motifs, we aggregate all motifs and then we use `TOMTOM` to determine which indentified motifs match the position weight matrices of known transcription factors. After identifying all transcription factors matched to the identified motifs, we match these TFs to all databaseTFs. Based on this analysis, we cluster the transcription factors and isolate exemplary factors from each cluster. We map these exemplary factors to the murine genome using `FIMO`. Finally, we evaluate factor binding enrichment in differentially expressed and dynamic ATAC peaks, along with regulatory elements (the latter 2 should be added to the analysis).

We perform *de novo* motif analysis using `MEME` based on dynamic cluster ATAC peaks, differentially expresed ATAC peaks, and differentially expressed dREG peaks. For the differential ATAC peaks, we implement *de novo* motif analysis with and without a Markov background model that specifies k-mer frequencies (i.e., order 0 accounts for single bases, order 3 accounts for 3-mers, 2-mers, and single bases). We use a third order background for either increased or decreased peaks. We use fasta-get-markov from the MEME-suite to generate background models and we use fastaFromBed from `Bedtools` to generate fasta files from the BED coordinates generated in R as described elsewhere . We compared increased peaks to both decreased and unchanged peaks, both with and without 3rd order background models. Similarly, we compared decreased peaks to both increased and unchanged peaks with and without backgrounds for each pairwise time point ATAC peak comparison. All `MEME` analyses are completed with an E-value threshold of 10^{-1}.

4.1 Differential ATAC peak data preparation for *de novo* motif analysis

We first perform sifferential expression analysis of pre-adipogenesis ATAC peaks for *de novo* motif analysis. Our goal was to identify potential functional transcription factor binding motifs within open chromatin regions that change during early adipogenesis. toward this end, we implemented all pairwise comparisons of ATAC peak levels between preadipogenic time points (i.e., 0hr vs 20min, 0hr vs 40min, ..., 3hr vs 4hr). The main code can be found in *atac_time_deg_meme.R*. The following function is used to upload the *bigWig* data (see *atac_norm_functions.R*).

```
# function to import atacseq read data maped to specific coordinates
# input: bed = bed file for mapping reads
# input: bigWig.path = path to bigWig files
# input: file.prefix = file prefix
# input: file.suffix = file suffix
# input: min.length = min gene length
# output: data frame with reads mapped for each condition
get.counts.interval <- function(bed=NULL, bigWig.path=NULL,
                                file.prefix=NULL,
                                file.suffix=NULL,
                                min.length=NULL) {

  # filter bed based on min gene length
  len = bed$end - bed$start
  indrem = which(len < min.length)
  if(length(indrem)>0){bed = bed[-indrem,]}
  # output files
  vec.names = c()
  output = data.frame(matrix(ncol = 0, nrow = nrow(bed)))

  # loop through each condition and load reads
  for(bigWig in Sys.glob(file.path(bigWig.path,
      paste0(file.prefix, paste0("*",file.suffix))))) {
    factor.name = strsplit(bigWig, "/")[[1]]
    factor.name = strsplit(factor.name[length(factor.name)],
                           file.suffix)[[1]][1]
    factor.name = strsplit(factor.name,file.prefix)[[1]][2]
    print(factor.name)
    vec.names = c(vec.names, factor.name)
    loaded.bw = load.bigWig(bigWig)
    reads = bed.region.bpQuery.bigWig(loaded.bw, bed)
    output = cbind(output, reads)
  } # bigWig

  # annotation and output
  colnames(output) = vec.names
  r.names = paste0(bed[,1],':',bed[,2],'-',bed[,3])
  row.names(output) = r.names
  return(output)
} # get.counts.interval
```

Initially, we load libraries, data, key parameters, and the data import function. The necessary input for this analysis is derived from *empiricalSummits.R*.

```
library(dplyr)
library(DESeq2)
library(ggplot2)
library(bigWig)
library(prodlim)

source("atac_norm_functions.R")

############################################################
## key analysis parameters
############################################################

# set number of base pairs around peaks for motif analysis
pk.dist = 50

# parameters for designating dynamic peaks
sig.thresh = 0.001
fc.thresh = 1

# parameters for designating non-dynamic peaks
sig.un = 0.5
fc.un = 0.25

############################################################
## import macs2 peak data
############################################################

# peak and summit coordinates (summits.bed, from empiricalSummits.R)
load("ATACsummits_20190914.RData")
bed0 = summits.bed
```

```
# chrom sizes
chrm.size = read.table("mm10.chrom.sizes",header=F,stringsAsFactors=F,
     sep="\t")
chrm.size = cbind(chrm.size[,1], 1, chrm.size[,2])
chrm.size = as.data.frame(chrm.size, stringsAsFactors=F)
names(chrm.size) = c("chr","start","end")
chrm.size[,2:3] = apply(chrm.size[,2:3],2,function(x){
     data.matrix(x) %>% as.numeric})
```

We take a brief look at the basic characteristics of the peak data. Here we filter the peak data for autosomes (chromosomes 1-19).

```
############################################################
## basic characteristics of MACS2 output
############################################################

# filter chromosomes
unique(bed0$chr)
dim(bed0) # 86345
chr.keep = paste0("chr",c(1:19))
bed0 = bed0[bed0$chr %in% chr.keep,]
dim(bed0) # 83716

# look at peak distances
d = bed0$end - bed0$start
median(d)
quantile(d)

# save coords
save(bed0, file="bed.map20190827.RData")
```

Next we perform some data normalization and formatting before implementing pairwise time-point differential expression analysis. We normalize the data based on sequenceing depth and remove the 6 day adipocyte data.

```
############################################################
## import atac-seq data and set directory for data output
############################################################

# atac data mapped to peak regions
bigWig.path = paste0("/media/wa3j/Seagate2/Documents/PRO/adipogenesis",
                     "/July2018/atac_bw_20181127/atac_all")
file.prefix = "3T3_"
file.suffix = ".bigWig"
min.length = 1
reads0 = get.counts.interval(bed=bed.map[,1:3], bigWig.path=bigWig.path,
                             file.prefix=file.prefix,
                             file.suffix=file.suffix,
                             min.length=min.length)



############################################################
## normalize raw counts to size factors, generate DESeq object
############################################################

# estimate size factors
size_factors = estimateSizeFactorsForMatrix(reads0)
```

```
## generate DESeqDataSet object and transform count data to log2 scale
## note. enter size factors before implementing log transform
conditions = sapply(names(reads0),function(x){strsplit(x,"_")[[1]][1]})
colData = as.data.frame(conditions)
deseq_obj_sizefac = DESeqDataSetFromMatrix(countData=reads0,
                                           colData=colData,
                                           design=~conditions)
sizeFactors(deseq_obj_sizefac) = size_factors

# note that each column of the count matrix was divided
# by the respective size factor
head(sapply(c(1:24),function(x){reads0[,x]/size_factors[x]})[,1:5])
head(counts(deseq_obj_sizefac, normalized=TRUE)[,1:5])

# save.image("atac.RData")
# load("atac.RData")

############################################################
## adjust data organization
############################################################

# basic counts, size factors, and annotation
counts = counts(deseq_obj_sizefac)
sizefac = sizeFactors(deseq_obj_sizefac)
times = colData(deseq_obj_sizefac)$conditions
t.order = cbind(c("t0","20min","40min","60min","2hr","3hr","4hr","6d"),
                c(0,0.33,0.67,1,2,3,4,144)) %>%
  as.data.frame(stringsAsFactors=FALSE)
names(t.order) = c("condition","time")

# specify new conditions as times (factor)
conditions = sapply(as.character(times),function(x){
  t.order$time[which(t.order$condition==x)]})
conditions = factor(conditions, levels=c(0,0.33,0.67,1,2,3,4,144))

# set new deseq object
colData = as.data.frame(conditions)
deseq_obj = DESeqDataSetFromMatrix(countData=counts,
                                   colData=colData, design=~conditions)
sizeFactors(deseq_obj) = estimateSizeFactorsForMatrix(counts)


############################################################
## exclude 6day data
############################################################

# remove the 6day data
counts = counts(deseq_obj) %>% as.data.frame
counts_preadip = counts[,-grep("6d",names(counts))]
sizefac_preadip = sizeFactors(deseq_obj)[-grep("6d",names(counts))]
conditions_preadip =colData(deseq_obj)$conditions[-grep("6d",names(counts))]
condit.map = cbind(as.character(conditions_preadip),
  names(counts_preadip)) %>%  as.data.frame(stringsAsFactors=F)
names(condit.map) = c("time","rep")

# set new deseq object
colData_preadip = as.data.frame(conditions_preadip)
deseq_obj_preadip = DESeqDataSetFromMatrix(countData=counts_preadip,
                                           colData=colData_preadip,
                                           design=~conditions_preadip)
sizeFactors(deseq_obj_preadip) = sizefac_preadip
```

Then we implement all pairwise differential expression analyses for every combination of time points, up to 4 hrs. The analysis results are saved as *pairwise.deg.RData*.

```
############################################################
## generate all pairwise comparisons
############################################################

# loop through all pairwise combinations and run DESeq
condits = c(0,0.33,0.67,1,2,3,4) %>% as.character
res.pairs = list()
for(ii in 1:(length(condits)-1)){
  for(jj in (ii+1):length(condits)){

    print(paste0("compare ",condits[ii]," to ",condits[jj]))

    # basic data annotation
    ind_ii = which(conditions_preadip == condits[ii])
    ind_jj = which(conditions_preadip == condits[jj])

    # set new deseq object
    des = conditions_preadip[c(ind_ii,ind_jj)]
    colData_ij = as.data.frame(des)
    cnt_ij = counts_preadip[,c(ind_ii,ind_jj)]
    deseq_obj_preadip = DESeqDataSetFromMatrix(countData=cnt_ij,
                                           colData=colData_ij,
                                           design=~des)
    sizeFactors(deseq_obj_preadip) = sizefac_preadip[c(ind_ii,ind_jj)]
    colData(deseq_obj_preadip)$condition = des

    # pairwise DEG analysis
    dds = DESeq(deseq_obj_preadip)
    res = results(dds)[order(results(dds)$padj),]
    res.pairs[[paste0(condits[ii],"_",condits[jj])]] = res

  } # jj
} # ii

# save(res.pairs, file="pairwise.deg.RData")
# load("pairwise.deg.RData")
```

We use the following functions for generating peak corrdinates that will be used in *de novo* motif identification. In particular, the function *pk.coords.for.meme* will generate 100 bp windows around each peak summit (*pk.dist* = 50).

```
############################################################
## generate output for meme
############################################################

# function to find peak info from rownames of the results frame
get.map.from.res = function(res,map){
  namen = rownames(res)
  chrs = sapply(namen,function(x){strsplit(x,":")[[1]][1]})
  ends = sapply(namen,function(x){strsplit(x,"-")[[1]][2]})
  strs = sapply(namen,function(x){y=strsplit(x,"-")[[1]][1];
        return(strsplit(y,":")[[1]][2]) })
  coords = cbind(chrs,strs,ends) %>% as.data.frame(stringsAsFactors=F)
  out = row.match(coords, map[,1:3])
  return( cbind(res, map[out,]) )
} # get.map.from.res

# set coordinates around each summit based on pk.dist
# generate output format for meme
pk.coords.for.meme = function(res=NULL, pk.dist=NULL){
  out = c()
  for(ii in 1:nrow(res)){
    pks = res$summits[ii]
    new = c(res$chr[ii], pks-pk.dist, pks+pk.dist, rownames(res)[ii],
              signif(res$log2FoldChange[ii],3), "+", signif(res$padj[ii],3) )
    out = rbind(out,new)
  } # ii
  out = out %>% as.data.frame(stringsAsFactors=FALSE)
  names(out) = c("chr","start","end","peak_name","log2fc","str","fdr")
  rownames(out) = c(1:nrow(out))
  return(out)
} # pk.coords.for.meme
```

Here we generate the data files that will be used for motif identification with `MEME`. For each pairwise comparison, we output three files with the corrdinates for increased, decreased, and unchanged peaks. For all comparisons, we compare the longer time point to the shorter time point such that an increase (log2 fold change > 0) indicates that the peak region exibits an increase in open chromatin with respect to time. For each pairwise comparison, significantly changed peaks are defined as those with FDR < 0.001 (*sig.thresh* = 0.001) and an absolute log2 fold change greater than 1 (*fc.thresh* = 1). Unchanged peaks are defined as those with FDR > 0.5 (*sig.un* = 0.5) and absolute log2 fold change less than 0.25 (*fc.un* = 0.25). For pairwise comparisons in which there are more significant peaks than insignificant peaks, we use a same number of insignificant peaks corresponding to the number of significant peaks. In such cases, we use the insignificant peaks with the highest FDRs.

```
# loop through all pairwise data and write output for meme
for(ii in 1:length(res.pairs)){

  # basic annotation
  comp_ii = names(res.pairs)[[ii]]
  res_ii = res.pairs[[ii]]
  indsig = which(res_ii$padj<sig.thresh &
     abs(res_ii$log2FoldChange)>fc.thresh)
  ind.up = which(res_ii$padj<sig.thresh &
     res_ii$log2FoldChange>fc.thresh)
  ind.dn = which(res_ii$padj<sig.thresh &
     res_ii$log2FoldChange<(-1)*fc.thresh)
  ind.un = which(res_ii$padj>sig.un &
     abs(res_ii$log2FoldChange)<fc.un)

  # if there are more insignificant peaks, use the number of sig peaks
  # select those with the highest FDRs
  if(length(indsig)==0){next}
  if(length(ind.un) > length(indsig)){
     ind.un = rev(ind.un)[1:length(indsig)]
  }

  # get macs2 data for sig and unsig peaks
  sig.pks.up = get.map.from.res(res=res_ii[ind.up,], map=bed0)
  sig.pks.dn = get.map.from.res(res=res_ii[ind.dn,], map=bed0)
  uns.pks = get.map.from.res(res=res_ii[ind.un,], map=bed0)
  out.sig.up = pk.coords.for.meme(res=sig.pks.up, pk.dist=pk.dist)
  out.sig.dn = pk.coords.for.meme(res=sig.pks.dn, pk.dist=pk.dist)
  out.uns = pk.coords.for.meme(res=uns.pks, pk.dist=pk.dist)

  # set directory and output data for meme analysis
  dir_ii = paste0("meme_",comp_ii)
  system(paste0("mkdir ",dir_ii))
  setwd(dir_ii)
  fname = paste0("upsig_",comp_ii,".bed")
  write.table(out.sig.up,fname,sep="\t",quote=FALSE,
     col.names=FALSE,row.names=FALSE)
  fname = paste0("downsig_",comp_ii,".bed")
  write.table(out.sig.dn,fname,sep="\t",quote=FALSE,
     col.names=FALSE,row.names=FALSE)
  fname = paste0("unsig_",comp_ii,".bed")
  write.table(out.uns,fname,sep="\t",quote=FALSE,
     col.names=FALSE,row.names=FALSE)
  setwd(dir)

} # ii
```

4.2 Dynamic ATAC peak data preparation for *de novo* motif analysis

Here we document an analysis in which we clustered the dynamic profiles associated with each ATAC peak. This analysis groups open chromatin peaks pased on similarities of their temporal trajectories with respect to adipogenesis time. For this analysis were consider only the pre-adipogenic time points, up to 4 hrs following the application of the adipogenic cocktail. Some of the code here is redundant with that for the differential expression analysis. We format the data in R prior to implementing the cluster analysis using the *Short Time-series Expression Miner* (STEM, `http://www.cs.cmu.edu/~jernst/stem/`) (Ernst and Bar-Joseph, 2006). The code for this analysis can be found in *atac_time_cluster.R*. Here are the data formatting steps.

```
library(dplyr)
library(DESeq2)
library(ggplot2)
library(bigWig)
library(prodlim)

source("atac_norm_functions.R")

############################################################
## key analysis parameters
############################################################

# set number of base pairs around peaks for motif analysis
pk.dist = 50

# parameters for designating dynamic peaks
sig.thresh = 0.001

# parameters for designating non-dynamic peaks
sig.un = 0.5

############################################################
## import macs2 peak data
############################################################
```

```
# load bed.map - filtered ATAC peak coords, see atac_time_deg_meme.R
load("bed.map20190827.RData")

############################################################
## import atac-seq data and set directory for data output
############################################################

# atac data mapped to peak regions
bigWig.path = paste0("/media/wa3j/Seagate2/Documents/PRO/adipogenesis",
                     "/July2018/atac_bw_20181127/atac_all")
file.prefix = "3T3_"
file.suffix = ".bigWig"
min.length = 1
reads0 = get.counts.interval(bed=bed.map[,1:3], bigWig.path=bigWig.path,
                             file.prefix=file.prefix,
                             file.suffix=file.suffix,
                             min.length=min.length)


############################################################
## normalize raw counts to size factors, generate DESeq object
############################################################

# estimate size factors
size_factors = estimateSizeFactorsForMatrix(reads0)

## generate DESeqDataSet object and transform count data to log2 scale
## note. enter size factors before implementing log transform
conditions = sapply(names(reads0),function(x){strsplit(x,"_")[[1]][1]})
colData = as.data.frame(conditions)
deseq_obj_sizefac = DESeqDataSetFromMatrix(countData=reads0,
                          colData=colData, design=~conditions)
sizeFactors(deseq_obj_sizefac) = size_factors


############################################################
## adjust data organization
## exclude 6day data
############################################################

# basic counts, size factors, and annotation
counts = counts(deseq_obj_sizefac)
sizefac = sizeFactors(deseq_obj_sizefac)
times = colData(deseq_obj_sizefac)$conditions
t.order = cbind(c("t0","20min","40min","60min","2hr","3hr","4hr","6d"),
               c(0,0.33,0.67,1,2,3,4,144)) %>%
  as.data.frame(stringsAsFactors=FALSE)
names(t.order) = c("condition","time")

# specify new conditions as times (factor)
conditions = sapply(as.character(times),function(x){
  t.order$time[which(t.order$condition==x)]})
conditions = factor(conditions, levels=c(0,0.33,0.67,1,2,3,4,144))

# set new deseq object
colData = as.data.frame(conditions)
deseq_obj = DESeqDataSetFromMatrix(countData=counts,
                  colData=colData, design=~conditions)
sizeFactors(deseq_obj) = estimateSizeFactorsForMatrix(counts)

# remove the 6day data
counts = counts(deseq_obj) %>% as.data.frame
counts_preadip = counts[,-grep("6d",names(counts))]
sizefac_preadip = sizeFactors(deseq_obj)[-grep("6d",names(counts))]
conditions_preadip=colData(deseq_obj)$conditions[-grep("6d",names(counts))]

# set new deseq object
colData_preadip = as.data.frame(conditions_preadip)
deseq_obj_preadip = DESeqDataSetFromMatrix(countData=counts_preadip,
                                           colData=colData_preadip,
                                           design=~conditions_preadip)
sizeFactors(deseq_obj_preadip) = sizefac_preadip
```

We identify peaks with dynamic profiles, considering the pre-adipogenesis time points, using a likelihood ratio test from the `DESeq2` package.

```
############################################################
## implement differential peak analysis
############################################################

# differential expreassion analysis - temporal dynamics based on LRT
deg_preadip = DESeq(deseq_obj_preadip, test="LRT",
                    full=~conditions_preadip,  reduced=~1)
res_preadip = results(deg_preadip)
```

Here we identify significantly time-varying peaks and peaks that are unchanged with respect to time, and we isolate the 100 bp intervals bracketing the respective peak summits.

```
############################################################
## isolate dynamic peaks for meme
############################################################

# set significant and unsignificant peaks
ind.sig = which(res_preadip$padj<sig.thresh)
ind.un = which(res_preadip$padj>sig.un)
length(ind.sig)
length(ind.un)

# function to find peak info from rownames of the results frame
get.map.from.res = function(res,map){
  namen = rownames(res)
  chrs = sapply(namen,function(x){strsplit(x,":")[[1]][1]})
  ends = sapply(namen,function(x){strsplit(x,"-")[[1]][2]})
  strs = sapply(namen,function(x){y=strsplit(x,"-")[[1]][1];
    return(strsplit(y,":")[[1]][2]) })
  coords = cbind(chrs,strs,ends) %>% as.data.frame(stringsAsFactors=F)
  out = row.match(coords, map[,1:3])
  return( cbind(res, map[out,]) )
} # get.map.from.res

# get data for sig and unsig peaks
sig.pks = get.map.from.res(res=res_preadip[ind.sig,], map=bed.map)
uns.pks = get.map.from.res(res=res_preadip[ind.un,], map=bed.map)
```

```
# set coordinates around each summit based on pk.dist
pk.coords.for.meme = function(res=NULL, pk.dist=NULL){
  out = c()
  for(ii in 1:nrow(res)){
    pks = sapply(res$summit[ii],function(x){
         strsplit(x,",")[[1]]}) %>% as.numeric
    for(jj in 1:length(pks)){
      new = c(res$chr[ii],pks[jj]-pk.dist,pks[jj]+pk.dist,rownames(res)[ii],
              signif(res$log2FoldChange[ii],3),signif(res$padj[ii],3) )
      out = rbind(out,new)
    } # jj
  } # ii
  out = out %>% as.data.frame(stringsAsFactors=FALSE)
  names(out) = c("chr","start","end","peakID","log2fc","fdr")
  rownames(out) = c(1:nrow(out))
  return(out)
} # pk.coords.for.meme

out.sig = pk.coords.for.meme(res=sig.pks, pk.dist=pk.dist)
out.uns = pk.coords.for.meme(res=uns.pks, pk.dist=pk.dist)
```

We log transform the normalized read data for the dynamic peaks, compute z-scores, and average across time points. The results of these computations is submitted to `STEM` for time-series clustering.

```
############################################################
## data transformations for STEM
############################################################

# significance filter
ind.sig = sapply(out.sig$peakID,function(x){
     which(rownames(deseq_obj_preadip)==x)
})

# log transformation
log2_exp = rlogTransformation(deseq_obj_preadip[ind.sig,], blind=TRUE)
log.pk.cnt = assay(log2_exp)

# z-score the data for each peak
pk.z.scores = scale(t(log.pk.cnt)) %>% t

# get mean z-scores for each time point
times = c("t0","20min","40min","60min","2h","3h","4h")
t.profiles = sapply(times,function(x){
  apply(pk.z.scores[,grep(x,colnames(pk.z.scores))],1,mean)
})
colnames(t.profiles) = c(0,0.33,0.67,1,2,3,4)

# revise names for timeseries clustering
gene_symbol = rownames(t.profiles)
out = cbind(gene_symbol, t.profiles)
write.table(out,"pkdata_20190828.txt",sep="\t",quote=F,
     col.names=T,row.names=F)
```

STEM is a java application with a GUI that can be implemented using the command line as follows. We used the defult parameters and set the number of clusters to 20 based on preliminary analyses. The file *pkdata_20190828.txt* was submitted for the analysis.

```
java -mx1024M -jar /stem/stem.jar
```

The STEM software ouput, consisting of dynamic cluster identifiers for each ATAC peak, was saved in *time_cluster_results_20190828.txt* and the results were analyzed in R.

```
##############################################################
## process/graph STEM clustering results
##############################################################

# function to capitalize first letter
simpleCap <- function(x) {
  s <- strsplit(x, " ")[[1]]
  paste(toupper(substring(s, 1,1)), substring(s, 2),
        sep="", collapse=" ")
}

# import STEM results, adjust gene names
stem.res0 = read.table("time_cluster_results_20190828.txt",sep="\t",
                       header=T,fill=T,stringsAsFactors=F,skip=1)
stem.res = stem.res0 %>% select(gene_symbol, Profile)
stem.res$gene_symbol = tolower(stem.res$gene_symbol)

# threshold number of profiles for plotting
n.profiles = 0

# generate plots
library(graphics)
tt = c(0,0.33,0.67,1,2,3,4)
pdf("ATACpeak_timeseries_clusters.pdf"); par(mfrow=c(3,3))
for(ii in 1:length(unique(stem.res$Profile))){
  pr = unique(stem.res$Profile)[ii]
  pk.pr = stem.res$gene_symbol[which(stem.res$Profile==pr)]
  if(length(pk.pr) < n.profiles){next}
  ex.pr = t.profiles[rownames(t.profiles) %in% pk.pr,]
  av.pr = apply(ex.pr,2,mean)
  tp = rep("l",nrow(ex.pr))
  ly = rep(1,nrow(ex.pr))
  matplot(tt,t(ex.pr),type=tp,lty=ly,col="black",
          ylab=paste0("profile ",pr),main="",xlab="time (hrs)")
  lines(tt,av.pr,col="red",cex=4)
}
dev.off()
```

Finally, we prepare the output for *de novo* motif analysis.

```
##############################################################
## output profiles of interest for meme analysis
##############################################################

# clusters
profiles = unique(stem.res$Profile)

# loop through profiles and write data for meme
for(ii in profiles){
  pk.pr = stem.res$gene_symbol[which(stem.res$Profile==ii)] %>%
    unlist
  pk.un = out.uns$peakID[sample.int( nrow(out.uns),
       min(nrow(out.uns), length(pk.pr)) )]
  fname = paste0("preadip_sigDynamics_profile",ii,".bed")
  out = out.sig[out.sig$peakID %in% pk.pr,]
  write.table(out,fname,sep="\t",quote=FALSE,
       col.names=FALSE,row.names=FALSE)
  fname = paste0("preadip_unDynamics_profile",ii,".bed")
  out = out.uns[out.uns$peakID %in% pk.un,]
  write.table(out,fname,sep="\t",quote=FALSE,
       col.names=FALSE,row.names=FALSE)
} ## ii, loop profiles
```

4.3 Regulatory element data preparation for *de novo* motif analysis

We process the `dREG` data in R to generate BED coordinate files that will be used for *de novo* motif analysis with `MEME` (Bailey *et al.*, 2009). The code can be found in *dREGanalysis.R*. First we load packages, specify analysis parameters, and load the dREG data.

```
library(dplyr)
library(DESeq2)
library(bigWig)

############################################################
## key analysis parameters
############################################################

# set number of base pairs around peaks for motif analysis
pk.dist = 50

# parameters for designating dynamic peaks
sig.thresh = 0.1
fc.thresh = 1

# parameters for designating non-dynamic peaks
sig.un = 0.5
fc.un = 0.25

###################################################################
## read in dREG data and perform some necessary processing
###################################################################

# read in data
fname = "out.dREG.peak.full.bed"
dreg0 = read.table(fname,header=F,stringsAsFactors=F,sep="\t")
names(dreg0) = c("chr","start","end","score","prob","center")
```

Note that 0.2% of the dREG peaks have documented summits outside of the peak regions. The outlier summits were either identical to the start/end coordinates or 10 bp outside of the start/end coordinates. In the latter cases, we adjusted the summits to be 1 bp within the peak boundries. Following this data adjustment, we separate plus strand and negative strand peaks.

```
# note that dREG gives some summits outside of
# the coordinate range (0.2 %)
hist(dreg0$end - dreg0$center)
100*length(which(dreg0$end < dreg0$center)) / nrow(dreg0)

# shift the summits to avert outside summits
dreg = dreg0
ind.hi = which(dreg0$end <= dreg0$center)
ind.lo = which(dreg0$start >= dreg0$center)
dreg$center[ind.hi] = dreg0$center[ind.hi] -
     (dreg0$center-dreg0$end)[ind.hi] - 1
dreg$center[ind.lo] = dreg0$center[ind.lo] +
     (dreg0$start-dreg0$center)[ind.lo] + 1
hist(dreg$end - dreg$center)
hist(dreg$center - dreg$start)

# separate positive and negative regions
# negative: start to summit
# positive: summit to end
dreg.minus = dreg %>% select(chr,start,center)
dreg.plus = dreg %>% select(chr,center,end)
min(dreg.minus[,3] - dreg.minus[,2])
min(dreg.plus[,3] - dreg.plus[,2])
```

Next we map PRO-seq reads to the regulatory elements identified by `dREG`. The following function imports PRO-seq reads from bigWig files and maps them to specified BED6 coordinates. Individual bigWig files were generated as described here `https://github.com/WarrenDavidAnderson/genomeAnalysis/tree/master/PROseq`.

```
# function to import proseq read data maped to specific coordinates
# input: bed = bed file for mapping reads
# input: bigWig.path = path to bigWig files
# input: file.prefix = file prefix
# input: file.suffix = file suffix
# input: min.length = min gene length
# output: data frame with reads mapped for each condition
get.counts.interval <- function(bed.plus=NULL, bed.minus=NULL,
                                file.prefix=NULL, file.suffix=NULL,
                                bigWig.path=NULL) {

  # output files
  vec.names = c()
  out.plus = data.frame(matrix(ncol = 0, nrow = nrow(bed.plus)))
  out.minus = data.frame(matrix(ncol = 0, nrow = nrow(bed.plus)))
  output = data.frame(matrix(ncol = 0, nrow = nrow(bed.plus)))

  # loop through each condition and load reads
  for(bigWig in Sys.glob(file.path(bigWig.path,
                                   paste0(file.prefix,
        paste0("*",file.suffix))))) {
    factor.name = strsplit(bigWig, "/")[[1]]
    factor.name = strsplit(factor.name[length(factor.name)],
          file.suffix)[[1]][1]
    factor.name = strsplit(factor.name,file.prefix)[[1]][2]
    strand = strsplit(factor.name,"sample_")[[1]][2]
    factor.name = strsplit(factor.name,"_sample")[[1]][1]
    loaded.bw = load.bigWig(bigWig)
    if(strand=="plus"){
      vec.names = c(vec.names, factor.name)
      reads = bed.region.bpQuery.bigWig(loaded.bw, bed.plus)
      out.plus = cbind(out.plus, reads)
      print(factor.name)
    }
    if(strand=="minus"){
      reads = bed.region.bpQuery.bigWig(loaded.bw, bed.minus)
      out.minus = cbind(out.minus, reads)
    }
  } # bigWig

  # annotation and output
  output = as.data.frame(out.plus + out.minus)
  names(output) = vec.names
  r.names = paste0(bed.plus[,1],':',bed.minus[,3])
  row.names(output) = r.names
  return(output)
} # get.counts.interval
```

Here we map the PRO-seq reads to the regulatory elements identified from `dREG`.

```
###################################################################
## map PRO reads to dREG elements
###################################################################

# map PRO data to dreg regions
# note that feature names are chr:summit
bigWig.path = paste0("/media/wa3j/Seagate2/Documents/PRO/adipogenesis",
                     "/July2018/bigWig_20181001/pro_preadip_bigWig")
file.prefix = "3T3_"
file.suffix = ".bigWig"
reads0 = get.counts.interval(bed.plus=dreg.plus,
                             bed.minus=dreg.minus,
                             bigWig.path=bigWig.path,
                             file.prefix=file.prefix,
                             file.suffix=file.suffix)
```

We normalize the data based on read depth within the regulatory elements, using `DESeq2` size factors (Love *et al.*, 2014).

```
############################################################
## normalize raw counts to size factors, generate DESeq object
############################################################

# estimate size factors
size_factors = estimateSizeFactorsForMatrix(reads0)

## generate DESeqDataSet object and transform count data to log2 scale
## note. enter size factors before implementing log transform
conditions = sapply(names(reads0),function(x)strsplit(x,"_")[[1]][1])
colData = as.data.frame(conditions)
deseq_obj_sizefac = DESeqDataSetFromMatrix(countData=reads0,
                                           colData=colData,
                                           design=~conditions)
sizeFactors(deseq_obj_sizefac) = size_factors


############################################################
## adjust data organization
############################################################

# basic counts, size factors, and annotation
counts = counts(deseq_obj_sizefac)
sizefac = sizeFactors(deseq_obj_sizefac)
times = colData(deseq_obj_sizefac)$conditions
```

```
t.order = cbind(c("t0","t20min","t40min","t60min","t2h","t3h","t4h","t6d"),
                c(0,0.33,0.67,1,2,3,4,144)) %>%
  as.data.frame(stringsAsFactors=FALSE)
names(t.order) = c("condition","time")

# specify new conditions as times (factor)
conditions = sapply(as.character(times),function(x){
  t.order$time[which(t.order$condition==x)]})
conditions = factor(conditions, levels=c(0,0.33,0.67,1,2,3,4,144))

# set new deseq object
colData = as.data.frame(conditions)
deseq_obj = DESeqDataSetFromMatrix(countData=counts,
                                   colData=colData, design=~conditions)
sizeFactors(deseq_obj) = estimateSizeFactorsForMatrix(counts)
```

We perform all pairwise analyses between time points up to 4 hrs to identify changes in regulatory regions.

```
############################################################
## generate all pairwise comparisons
############################################################

# loop through all pairwise combinations and run DESeq
condits = c(0,0.33,0.67,1,2,3,4) %>% as.character
res.pairs = list()
for(ii in 1:(length(condits)-1)){
  for(jj in (ii+1):length(condits)){

    print(paste0("compare ",condits[ii]," to ",condits[jj]))

    # basic data annotation
    ind_ii = which(conditions == condits[ii])
    ind_jj = which(conditions == condits[jj])

    # set new deseq object
    des = conditions[c(ind_ii,ind_jj)]
    colData_ij = as.data.frame(des)
    cnt_ij = counts[,c(ind_ii,ind_jj)]
    deseq_obj_preadip = DESeqDataSetFromMatrix(countData=cnt_ij,
                                               colData=colData_ij,
                                               design=~des)
    sizeFactors(deseq_obj_preadip) = sizefac[c(ind_ii,ind_jj)]
    colData(deseq_obj_preadip)$condition = des

    # pairwise DEG analysis
    dds = DESeq(deseq_obj_preadip)
    res = results(dds)[order(results(dds)$padj),]
    res.pairs[[paste0(condits[ii],"_",condits[jj])]] = res

  } # jj
} # ii
```

Finally, we process the pairwise analyses and output data files for the subsequent MEME analysis. Because the dREG peaks are generally much smaller than the ATAC peaks, we set the significance threshold to FDR = 0.1 to increase the sensitivity of the analysis (*sig.thresh* = 0.1). All other parameters were identical to those used for the ATAC analysis described elsewhere . For instance, we consider 50 bp around each differentially expressed for motif analysis. Three files are produced for each comparison: increased, decreased, and unchanged regulatory element coordinates.

```
############################################################
## generate output for meme
############################################################

# function to find peak info from rownames of the results frame
get.map.from.res = function(res=NULL, pk.dist=NULL){
  namen = rownames(res)
  chrs = sapply(namen,function(x){strsplit(x,":")[[1]][1]})
  mids = sapply(namen,function(x){strsplit(x,":")[[1]][2]}) %>% as.numeric
  strs = mids - pk.dist
  ends = mids + pk.dist
  out = cbind(chrs,strs,ends,rownames(res),
    res$log2FoldChange,"+",res$padj) %>%
    as.data.frame(stringsAsFactors=F)
  names(out) = c("chr","start","end","peak_name","log2fc","str","fdr")
  return(out)
} # get.map.from.res

# loop through all pairwise data and write output for meme
for(ii in 1:length(res.pairs)){

  # basic annotation
  comp_ii = names(res.pairs)[[ii]]
  res_ii = res.pairs[[ii]]
  indsig = which(res_ii$padj<sig.thresh &
     abs(res_ii$log2FoldChange)>fc.thresh)
  ind.up = which(res_ii$padj<sig.thresh &
     res_ii$log2FoldChange>fc.thresh)
  ind.dn = which(res_ii$padj<sig.thresh &
     res_ii$log2FoldChange<(-1)*fc.thresh)
  ind.un = which(res_ii$padj>sig.un &
     abs(res_ii$log2FoldChange)<fc.un)

  # if there are more insignificant peaks, use the number of sig peaks
  # select those with the highest FDRs
  if(length(indsig)==0){next}
  if(length(ind.un) > length(indsig)){ind.un=rev(ind.un)[1:length(indsig)]}
  # generate output for meme
  out.sig.up = get.map.from.res(res=res_ii[ind.up,], pk.dist=pk.dist)
  out.sig.dn = get.map.from.res(res=res_ii[ind.dn,], pk.dist=pk.dist)
  out.uns = get.map.from.res(res=res_ii[ind.un,], pk.dist=pk.dist)

  # set directory and output data for meme analysis
  dir_ii = paste0("meme_",comp_ii)
  system(paste0("mkdir ",dir_ii))
  setwd(dir_ii)
  fname = paste0("upsig_",comp_ii,".bed")
  write.table(out.sig.up,fname,sep="\t",
     quote=FALSE,col.names=FALSE,row.names=FALSE)
  fname = paste0("downsig_",comp_ii,".bed")
  write.table(out.sig.dn,fname,sep="\t",
     quote=FALSE,col.names=FALSE,row.names=FALSE)
  fname = paste0("unsig_",comp_ii,".bed")
  write.table(out.uns,fname,sep="\t",
     quote=FALSE,col.names=FALSE,row.names=FALSE)
  setwd(dir)

} # ii
```

4.4 `MEME` analysis of differentially expressed ATAC peaks

The analyses were completed using job array submissions to a SLURM system. For running an array job in the SLURM system, we revise the file names. Instead of file names that ndicate the pairwise timepoint comparisons, we substitute integer identifers as follows (see *reformat_deg.sh*).

```
######################################################
## get all individual bed files in the same place
######################################################

folders=$(ls -d */)
for ii in ${folders}
do
cd ${ii}
cp *.bed ..
cd ..
done


######################################################
## R script to change file names and save name key
# generates mapping between 1-20 and time comparisons
######################################################

dir = "~/allbed"
setwd(dir)
f.dn = list.files(pattern="downsig")

# loop through files, change file names, save key
cnt = 1
key = c()
for(ii in f.dn){

condit = strsplit(ii,"sig_")[[1]][2]
condit = strsplit(condit,".bed")[[1]][1]

comm1 = paste0("cat downsig_",condit,".bed > downsig_",cnt,".bed")
comm2 = paste0("cat upsig_",condit,".bed > upsig_",cnt,".bed")
comm3 = paste0("cat unsig_",condit,".bed > unsig_",cnt,".bed")
system(comm1)
system(comm2)
system(comm3)

new = c(condit, cnt)
key = rbind(key, new)
cnt = cnt + 1

}
colnames(key) = c("comparison","id")
rownames(key) = key[,2]
key = as.data.frame(key,stringsAsFactors=F)

# output key
fname = "memeTimeKey.txt"
write.table(key,fname,sep="\t",col.names=T,row.names=F)

# separate files based on identifier
system("mkdir array") # ordinal identifier for slurm array
for(ii in 1:nrow(key)){
```

```
fname = list.files(pattern=paste0("sig_",ii,".bed"))
comm = paste0("mv -t array ",paste(fname,collapse=" "))
system(comm)
}
```

Next we use code from two *.sh* files that implement the analysis with a background model using the SLURM system. The first file is the main file called using the sbatch command and the second file implements the analysis in parallel for each pairwise timepoint comparison. The following code can be found in *meme_deg_bkg_0.1.sh*. First, the main file bkg_de_slurm.sh is created.

```
###############################################################
## slurm script, bkg_de_slurm.sh
###############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=/motifs/meme/bkg_0.1_de
cd $dir

# sbatch --array=1-20 bkg_de_slurm.sh
cnt=$SLURM_ARRAY_TASK_ID
$dir/bkg_de_slurm_memerun.sh $cnt
```

The second file bkg_de_slurm_memerun.sh implements the four MEME analyses: increased versus unchanged, increased versus decreased, decreased versus unchanged, and decreased versus increased.

```
###############################################################
## analysis script, bkg_de_slurm_memerun.sh
###############################################################

#!/bin/bash

# chmod u+x /motifs/meme/bkg_0.1_de/*.sh

cnt="$1"

dir=/motifs/meme/bkg_0.1_de
cd ${dir}

# subdirectory for specific analyses
mkdir condit_id_${cnt}
mv -t condit_id_${cnt} *_${cnt}.bed
cd condit_id_${cnt}

# modules, paths, & files
module load bedtools
module load ucsc-tools
PATH=$PATH:/home/wa3j/meme-5.0.3/bin
PATH=$PATH:/home/wa3j/meme-5.0.3/libexec/meme-5.0.3
mm10=/genomes/mm10/mm10.fa

# meme parameters
nmotifs=50
minw=5
```

```
maxw=15
maxsize=10000000
thresh=0.1
bkg=3
objfun=de

# condition identifiers
fsigup=upsig_${cnt}
fsigdn=downsig_${cnt}
funs=unsig_${cnt}
cond=${cnt}

# convert bed to fasta
fastaFromBed -fi ${mm10} -bed ${fsigup}.bed -fo ${fsigup}.fasta
fastaFromBed -fi ${mm10} -bed ${fsigdn}.bed -fo ${fsigdn}.fasta
fastaFromBed -fi ${mm10} -bed ${funs}.bed -fo ${funs}.fasta

# generate background models (order 3)
fasta-get-markov -m ${bkg} ${fsigup}.fasta up.bkg.fasta
fasta-get-markov -m ${bkg} ${fsigdn}.fasta dn.bkg.fasta

############################
# increased versus unchanged
comp=upvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile up.bkg.fasta \
-neg ${funs}.fasta \
${fsigup}.fasta

############################
# increased versus decreased
comp=upvdn

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile up.bkg.fasta \
-neg ${fsigdn}.fasta \
${fsigup}.fasta

############################
```

```
# decreased versus unchanged
comp=dnvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile dn.bkg.fasta \
-neg ${funs}.fasta \
${fsigdn}.fasta

############################
# decreased versus increased
comp=dnvup

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile dn.bkg.fasta \
-neg ${fsigup}.fasta \
${fsigdn}.fasta
```

The entire analysis can be implemented in for each pairwise comparison (condition) in parallel with the following command line entries.

```
chmod u+x *.sh
sbatch --array=1-20 bkg_de_slurm.sh
```

Similar to the MEME analyses with background models, we implemented the same comparisons without background models (i.e., increased versus decreased, increased versus unchanged, decreased versus increased, and decreased versus unchanged; see *meme_deg_nobkg_0.1.sh*). The SLURM codes are as follows.

```
###############################################################
## slurm script, nobkg_de_slurm.sh
###############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

# sbatch --array=1-20 nobkg_de_slurm.sh
cnt=${SLURM_ARRAY_TASK_ID}
```

```
/nobkg_de_slurm_memerun.sh ${cnt}


###############################################################
## analysis script, bkg_de_slurm_memerun.sh
###############################################################

#!/bin/bash

# chmod u+x /*.sh

cnt="$1"

# modules, paths, & files
module load bedtools
module load ucsc-tools
PATH=$PATH:/home/wa3j/meme-5.0.3/bin
PATH=$PATH:/home/wa3j/meme-5.0.3/libexec/meme-5.0.3
mm10=/genomes/mm10/mm10.fa

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
objfun=de

# condition identifiers
fsigup=upsig_${cnt}
fsigdn=downsig_${cnt}
funs=unsig_${cnt}
cond=${cnt}

# convert bed to fasta
fastaFromBed -fi ${mm10} -bed ${fsigup}.bed -fo ${fsigup}.fasta
fastaFromBed -fi ${mm10} -bed ${fsigdn}.bed -fo ${fsigdn}.fasta
fastaFromBed -fi ${mm10} -bed ${funs}.bed -fo ${funs}.fasta

############################
# increased versus unchanged
comp=upvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-neg ${funs}.fasta \
${fsigup}.fasta

############################
# increased versus decreased
```

```
comp=upvdn

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-neg ${fsigdn}.fasta \
${fsigup}.fasta

############################
# decreased versus unchanged
comp=dnvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-neg ${funs}.fasta \
${fsigdn}.fasta

############################
# decreased versus increased
comp=dnvup

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-neg ${fsigup}.fasta \
${fsigdn}.fasta
```

Next we implement `MEME` using background models with the classic objective function. In the previous analyses we compared one set of peaks to another set (e.g., increased versus unchanged) using the differential enrichment objective function (-objfun de). Using the classic objective function (-objfun classic), the E-value is approximated based on the information content of a given motif. The analysis is completed for both increased peaks and decreased peaks for each pair of time points. This SLURM code can implement the analysis (*meme_deg_bkg_classic_0.1.sh*).

```
###############################################################
## slurm script, bkg_classic_slurm.sh
```

```
###############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

# sbatch --array=1-20 bkg_classic_slurm.sh
cnt=${SLURM_ARRAY_TASK_ID}
/bkg_classic_slurm_memerun.sh ${cnt}

###############################################################
## analysis script, bkg_classic_slurm_memerun.sh
###############################################################

#!/bin/bash

# chmod /*.sh

cnt="$1"

# subdirectory for specific analyses
mkdir condit_id_${cnt}
mv -t condit_id_${cnt} *_${cnt}.bed
cd condit_id_${cnt}

# modules, paths, & files
module load bedtools
module load ucsc-tools
PATH=$PATH:/home/wa3j/meme-5.0.3/bin
PATH=$PATH:/home/wa3j/meme-5.0.3/libexec/meme-5.0.3
mm10=/genomes/mm10/mm10.fa

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
bkg=3
objfun=classic

# condition identifiers
fsigup=upsig_${cnt}
fsigdn=downsig_${cnt}
funs=unsig_${cnt}
cond=${cnt}

# convert bed to fasta
fastaFromBed -fi ${mm10} -bed ${fsigup}.bed -fo ${fsigup}.fasta
fastaFromBed -fi ${mm10} -bed ${fsigdn}.bed -fo ${fsigdn}.fasta
fastaFromBed -fi ${mm10} -bed ${funs}.bed -fo ${funs}.fasta

# generate background models (order 3)
fasta-get-markov -m ${bkg} ${fsigup}.fasta up.bkg.fasta
fasta-get-markov -m ${bkg} ${fsigdn}.fasta dn.bkg.fasta
```

```
############################
# increased
comp=upvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile up.bkg.fasta \
${fsigup}.fasta

############################
# decreased
comp=dnvun

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile dn.bkg.fasta \
${fsigdn}.fasta
```

We also applied the classic analysis without a background model (*meme_deg_nobkg_classic_0.1.sh*).

```
##############################################################
## slurm script, nobkg_classic_slurm.sh
##############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

# sbatch --array=1-20 nobkg_classic_slurm.sh
cnt=${SLURM_ARRAY_TASK_ID}
/nobkg_classic_slurm_memerun.sh ${cnt}


##############################################################
## analysis script, bkg_de_slurm_memerun.sh
##############################################################

#!/bin/bash

# chmod u+x /motifs/meme/nobkg_0.1_classic/*.sh
```

```
cnt="$1"

# modules, paths, & files
module load bedtools
module load ucsc-tools
PATH=$PATH:/home/wa3j/meme-5.0.3/bin
PATH=$PATH:/home/wa3j/meme-5.0.3/libexec/meme-5.0.3
mm10=/genomes/mm10/mm10.fa

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
objfun=classic

# condition identifiers
fsigup=upsig_${cnt}
fsigdn=downsig_${cnt}
funs=unsig_${cnt}
cond=${cnt}

# convert bed to fasta
fastaFromBed -fi ${mm10} -bed ${fsigup}.bed -fo ${fsigup}.fasta
fastaFromBed -fi ${mm10} -bed ${fsigdn}.bed -fo ${fsigdn}.fasta
fastaFromBed -fi ${mm10} -bed ${funs}.bed -fo ${funs}.fasta

############################
# increased
comp=up

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
${fsigup}.fasta

############################
# decreased
comp=dn

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
${fsigdn}.fasta
```

4.5 MEME analysis of dynamic ATAC peak clusters

We next implement MEME analyses for the dynamic peak clusters identified from ATAC data. The main analysis can be found in *meme_dynamicpeaks.sh*. First, we reformat the cluster identifiers for the SLURM array analysis (see *reformat_stem*).

```
#######################################################
## R script to change file names and save name key
#######################################################

# sort original profile ids
files = list.files(pattern = "sigDyn")
prof = sapply(files,function(x){strsplit(x,"_profile")[[1]][2]})
prof = sapply(prof,function(x){strsplit(x,".bed")[[1]][1]})
prof.sort = sort( as.numeric(prof) )

# loop through files, change file names, save key
setwd(new.dir)
cnt = 1
key = c()
for(ii in prof.sort){

sg.old = paste0(main.dir,"/preadip_sigDynamics_profile",ii,".bed")
sg.new = paste0("preadip_sigDynamics_profile",cnt,".bed")
un.old = paste0(main.dir,"/preadip_unDynamics_profile",ii,".bed")
un.new = paste0("preadip_unDynamics_profile",cnt,".bed")

comm1 = paste0("cat ",sg.old," > ",sg.new)
comm2 = paste0("cat ",un.old," > ",un.new)
system(comm1)
system(comm2)

new = c(ii, cnt)
key = rbind(key, new)
cnt = cnt + 1

}
colnames(key) = c("origprofile","id")
rownames(key) = key[,2]
key = as.data.frame(key,stringsAsFactors=F)

# write out the key
fname = "bedKey_STEM.txt"
write.table(key,fname,sep="\t",quote=F,col.names=T,row.names=F)
```

The following SLURM analysis will implement MEME as follows (meme_dynamicpeaks.sh). For each dynamic cluster, we compare the cluster peaks to the unchanged peaks with and without a third order background model. Similarly, we apply the classic objective function for each cluster with and without a background.

```
###############################################################
## slurm script, bkg_dyn_de_slurm.sh
###############################################################
```

```
#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=/motifs/meme/dynclust/bkg_0.1_de
cd ${dir}

# sbatch --array=1-18 bkg_dyn_de_slurm.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/bkg_dyn_de_slurm_memerun.sh ${cnt}

###############################################################
## analysis script, bkg_dyn_de_slurm_memerun.sh
###############################################################

#!/bin/bash

# chmod u+x /motifs/meme/dynclust/bkg_0.1_de/*.sh

cnt="$1"

dir=/motifs/meme/dynclust/bkg_0.1_de
cd ${dir}

# subdirectory for specific analyses
mkdir profile_id_${cnt}
mv -t profile_id_${cnt} *profile${cnt}.bed
cd profile_id_${cnt}

# modules, paths, & files
module load bedtools
module load ucsc-tools
PATH=$PATH:/home/wa3j/meme-5.0.3/bin
PATH=$PATH:/home/wa3j/meme-5.0.3/libexec/meme-5.0.3
mm10=/genomes/mm10/mm10.fa

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
bkg=3
objfun=de

# condition identifiers
fsig=dyn_${cnt}
funs=unsig_${cnt}
cond=${cnt}

# convert bed to fasta
fastaFromBed -fi ${mm10} -bed preadip_sigDynamics_profile${cnt}.bed \
 -fo ${fsigup}.fasta
fastaFromBed -fi ${mm10} -bed preadip_unDynamics_profile${cnt}.bed \
 -fo ${funs}.fasta
```

```
# generate background models (order 3)
fasta-get-markov -m ${bkg} ${fsigup}.fasta up.bkg.fasta

############################
# dynamic versus unchanged, with background
comp=upvun_bkg_de

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile up.bkg.fasta \
-neg ${funs}.fasta \
${fsigup}.fasta


############################
# dynamic versus unchanged, no background
comp=upvun_nobkg_de

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
objfun=de

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-neg ${funs}.fasta \
${fsigup}.fasta

############################
# dynamic, classic model with background
comp=upvun_bkg_classic

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
objfun=classic
```

```
# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
-bfile up.bkg.fasta \
${fsigup}.fasta


############################
# dynamic, classic model without background
comp=upvun_nobkg_classic

# meme parameters
nmotifs=50
minw=5
maxw=15
maxsize=10000000
thresh=0.1
objfun=classic

# generate log
exec &> log_${comp}_${cond}.txt
echo comparison ${cond}

# run meme
meme -o ${comp}_${cond} \
-dna -revcomp -objfun ${objfun} \
-nmotifs ${nmotifs} \
-minw ${minw} -maxw ${maxw} \
-evt ${thresh} -maxsize ${maxsize} \
${fsigup}.fasta
```

4.6 MEME analysis of differentially expressed dREG peaks

All MEME analyses described for the pairwise timepoint ATAC-seq expression analyses were completed in an analogous fashion for the pairwise timepoint dREG expression analyses. We do not display the code here for the purpose of brevity. The code files described above can be simply modified with the directories and file naming conventions for the dREG data. The following files can be used to implement the analyses: *reformat_deg.sh*, *meme_deg_bkg_0.1.sh*, *meme_deg_nobkg_0.1.sh*, *meme_deg_bkg_classic 0.1.sh*, and *meme_deg_nobkg_classic_0.1.sh*.

5 Processing *de novo* motif analysis significance results

For each MEME analysis documented above, we aggregate all *de novo* motif identification results from all analysis conditions. For the analyses using a background model, the command line code can be found in *processResults_meme_bkg.sh* (see below). Complementary codes for other analysis conditions can be found in these files: *processResults_meme_bkg_classic.sh*, *processResults_meme_nobkg_classic.sh*, and *processResults_meme_nobkg_de.sh*. The analysis entails looping through folders of results (one for each condition), then looping through folders with specific comparisons (e.g., increased versus unchanged). For each analysis, we rename the meme.txt output file and copy it to a central location.

```
#####################################################
## after the meme loop has run, collect all meme results
#####################################################

# main directory
dir=/motifs/meme/bkg_0.1_de
cd ${dir}

# directory for meme data
datdir=bkg_0.1_de_pairwise_memeResults
mkdir ${datdir}

# folders with meme data
folders=$(ls -d *condit_*/)

# loop through all data and aggregate meme.txt results
for ii in ${folders}
do

cd ${ii}

# loop through each subfolder and copy the Evalue data
folds=$(ls -d */)
for jj in ${folds}
do
cd ${jj}
id=$(echo ${jj} | awk -F"/" '{print $1}')
cat meme.txt > ${id}_meme.txt
cp ${id}_meme.txt ${dir}/${datdir}
cd ..
done

cd ..
done
```

Next we isolate all unique motifs with E-values that are less than a set threshold of 10^{-1}. This analysis is completed using both the unix command line as well as R.

```
#####################################################
## isolate all unique motifs with E-values
## that are less than a set threshold
#####################################################

subdir=bkg_0.1_de_pairwise_memeResults
dir=/motifs/meme/bkg_0.1_de
cd ${dir}/${subdir}

# document meme run conditions
comp=bkg_0.1_de

# isolate E-value data for each motif
for ii in *.txt
do
cond=$(echo ${ii} | awk -F"_meme.txt" '{print $1}')
cat ${ii} | grep E-value > ${comp}_${cond}_Eval.txt
done

# separate motif Evalue data and total meme output files
```

```
mkdir memeEval
mkdir memeAll
mv *Eval.txt* memeEval
mv *meme.txt* memeAll

cd ${dir}/${subdir}/memeEval

# remove all "Stopped because motif E-value >" lines
for ii in *.txt
do
sed -i "/\b\Stopped\b/d" ${ii}
done

# use R to loop through data and set identity all motifs of interest
Eval.thresh = 0.1 # E-value threshold

# loop to get all significant motifs
files = list.files(getwd())
info = file.info(files)
nonempty = rownames(info[info$size != 0, ])
Edat = matrix(NA,nrow=0,ncol=7)
for(ii in nonempty){
cond = strsplit(ii,"_Eval.txt")[[1]][1]
datii = read.table(ii,stringsAsFactors=FALSE)
indsig = which(datii[,18] < Eval.thresh)
if(length(indsig)==0){next}
new = cbind(cond, datii[indsig,2], datii[indsig,6], datii[indsig,9],
datii[indsig,12], datii[indsig,15], datii[indsig,18])
Edat = rbind(Edat, matrix(new,length(indsig),7))
}
Edat = as.data.frame(Edat,stringsAsFactors=FALSE)
names(Edat) = c("condition","motif","width","nsites","llr","pval","Eval")
Edat[,c(3:7)] = apply(Edat[,c(3:7)],2,function(x){
     as.numeric(data.matrix(x))})

# isolate unique motifs
# document lowest eval and highest llr
motifs.all = Edat
motifs.unq = matrix(0,0,8)
while( nrow(motifs.all)>0 ){
motif = motifs.all$motif[1]
ind = which(motifs.all$motif == motif)
condits = unique(motifs.all$condition[ind])
con = paste(condits, collapse=",")
ind.min = which(motifs.all$Eval[ind] ==
             min(motifs.all$Eval[ind]))
ind.max = which(motifs.all$llr[ind][ind.min] ==
             max(motifs.all$llr[ind][ind.min]))
  new = cbind(motifs.all[ind[ind.min][ind.max],c(2:7)], length(ind), con)
motifs.unq = rbind(motifs.unq, new)
motifs.all = motifs.all[-ind,]
}
motifs.unq = as.data.frame(motifs.unq,stringsAsFactors=FALSE)
names(motifs.unq) = c("motif","width","nsites","llr","pval","Eval",
     "ncond","condits")
nrow(Edat)
nrow(motifs.unq)
motifs.unq$condits = as.character(motifs.unq$condits)
```

```
# save.image("meme.res.for.tomtom.RData")
# load("meme.res.for.tomtom.RData")

# generate output file for generating a db of meme motifs
fname = "all_sig_motifs_bkg_0.1_de.txt"
write.table(Edat,fname,quote=F,sep="\t",col.names=T,row.names=F)
```

Finally, we isolate unique motif data (motif identifiers and position weight matrices or PWMs) to a central directory for subsequent analysis.

```
####################################################
## isolate key results for downstream analysis
####################################################

dir=/motifs/meme/bkg_0.1_de/bkg_0.1_de_pairwise_memeResults
cd ${dir}
mkdir pswm

# directory for meme pswms to be used with tomtom
# mkdir pswm
cp ${dir}/memeEval/all_sig_motifs_bkg_0.1_de.txt ${dir}/pswm
cp ${dir}/memeEval/*RData ${dir}/pswm
cd ${dir}/pswm
```

The codes for aggregating data corresponding to the other MEME analysis conditions, including differential dREG peak and dynamic ATAC peak cluster conditions, are analogous to those described above.

6 Identification of transcription factor motifs from *de novo* motifs

In the next set of analyses, we will identify transcription factor motifs that correspond to the *de novo* motifs. For this analysis, we identify all *de novo* that are identified in multiple MEME runs, and then we match these selected motifs against documented transcription factor motifs using TOMTOM. NOTE - RECHECK THIS SOME OF THE CODES IN THIS GITHUB HAVE BEEN UPDATED AND THIS DOCUMENT SHOULD BE AS WELL.

6.1 Isolation of *de novo* motif position weight matrices

The first step is to extract the position weight matrices (PWMs) for the de novo motifs (generate_denovoPWM_db.sh). First we specify the locations of the results from applying MEME to differentially expressed ATAC regions and we set a header for a file that will contain all of the PWMs.

```
# locations for all meme motif comparison data
f1=/meme/bkg_0.1_classic/bkg_0.1_classic_pairwise_memeResults/memeAll
f2=/meme/bkg_0.1_de/bkg_0.1_de_pairwise_memeResults/memeAll
f3=/meme/nobkg_0.1_de/nobkg_0.1_de_pairwise_memeResults/memeAll
f4=/meme/nobkg_0.1_classic/nobkg_0.1_classic_pairwise_memeResults/memeAll

# motif db file
outfile=denovoPWM.txt

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
```

```
cat header1 header2 header3 > ${outfile}
rm header1 header2 header3
```

Next we loop through all motifs identified using the classic objective with a third order background model.

```
#################
# loop through background classic meme motifs (f1)
subdir=${f1}
condit=deg

# loop through all motif ids and add the PWMs
ind=1
for ii in ${subdir}/*.txt
do
file=$(echo ${ii} | awk -F"${subdir}/" '{print $2}')
cond=$(echo ${file} | awk -F"_" '{print $2}')
motifs=$(cat ${ii} | grep MOTIF | awk -F"MOTIF " '{print $2}' | \
awk -F" MEME" '{print $1}')
for jj in ${motifs}
do
linenum=$(cat ${ii} | grep -n -w ${jj} | grep probability | \
awk -F":" '{print $1}')
if [ ! -z "${linenum}" ]; then
linenum=$(expr ${linenum} + 2)
tail -n +${linenum} ${ii} > topWM
lineend=$(cat topWM | grep -n "-----" | head -1 | awk -F":" '{print $1}')
head -$(expr $lineend - 1) topWM > PWM
printf '%s\n' '' 'MOTIF '${jj}_${condit}${cond}_${ind} > headerWM
mv ${outfile} tmp
cat tmp headerWM PWM > ${outfile}
rm tmp headerWM PWM
ind=$(expr ${ind} + 1)
fi
done
done
```

Similarly, we loop through the motifs identified under other conditions and add the respective PWMs to the PWM output file.

```
#################
# loop through background differential meme motifs (f2)
subdir=${f2}
condit=deg
for ii in ${subdir}/*.txt
do
file=$(echo ${ii} | awk -F"${subdir}/" '{print $2}')
cond=$(echo $file | awk -F"_" 'print $2')
motifs=$(cat ${ii} | grep MOTIF | awk -F"MOTIF " '{print $2}' | \
awk -F" MEME" '{print $1}')
for jj in ${motifs}
do
linenum=$(cat ${ii} | grep -n -w ${jj} | grep probability | \
awk -F":" '{print $1}')
if [ ! -z "${linenum}" ]; then
linenum=$(expr ${linenum} + 2)
tail -n +${linenum} ${ii} > topWM
lineend=$(cat topWM | grep -n "-----" | head -1 | awk -F":" '{print $1}')
```

```
head -$(expr ${lineend} - 1) topWM > PWM
printf '%s\n' '' 'MOTIF '${jj}_${condit}${cond}_${ind} > headerWM
mv ${outfile} tmp
cat tmp headerWM PWM > ${outfile}
rm tmp headerWM PWM
ind=$(expr ${ind} + 1)
fi
done
done

#################
# loop through no-background differential meme motifs (f3)
subdir=${f3}
condit=deg
for ii in ${subdir}/*.txt
do
file=$(echo ${ii} | awk -F"${subdir}/" '{print $2}')
cond=$(echo ${file} | awk -F"_" '{print $2}')
motifs=$(cat ${ii} | grep MOTIF | awk -F"MOTIF " '{print $2}' | \
awk -F" MEME" '{print $1}')
for jj in ${motifs}
do
linenum=$(cat ${ii} | grep -n -w ${jj} | grep probability | \
awk -F":" '{print $1}')
if [ ! -z "${linenum}" ]; then
linenum=$(expr ${linenum} + 2)
tail -n +${linenum} ${ii} > topWM
lineend=$(cat topWM | grep -n "-----" | head -1 | awk -F":" '{print $1}')
head -$(expr ${lineend} - 1) topWM > PWM
printf '%s\n' '' 'MOTIF '${jj}_${condit}${cond}_${ind} > headerWM
mv ${outfile} tmp
cat tmp headerWM PWM > ${outfile}
rm tmp headerWM PWM
ind=$(expr ${ind} + 1)
fi
done
done

#################
# loop through no-background classic meme motifs (f4)
subdir=${f4}
condit=deg
for ii in ${subdir}/*.txt
do
file=$(echo ${ii} | awk -F"${subdir}/" '{print $2}')
cond=$(echo ${file} | awk -F"_" '{print $2}')
motifs=$(cat ${ii} | grep MOTIF | awk -F"MOTIF " '{print $2}' | \
awk -F" MEME" '{print $1}')
for jj in ${motifs}
do
linenum=$(cat ${ii} | grep -n -w ${jj} | grep probability | \
awk -F":" '{print $1}')
if [ ! -z "${linenum}" ]; then
linenum=$(expr ${linenum} + 2)
tail -n +${linenum} ${ii} > topWM
lineend=$(cat topWM | grep -n "-----" | head -1 | awk -F":" '{print $1}')
head -$(expr ${lineend} - 1) topWM > PWM
printf '%s\n' '' 'MOTIF '${jj}_${condit}${cond}_${ind} > headerWM
```

```
mv ${outfile} tmp
cat tmp headerWM PWM > ${outfile}
rm tmp headerWM PWM
ind=$(expr ${ind} + 1)
fi
done
done
```

The PWMs associated with the differential dREG regions and dynamics ATAC peak clusters can be added to the PWM file in a simmilar manner.

Next we generate a separate file for each *de novo* PWM. These PWM files will be used in subsequent `TOMTOM` analyses.

```
####################################################
## generate pwm files for each de novo motif
####################################################

# isolate de novo motif ids and specify the original motif db file
motiffile=denovoPWM.txt
motifs=$(cat ${motiffile} | grep MOTIF | awk -F"MOTIF " '{print $2}')

# loop through de novo motifs and create pwm files
for ii in ${motifs}
do
linenum=$(cat ${motiffile} | grep -n -w ${ii} | awk -F":" '{print $1}')
line=$(expr ${linenum} + 2)
tail -n +${line} ${motiffile} > part
lineend=$(grep -E --line-number --with-filename '^$' part | head -1 | \
awk -F":" '{print $2}')
if [ -z "${lineend}" ]; then
endline=$(wc -l part | awk -F" " '{print $1}')
else
endline=$(expr ${lineend} - 1)
fi
head -${endline} part > tmp
awk '{OFS="\t";} {print $1,$2,$3,$4}' tmp > ${ii}.txt
rm part tmp
done
```

6.2 Identification of associated *de novo* motifs

Here we compare each *de novo* motif to every other *de novo* motif using `TOMTOM`. Associated code can be found in tomtom_eachVersusAll.sh. First we generate 60 list of *de novo* motifs for parallelized analysis with a SLURM array job.

```
####################################################
## first break all de novo motifs into 60 groups
## for array analysis
####################################################

# get a list of all motif ids
cat denovoPWM.txt | grep MOTIF | awk -F"MOTIF " '{ print $2}' > motif.ids

# R code to generate a set of motif lists
# each list will be processed with an individual job
motifs = read.table("motif.ids",header=F,stringsAsFactors=F)[,1]
narray = 59
```

```
nmotif = length(motifs)
motif.per = floor( nmotif / narray )
for(ii in 1:narray){
     ind1 = (ii-1) * motif.per + 1
     ind2 = ind1 + motif.per - 1
     ind = ind1:ind2
     out = motifs[ind]
     fname = paste0("motifs_",ii,".txt")
     write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
     if(ii==narray){
          ii = ii+1
          ind = (ind2+1):length(motifs)
          out = motifs[ind]
          fname = paste0("motifs_",ii,".txt")
          write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
     }
}
```

The SLURM array job submission entails the preparation of two files: tomtom_main.sh and tomtom_run.sh. Thie main file is structured as follows.

```
###########################
## slurm script, tomtom_main.sh
###########################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=~/motifdb
cd $dir

# sbatch --array=1-60 tomtom_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
$dir/tomtom_run.sh ${cnt}
```

The following file was prepared for running `TOMTOM`.

```
###########################
## tomtom run script, tomtom_run.sh
###########################

#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=~/motifdb
cd ${dir}

# subdirectory for specific analyses
mkdir condit_id_${cnt}
cp motifs_${cnt}.txt condit_id_${cnt}
cd condit_id_$cnt
```

```
# specify output file name
outfile=tomtom_${cnt}_denovo.txt

# motif pwm directory and files/identifiers
tfdir=${dir}/alldenovo
motifs=$(cat motifs_${cnt}.txt)

# tomtom information
motiffile=denovoPWM.txt
tomtom=/home/wa3j/meme-5.0.3/bin/tomtom
motifdb=${dir}/${motiffile}
eval=66.93 # p < 0.001 for old run, filter in R
minoverlap=5

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > header
rm header1 header2 header3

# tomtom data
printf '%s\n' 'Query_ID Target_ID Optimal_offset p-value E-value q-value \
Overlap Query_consensus Target_consensus Orientation' | \
tr ' ' '\t' > ${outfile}

# loop through all TF averages
for tf in ${motifs}
do
echo MOTIF ${tf} > line1
echo letter-probability matrix: > line2
cat header line1 line2 ${tfdir}/${tf}.txt > meme.txt
${tomtom} -dist pearson -no-ssc -evalue -thresh ${eval} -m ${tf} \
-min-overlap ${minoverlap} meme.txt ${motifdb}
cd tomtom_out
mv tomtom.tsv ..
cd ..
rm -r tomtom_out
tail -n +2 tomtom.tsv | head -n -4 > new
mv ${outfile} tmp
cat tmp new > ${outfile}
rm tomtom.tsv new tmp
rm line1 line2 meme.txt
done

cp ${outfile} ..
```

The following code can be entered into the command line to run the analysis.

```
chmod u+x *.sh
sbatch --array=1-60 tomtom_main.sh
```

The results of the paralellized motif matching `TOMTOM` analyses can aggregated as follows.

```
############################################################
## aggregate data and move for local analysis in R
```

```
############################################################

# tomtom data and header
outfile=tomtomDEvDE.txt
printf '%s\n' 'Query_ID Target_ID Optimal_offset p-value E-value q-value \
Overlap Query_consensus Target_consensus Orientation' | \
tr ' ' '\t' > ${outfile}

# loop through all results and combine data
for ii in *_denovo.txt
do
tail -n +2 ${ii} > new
mv ${outfile} tmp
cat tmp new > ${outfile}
rm tmp new
done
```

Next we identify *de novo* motifs that match other *de novo* motifs as described in motif_communities_memeMotif.R. In this analysis we are isolating distinct though highly similar motifs that are identified through multiple `MEME` analysis conditions. For this analysis, we used a permissive p-value threshold of 0.001. This analysis generates a list of motifs identified through multiple analyses.

```
##############################################################
## import data and generate three column graph format
##############################################################

library(dplyr)

# de novo motif pwm dir
pwm.dir = paste0("~/alldenovo")

# original motif list for each versus all analysis
fname = "motif.ids"
motifs0 = read.table(fname,header=F,stringsAsFactors=F)[,1]

# basic data
fname = "tomtomDEvDE.txt"
dat0 = read.table(fname,header=T,stringsAsFactors=F)
dat0 = dat0 %>% filter(p.value < 0.001)

# remove self to self edges
ind.rem = which(dat0$Query_ID == dat0$Target_ID)
dat0 = dat0[-ind.rem,]

# generate edge data with -log10(pval)
threecol = dat0 %>% select(Query_ID, Target_ID, p.value)
names(threecol) = c('from','to','pval')
weight = -log10(threecol$pval)
threecol$weight = weight

# unique tomtom motifs
tomtom.uniq = unique(c(threecol$from, threecol$to))
length(tomtom.uniq)

# write out motifs with matches
fname = "denovo2901.txt"
write.table(tomtom.uniq,fname,col.names=F,row.names=F,sep="\t",quote=F)
```

The motifs identified through these analyses can be matched against transcription factor motifs, documented in public databases, by using TOMTOM to compare the respective PWMs.

6.3 Identification of transcription factor motifs associated with *de novo* motifs

In order to match *de novo* motifs to documented transcription factor (TF) motifs, we first isolate the PWMs from TF databases of interest. We consider both databases that are available through the MEME suite (Bailey *et al.*, 2009) and TF PWMs available from HOMER (Heinz *et al.*, 2010). The MEME suite already contains TF motifs from CIS-BP (Weirauch *et al.*, 2014) and JASPAR (Khan *et al.*, 2018) that we include in our analysis. We isolated selected MEME database motif files as follows (see getDbPWM.sh). Note that we downloaded the databases from the MEME website on 3/12/2018.

```
#####################################################
## download databases 3/12/2018
#####################################################

# download link
http://meme-suite.org/doc/download.html
Motif Databases (updated 28 Aug 2018)
mv motif_databases.12.181.tgz memeTF.tgz
gunzip memeTF.tgz

# unpack databases
tar -xvf memeTF.tar

#####################################################
## select specific databases for further analysis
#####################################################

# cisbp
cd CIS-BP
cp Mus_musculus.meme Rattus_norvegicus.meme Homo_sapiens.meme ..

# all eukaryote
cd EUKARYOTE
cp *.meme ..

# all mouse
cd MOUSE
cp *.meme ..

# jaspar
cd JASPAR
cp *vertebrates* ..

# TFBS shape
cd TFBSshape
cp *.meme ..
```

We also download the HOMER website PWMs and parse the data into an appropriate format for combining the PWMs with MEME database PWMs. We downloaded HOMER PWMs during the week of 3/12/2018. Code can be found in add_Homer.sh.

```
wget http://homer.ucsd.edu/homer/custom.motifs

# specify name for home motif file
outfile=HOMER.meme
infile=custom.motifs
```

```
# specify motifs
motifsids=( $(cat custom.motifs | grep ">" | awk -F" " '{print $2}') )
motifsseq=( $(cat custom.motifs | grep ">" | awk -F" " '{print $1}' | \
awk -F">" '{print $2}') )
nmotifs=$(cat custom.motifs | grep ">" | wc -l)

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > ${outfile}
rm header1 header2 header3

# loop through motifs and generate PWM file
for ii in `seq 0 $(expr ${nmotifs} - 1)`
do
tf=$(echo ${motifsids[ii]})
id=$(echo ${motifsseq[ii]})
linestart=$(grep -w -n ${tf} ${infile} | grep -w ${id} | \
awk -F":" '{print $1}')
linestart=$(expr ${linestart} + 1)
tail -n +${linestart} ${infile} > topWM
lineend=$(cat topWM | grep -n ">" | head -1 | awk -F":" '{print $1}')
if [ -z "$lineend" ]
then
lineend=$(wc -l topWM | awk -F" " '{print $1}')
lineend=$(expr ${lineend} + 1)
fi
head -$(expr ${lineend} - 1) topWM > PWM
echo MOTIF ${tf}_${id} > line1
echo > line3
echo letter-probability matrix: > line2
mv ${outfile} tmp
cat tmp line1 line2 PWM line3 > ${outfile}
rm topWM PWM line1 line2 line3 tmp
done
```

The following code aggregates all TF motif PWMs into a single file with a header and format compatible with the subsequent `TOMTOM` analysis.

```
# aggregate all meme files
cat *.meme > ALLmeme.txt

# get list of all motif ids
cat ALLmeme.txt | grep MOTIF | awk -F"MOTIF " '{print $2}' \
> all.TFmotif.ids.txt

# get TFid array and number of TFs
readarray tfs < all.TFmotif.ids.txt
ntf=$(wc -l all.TFmotif.ids.txt | awk -F" " '{print $1}')

# raw data file with all meme motifs
motiffile=ALLmeme.txt

# motif identifier key
echo "idnum orig" > motif.id.key.txt
```

```
# create motif database file
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > TFmotifs.txt
rm header1 header2 header3

# loop through all TFs and collect PWMs into a database file
# retain the motif id key
for ii in `seq 0 $(expr ${ntf} - 1)`
do

tfid=$(echo ${tfs[ii]})
linenum=$(cat ${motiffile} | grep -n -w "${tfid}" | hexdump -C | \
head -1 | awk -F":" '{print $1}' | awk -F"|" '{print $2}')
tail -n +${linenum} ${motiffile} > part1
linestart=$(cat part1 | grep -n "letter-probability matrix:" | \
head -1 | cut -d':' -f1)
tail -n +${linestart} part1 > part2
lineend=$(cat part2 | grep -n -e '^[[:space:]]*$' | head -1 | cut -d':' -f1)
cat part2 | head -${lineend} > PWM

echo "MOTIF motif${ii}" > newid
mv TFmotifs.txt tmp
cat tmp newid PWM > TFmotifs.txt
mv motif.id.key.txt tmp
echo "motif${ii} ${tfid}" > newid
cat tmp newid > motif.id.key.txt
rm part1 part2 PWM tmp newid

done

# remove URLs
grep -vwE "(URL)" TFmotifs.txt > tmp
rm TFmotifs.txt
mv tmp TFmotifs.txt
```

To determine which *de novo* motifs are associated with TF motifs, we use TOMTOM to compare the respective PWMs. The code for this analysic can be found in tomtom_denovo_vs_db_array.sh. As in previously described analyses, we implement a SLURM array job for parallelization. We first separate the *de novo* motifs into 60 motif lists.

```
# R code to generate a set of motif lists
# each list will be processed with an individual job
narray = 59
library(dplyr)
motifs = read.table("denovo2901.txt",stringsAsFactors=F,header=F)
nmotif = nrow(motifs)
motif.per = floor( nmotif / narray )
for(ii in 1:narray){
   ind1 = (ii-1) * motif.per + 1
   ind2 = ind1 + motif.per - 1
   ind = ind1:ind2
   out = motifs[ind,1]
   fname = paste0("motifs_",ii,".txt")
   write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
   if(ii==narray & ind[length(ind)]!=nmotif){
      ii = ii + 1
      ind1 = ind2 + 1
      ind2 = nmotif
      ind = ind1:ind2
      out = motifs[ind,1]
      fname = paste0("motifs_",ii,".txt")
      write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
   }
}
```

The main SLURM script is as follows.

```
###############################################################
## slurm script, tomtom_main.sh
###############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=~/tomtomDENOVOvsTF
cd $dir

# sbatch --array=1-60 tomtom_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/tomtom_run.sh ${cnt}
```

The script with the `TOMTOM` analysis is as follows. Note that we use an E-value cutoff of 66 because this Bonferroni-corrected value corresponds to a p-value of 0.001 for an individual comparison between two motifs.

```
###############################################################
## tomtom run script, tomtom_run.sh
## Run tomtom for de novo motifs against database TFs
###############################################################

#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=~/tomtomDENOVOvsTF
cd ${dir}

# subdirectory for specific analyses
mkdir condit_id_${cnt}
cp motifs_${cnt}.txt condit_id_${cnt}
cd condit_id_${cnt}

# specify output file name
outfile=tomtom_${cnt}_denovo.txt

# directory with de novo motif pwms
mdir=~/motifdb/alldenovo
```

```
# specify motifs files
motifs=$(cat motifs_${cnt}.txt)

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > header
rm header1 header2 header3

# tomtom header
printf '%s\n' 'Query_ID Target_ID Optimal_offset p-value E-value q-value \
Overlap Query_consensus Target_consensus Orientation' | \
tr ' ' '\t' > ${outfile}

# tomtom information
tomtom=/home/wa3j/meme-5.0.3/bin/tomtom
motifdb=~/TFdb/TFreduced/TFmotifs.txt
eval=66.93 # p < 0.001
minoverlap=5

# loop analysis
for ii in ${motifs}
do

# get the motif file
echo MOTIF ${ii} > line1
echo letter-probability matrix: > line2
cat header line1 line2 ${mdir}/${ii}.txt > meme.txt

# implement tomtom
${tomtom} -dist pearson -no-ssc -evalue -thresh ${eval} -m ${ii} \
-min-overlap ${minoverlap} meme.txt ${motifdb}

# store data
cd tomtom_out
mv tomtom.tsv ..
cd ..
rm -r tomtom_out
tail -n +2 tomtom.tsv | head -n -4 > new
mv ${outfile} tmp
cat tmp new > ${outfile}

# remove extras
rm tomtom.tsv new tmp
rm line1 line2 meme.txt
done
cp ${outfile} ..
```

We generate a file containing all TOMTOM results as follows.

```
####################################################
## generate summary file with all tomtom results
####################################################

# tomtom data and header
```

```
outfile=tomtomDenovoTFall.txt
printf '%s\n' 'Query_ID Target_ID Optimal_offset p-value E-value q-value \
Overlap Query_consensus Target_consensus Orientation' | \
tr ' ' '\t' > ${outfile}

# loop through all results and combine data
for ii in *_denovo.txt
do
tail -n +2 ${ii} > new
mv ${outfile} tmp
cat tmp new > ${outfile}
rm tmp new
done
```

Next we select the top TF match for each *de novo* motif, based on the p-values of association between PWMs, using R.

```
library(dplyr)
fname = "tomtomDenovoTFall.txt"
dat0 = read.table(fname,header=T,stringsAsFactors=F,sep="\t")

# filter out poorly matched de novo motifs
thresh = 0.001
dat1 = dat0 %>% filter(p.value < thresh)
denovo.all = unique(dat0$Query_ID)
denovo.flt = unique(dat1$Query_ID)
unmatched = denovo.all[!(denovo.all %in% denovo.flt)]

# loop through each de novo motif and get the top TF match
filtered = c()
for(ii in 1:length(denovo.flt)){
   mot = denovo.flt[ii]
   ttdat = dat1[dat1$Query_ID == mot,]
   ttdat = ttdat[with(ttdat, order(p.value,Target_ID)),]
   filtered = rbind(filtered, ttdat[1,])
}

# factor PWMs for denovo motifs
length(unique(filtered$Target_ID))
hist(-log10(filtered$p.value))
max(filtered$p.value)

# output information for downstream analysis
fname = "TFfromDENOVO.txt"
out = unique(filtered$Target_ID)
write.table(out,fname,row.names=F,col.names=F,quote=F,sep="\t")

fname = "DENOVOunmatchedtoTF001.txt"
out = unmatched
write.table(out,fname,row.names=F,col.names=F,quote=F,sep="\t")
```

6.4 Identification of similar TF motifs

The next step in the analysis pipeline is to identify representative TFs from clusters of similar TFs. We identify TF clusters by matching each of the previously identified TF motifs (i.e., those matched to *de novo* motifs) to all other TF motifs associated with *de novo* motifs using TOMTOM. We then cluster the motifs based on the TOMTOM data based on a previously described approach from the Guertin lab (Liu *et al.,* 2017). To select representative motifs out of motif clusters, we average the cluster TF PWMs as described below. The code for this analysis can be found in allTFs_eachVersusAll.sh. We parallelize the analysis using a SLURM array job. The first step is to separate to the TF PWMs of interest into a set of 50 motif lists.

```
#####################################################
## generate motif text files for array implementation
#####################################################

dir=~/tomtomTFvsTF
cd $dir

# read motif key into R
library(dplyr)
motifs = read.table("TFfromDENOVO.txt",header=F,stringsAsFactors=F)[,1]

# R code to generate a set of motif lists
# each list will be processed with an individual job
narray = 49
nmotif = length(motifs)
motif.per = floor( nmotif / narray )
for(ii in 1:narray){
   ind1 = (ii-1) * motif.per + 1
   ind2 = ind1 + motif.per - 1
   ind = ind1:ind2
   out = motifs[ind]
   fname = paste0("motifs_",ii,".txt")
   write.table(out,fname,col.names=F,row.names=F,quote=F,sep="")
   if(ii == narray){
      ind = (ind2+1):nmotif
      ii = ii + 1
      out = motifs[ind]
      fname = paste0("motifs_",ii,".txt")
      write.table(out,fname,col.names=F,row.names=F,quote=F,sep="")
   }
}
```

Here is the main file for the SLURM analysis.

```
###############################################################
## slurm script, tomtom_main.sh
###############################################################

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=~/tomtomTFvsTF
cd ${dir}

# sbatch --array=1-50 tomtom_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/tomtom_run.sh ${cnt}
```

Here is the file with the implementation details for running `TOMTOM`.

```
##############################################################
## tomtom run script, tomtom_run.sh
## Run tomtom for TF motifs against TF motifs
##############################################################

#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=~/tomtomTFvsTF
cd ${dir}

# subdirectory for specific analyses
mkdir condit_id_${cnt}
cp motifs_${cnt}.txt condit_id_${cnt}
cd condit_id_${cnt}

# isolate all motif identifiers
motifs=$(cat motifs_${cnt}.txt)

# output
outfile=tomtom${cnt}.txt

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > header
rm header1 header2 header3

# tomtom header
printf '%s\n' 'Query_ID Target_ID Optimal_offset p-value E-value q-value \
Overlap Query_consensus Target_consensus Orientation' | \
 tr ' ' '\t' > ${outfile}

# tomtom information
motiffile=~/TFdb/TFreduced/TFmotifs.txt
motifdb=~/TFdb/TFreduced/TFmotifs.txt
tomtom=/home/wa3j/meme-5.0.3/bin/tomtom
eval=66.93 # p < 0.001 for orig run, filter in R
minoverlap=5

# loop through motifs and implement tomtom
for ii in ${motifs}
do

# get meme format for the motif
linenum=$(cat ${motiffile} | grep -n -w ${ii})
linenumID=$(echo ${linenum} | awk -F":" '{print $1}')
motif=$(echo ${linenum} | awk -F":" '{print $2}' | \
awk -F"MOTIF " '{print $2}')
motif=$(echo ${motif} | awk -F" " '{print $1}')
linenumWM=$(expr ${linenumID} + 2)
tail -n +${linenumWM} ${motiffile} > part
```

```
lineend=$(grep -E --line-number --with-filename '^$' part | \
 head -1 | awk -F":" '{print $2}')
lineend=$(expr ${lineend} + 1)
tail -n +${linenumID} ${motiffile} | head -${lineend} > PWM
cat header PWM > meme.txt
rm part PWM

# implement tomtom
${tomtom} -dist pearson -no-ssc -evalue -thresh ${eval} -m ${motif} \
-min-overlap ${minoverlap} meme.txt ${motifdb}
rm meme.txt

# aggregate tomtom outputs
cd tomtom_out
mv tomtom.tsv ..
cd ..
rm -r tomtom_out
tail -n +2 tomtom.tsv | head -n -4 > new
mv ${outfile} tmp
cat tmp new > ${outfile}
rm tomtom.tsv new tmp
cp ${outfile} ..

done
```

The analysis can be run with the following commands.

```
chmod u+x *.sh
sbatch --array=1-50 tomtom_main.sh
```

Next we cluster the TF motifs based on the `TOMTOM results` using the igraph package for R as described previously (Csardi and Nepusz, 2006; Liu *et al.*, 2017). The following R script can be found in motif_communities_TF.R. Key functions for our analyses can be found in motif_clust_functions.R. The edge weight indicating the degree of similarity between two TF PWMs is taken as the -log10(p-value), based on the `TOMTOM` p-values. For these analyses, we considered -log10(p-value) $>$ 6 as a connectivity threshold, and TFs that were not connected to at least one other TF at this edge weight threshold were considered unmatched. The unmatched TFs were condered as isolated clusters and were not included in the motif cluster analysis.

```
library(NMF)
library(dplyr)
library(igraph)
library(stringr)
library(randomcoloR)
library(RColorBrewer)

###############################################################
## import data and generate three column graph format
###############################################################

# basic tomtom data and pval filter
fname = "tomtomTFvsTF.txt"
dat0 = read.table(fname,header=T,stringsAsFactors=F)
facs = unique(dat0$Query_ID)
length(facs)
dat0 = dat0[dat0$Target_ID %in% facs,]
length(unique(dat0$Target_ID))
dat0 = dat0 %>% filter(p.value<0.001)
```

```
# upload mapping between TF motif and database names
pathToFile <- paste0(getwd(),"/motif.id.key.txt")
f <- file(pathToFile, "rb")
rawText <- readLines(f)
close(f)
motifs0 <- sapply(rawText, str_split, "\t")
motifs.raw = unlist(motifs0)[-c(1)] %>% unname
ids.new = sapply(motifs.raw,function(x){
     strsplit(x," ",fixed=TRUE)[[1]][1]}) %>% unname
ids.orig = sapply(motifs.raw,function(x){
  id = strsplit(x," ",fixed=TRUE)[[1]]
  out = paste(id[2:length(id)], collapse=" ")
}) %>% unlist
id.map = as.data.frame(cbind(ids.new,ids.orig),stringsAsFactors=F)
rownames(id.map) = 1:nrow(id.map)

# directory for database TF pwms
pwm.dir = paste0("/communities/TF")

###############################################################
## generate three column graph format
###############################################################

# remove self to self edges
ind.rem = which(dat0$Query_ID == dat0$Target_ID)
dat0 = dat0[-ind.rem,]

# generate edge data with -log10(pval)
threecol = dat0 %>% select(Query_ID, Target_ID, p.value)
names(threecol) = c('from','to','pval')
minp = threecol %>% filter(pval>0) %>% select(pval) %>% min
threecol$pval[threecol$pval == 0] = minp
weight = -log10(threecol$pval)
threecol$weight = weight

# filter weights
threecol = threecol %>% filter(weight>6)

# quantify tomtom motifs, document unmatched
tomtom.uniq = unique(c(threecol$from, threecol$to))
length(tomtom.uniq)
matched = tomtom.uniq
unmatched = facs[!(facs %in% matched)]

# weight distribution
hist(threecol$weight)

###############################################################
## create the graph format and detect communities
###############################################################

# create the graph
g = graph.data.frame(threecol, directed=F)
g = simplify(g)

# commnity detection
comm.g = fastgreedy.community(g)
```

```
# community annotation
ann.comm = sapply(unique(comm.g$membership),function(x){
     length(which(comm.g$membership==x))})
ann.comm = cbind(unique(comm.g$membership),ann.comm) %>%
     as.data.frame(stringsAsFactors=F)
names(ann.comm) = c("comm","count")
ann.comm = ann.comm[order(ann.comm$count,decreasing=T),]
```

For this analysis, we also sub-cluster large clusters iteratively until all clusters have a maximal number of TF equal to the value of max.clust, under the constrain that max.iter is the maximal number of sub-clustering iterations performed (see subcluster() in motif_clust_functions.R).

```
###############################################################
## subcluster large communities
###############################################################

# generate subclusters
subclust.dat = subcluster(clusters=comm.g, graph=threecol,
                  max.clust=10, max.iter=10, change.thresh=3)
```

We visualize the clustering results as follows. Note that we scale the relative vertex sizes and edge weights arbitrarily to facilitate visual interpretation.

```
###############################################################
## plot graph of all communities
###############################################################

# plot with motif colors
cols = distinctColorPalette(length(unique(ann.comm$comm)))

# comunity cluster plot function
comm.plt = function(g=NULL,layout=NULL,vertex.color=NULL,
                    edge.width=NULL,vertex.size=NULL){
  #pdf("clust.pdf")
  plot(g,layout=layout, rescale=F, vertex.label.cex=.5,
       xlim=range(layout[,1]), ylim=range(layout[,2]),
       edge.width=edge.width, vertex.size=vertex.size,
       edge.curved=T, vertex.label=NA,
       vertex.color=vertex.color, edge.color="black",
       margin=0, asp=0)
  #dev.off()
}

# annotate community colors
vertex.color = cols[comm.g$membership]
names(vertex.color) = comm.g$membership

# layout.fruchterman.reingold
layout = layout.fruchterman.reingold(g)
layout = layout.norm(layout,-1,1,-1,1)
vertex.size = (degree(g,mode='out') + 10)/3
edge.width = E(g)$weight/4

pdf("clusters.pdf", width=6); par(mfrow=c(2,1))
comm.plt(g, layout, vertex.color, edge.width, vertex.size)
dev.off()

# annotate communities
```

```
length(comm.g$membership)
length(comm.g$names)
comm.ann = list()
for(comm in ann.comm$comm){
  ind.comm = which(comm.g$membership == comm)
  tfs = comm.g$names[ind.comm]
  color = vertex.color[ind.comm[1]]
  comm.ann[[paste0("community_",comm)]] = list(color=color, tfs=tfs)
}
```

We output TF clustering results, and un-clustered TFs, for downstream analysis.

```
##############################################################
## output community annotation
## incorporate subclusters and unmatched motifs
##############################################################

# include un-matched motifs as additional clusters
motif.community = subclust.dat
ncom = max(subclust.dat$clust)
newcomms = c( (ncom+1) : (ncom+length(unmatched)) )
new = cbind(newcomms, unmatched) %>% as.data.frame(stringsAsFactors=F)
names(new) = names(motif.community)
motif.community = rbind(motif.community, new)
motif.community$clust = data.matrix(motif.community$clust) %>% as.numeric

# add original TF ids
inds = sapply(motif.community$motif,function(x){which(id.map$ids.new==x)})
ids.orig = id.map$ids.orig[inds]
motif.community = motif.community %>% mutate(ids.orig = ids.orig)

# checks
length(unique(motif.community$clust))
length(unique(motif.community$motif))

# generate motif list
motif.clusters = list()
for(ii in 1:length(unique(motif.community$clust))){
  clust = unique(motif.community$clust)[ii]
  ind = which(motif.community$clust == clust)
  namen = paste0("community_",clust)
  motif.clusters[[namen]] = motif.community$motif[ind]
}

# generate cluster TF 'spreadsheet' format
maxlen = lapply(motif.clusters,function(x){length(x)}) %>% unlist %>% max
datTF = as.data.frame(matrix(0,maxlen,length(motif.clusters)))
dat = as.data.frame(matrix(0,maxlen,length(motif.clusters)))
for(ii in 1:ncol(dat)){
  len = length(motif.clusters[[ii]])
  dat[1:len,ii] = motif.clusters[[ii]]
  inds = sapply(motif.clusters[[ii]],function(x){which(id.map$ids.new==x)})
  datTF[1:len,ii] = id.map$ids.orig[inds]
}
names(dat) = names(datTF) = names(motif.clusters)
write.table(dat,"TFclusters.txt",col.names=T,row.names=F,sep="\t")
```

Next we implement an analysis framework for averaging clustered TF motifs. For finding the average of a cluster, we first identify the most connected cluster motif as that motif with the greatest number of connections within the cluster. Connections are defined by the -log10 of the TOMTOM p-values (see get.max.connect.motifs2()). Next we average all cluster PWMs by aligning each the the most connected motif using denovo.mean.plot.TF(). This function first identifies all motifs directly connected to the most connected motif and determines the respective positional offsets, considering the reverse complement motifs as well. Our find.offset() function defines offsets based on both the sum of site information for query motif sites with information > 1, or based on the sum of absolute differences between the PWMs in question (i.e., $\sum_i \sum_j |a_{ij} - b_{ij}|$ for PWMs a and b where $i \in \{A, C, G, T\}$ and j is the number of sites). We calculate the information at a given site s as the dot product between site frequencies and the log ot these frequencies relative to the background: $I_s = f_s \cdot (log_2(f_s) - log_2(bkg))$, where f_s is a vector of four nucleotide frequencies at a given site and bkg is a vector of background frequencies for the four nucleotides (e.g., we use bkg = 0,25, 0.25, 0.25, 0.25). For our analysis, we defined offsets as the offset with the minimum absolute summed differences. All PWMs are padded with low information content sites based on a user-defined background (bkg) of a specified length (pad.len). The offsets for all PWMs, relative to the central PWM, are documented. After finding the offsets for PWMs not directly connected to the center, we iterative go through the rest of the PWMs until each has been associated with another PWM with an identified offset relative to the center. Once all offsets have been identified, as well as whether the reverse complement of a given PWM should be considered to correspond to the central motif, we arithmetically average the aligned/padded PWMs. We then remove consecutive sites at the flanks of the averages PWM for which the information is less than site.thr $= 10^{-1}$. Code details can be found in motif_clust_functions.R. A similar approach has been implemented previously (Spivak and Stormo, 2012). We compute the TF motif cluster avaerages and plot the respective motif signatures for the central and average motifs as follows.

```
##############################################################
## find avaerage motif PWM for each cluster
##############################################################

require(ggplot2)
require(ggseqlogo)
library(gridExtra)
library(dplyr)
library(seqLogo)
library(prodlim)
library(lattice)

# find the most connected motif for each community based on connection sum
names(motif.community)[1] = "community"
max.connect.motifs = get.max.connect.motifs2(
     motif.community=motif.community,
     graph=threecol)

# average motif analysis
site.thr = 0.1 # threshold for keeping a margin site in the pwm
pad.len = 60
max.offset = 10
bkg = c(0.25, 0.25, 0.25, 0.25)
pltdat = denovo.mean.plot.TF(dat=dat, pwm.dir=pwm.dir, tomtom=dat0,
                max.connect.motifs=max.connect.motifs,bkg=bkg,
                pad.len=pad.len,max.offset=max.offset)

# plot the motif information
pdf("TFsubclust10Filt1e6_new.pdf", onefile = TRUE, height=9, width=4.5)
marrangeGrob(grobs=pltdat$plt, nrow=6, ncol=1, top=NULL)
dev.off()
```

We then filter the TF cluster average PWMs based on the following exclusion criteria:

1. The center and average motif have < 3 matching sites with information > 1 (n.info.match.sites =3, info.thresh =1)

2. The average PWM contains 4 consecutive sites with an identical base (nsite.in.row = 4)

3. The central PWM contains more than 12 sites with information greater than 1.6 (n.sties.info.thresh = 12, n.sites.info = 1.6)

After removing TF clusters/averages that satisfy the exclusion criteria, we use `TOMTOM` to evaluate the similarities amongst TF cluster averages. We chack each average PWM against all others as in previous cases. First we ouput individual files with average PWMs.

```
##############################################################
## output PWMs for tomtom
##############################################################

length(flt.pwm)

pwm.out = flt.pwm
cdir = getwd()
pwm.dir = paste0(cdir,"/center_av_pwm")
setwd(pwm.dir)

for(ii in 1:length(pwm.out)){
  tf = names(pwm.out)[ii]
  mat = t( pwm.out[[ii]] )
  fname = paste0(tf,".txt")
  write.table(mat,fname,sep="\t",quote=F,col.names=F,row.names=F)
}
```

As described above, we submit a parallelized SLURM array job to complete the analysis. The code can be found in TFaverage_eachVersusAll.sh. We omit the code here because the details are similar to those shown above. Following the generation of the `TOMTOM` results, the analysis continues in R as detailed in motif_communities_TF.R.

The next step of our TF motif analysis is to cluster the set of average motifs based on their similarities, as defined by `TOMTOM` p-values of association (-log10 transformed).

```
##############################################################
## cluster average TFs based on tomtom results
##############################################################

# load("TFclusteravs.RData")

library(NMF)
library(dplyr)
library(igraph)
library(stringr)
library(randomcoloR)
library(RColorBrewer)

# import tomtom data and pval filter
fname = "tomtomTFAVvsTFAV.txt"
tomtomAv0 = read.table(fname,header=T,stringsAsFactors=F)
tomtomAv0 = tomtomAv0[duplicated(tomtomAv0)==F,]
tomtomAv0 = tomtomAv0 %>% filter(p.value<0.001)
length(unique(tomtomAv0$Query_ID))
length(unique(tomtomAv0$Target_ID))

# remove self to self edges
ind.rem = which(tomtomAv0$Query_ID == tomtomAv0$Target_ID)
```

```
tomtomAv0 = tomtomAv0[-ind.rem,]

# generate edge data with -log10(pval)
threecolAv = tomtomAv0 %>% select(Query_ID, Target_ID, p.value)
names(threecolAv) = c('from','to','pval')
minp = threecolAv %>% filter(pval>0) %>% select(pval) %>% min
threecolAv$pval[threecolAv$pval == 0] = minp
weight = -log10(threecolAv$pval)
threecolAv$weight = weight

# filter weights
threecolAv = threecolAv %>% filter(weight>6)

# create the graph
gAv = graph.data.frame(threecolAv, directed=F)
gAv = simplify(gAv)

# commnity detection
comm.gAv = fastgreedy.community(gAv)

# annotate community colors
cols = distinctColorPalette(length(unique(comm.gAv$membership)))
vertex.color = cols[comm.gAv$membership]
names(vertex.color) = comm.gAv$membership

# layout.fruchterman.reingold
layout = layout.fruchterman.reingold(g)
layout = layout.norm(layout,-1,1,-1,1)
vertex.size = (degree(gAv,mode='out') + 10)/3
edge.width = E(gAv)$weight/4
comm.plt(gAv, layout, vertex.color, edge.width, vertex.size)
```

For each of the final resulting clusters of averaged PWMs, we select the PWM with the highest total amount of information as a representative. We plot the motif signatures for all final average PWMs, as well as previously identified ‘isolated cluster’ motifs.

```
##############################################################
## for each cluster, pick the average with the highest information
## integrate with unclustered TFs to get the final set of TFs
##############################################################

require(ggplot2)
require(ggseqlogo)
library(gridExtra)
library(dplyr)
library(seqLogo)
library(prodlim)
library(lattice)

aa.flt.pwm = names(flt.pwm)
clustered = comm.gAv$names
unclusted = aa.flt.pwm[!(aa.flt.pwm %in% clustered)]
length(aa.flt.pwm) == length(clustered) + length(unclusted)
newclust = comm.gAv$membership %>% unique
nclust = length(newclust)

# loop through clusters, pick the element with highest info
clustTFs = c()
```

```
for(ii in newclust){

  # get cluster motifs
  ind = which(comm.gAv$membership == ii)
  motifs = comm.gAv$names[ind]

  # get the cluster pwms
  clust.pwm = list()
  for(jj in motifs){
    fname = paste0(pwm.dir,"/",jj,".txt")
    mat = read.table(fname,sep="\t",
       header=F,stringsAsFactors=F,fill=T) %>% t
    nas = apply(mat,1,function(x){all(is.na(x))})
    ind = which(nas==TRUE)
    if(length(ind)>0){mat = mat[-ind,]}
    rownames(mat) = c("A","C","G","T")
    colnames(mat) = c(1:ncol(mat))
    clust.pwm[[jj]] = mat
  }

  # get summed info for each pwm
  infos = lapply(clust.pwm,function(x){
    pwm = add.pseudocount(PWM=x, cnt=1e-6)
    info.tf = apply(pwm,2,function(x){x %*% (log2(x)-log2(bkg))})
    return(mean(info.tf))
  }) %>% unlist

  out = names(infos)[infos == max(infos)]
  clustTFs = c(clustTFs, out)
}

# aggregate filtered/clustered TF pwms for plotting
length(flt.pwm)
length(clustered)
length(unclusted)
length(clustTFs)
flt.pwm.clust = list()
for(ii in unclusted){
  flt.pwm.clust[[ii]]=flt.pwm[[ii]]
}
for(ii in clustTFs){
  flt.pwm.clust[[ii]]=flt.pwm[[ii]]
}
length(flt.pwm.clust) # 344

# plot filtered/clustered TFs
filt.clust.plts = list()
p=1
for(ii in names(flt.pwm.clust)){
  id = motif.community$ids.orig[motif.community$motif==ii]
  mat = flt.pwm.clust[[ii]]
  filt.clust.plts[[p]] = logo.plt(pwm=mat, title=id)
  p=p+1
}
pdf("TF_filteredClust344.pdf", onefile = TRUE, height=9, width=4.5)
marrangeGrob(grobs=filt.clust.plts, nrow=6, ncol=1, top=NULL)
dev.off()
```

6.5 Filtering TF motifs based on coorespondance to *de novo* motifs

The final step of our *de novo* motif analysis is to filter the final set of TF motifs (i.e., average PWMs) based on their coorespondance to *de novo* motifs. We remove all TF motifs that were not derived from two or more distinct instances of identified *de novo* motifs. First we load previously acquired data and distinguish cluster TF motifs from singlet motifs. NOTE TO UPDATE THIS CODE.

```
###############################################################
## filter TF pwm set based on number of de novo motifs matching
###############################################################

# import data
fname = "denovomotifINFO.txt"
denovo0 = read.table(fname,header=F,stringsAsFactors=F,sep="\t")
names(denovo0) = c("motif","comps","ncomps")

# get mapping between de novo motifs and database TFs
# see tomtom_denovo_vs_db_array.sh
fname = "tomtomDenovoTFall.txt"
tomtom_DETF = read.table(fname,header=T,stringsAsFactors=F,sep="\t")
tomtom_DETF = tomtom_DETF %>% filter(p.value < 0.001)
consensus = sapply(tomtom_DETF$Query_ID,function(x)strsplit(x,"_")[[1]][1])
tomtom_DETF = tomtom_DETF %>% mutate(consensus=consensus)
denovo.flt = unique(tomtom_DETF$Query_ID) # 2611

# loop through each de novo motif and get the top TF match
filtered.denovo = c()
for(ii in 1:length(denovo.flt)){
  mot = denovo.flt[ii]
  ttdat = tomtom_DETF[tomtom_DETF$Query_ID == mot,]
  ttdat = ttdat[with(ttdat, order(p.value,Target_ID)),]
  filtered.denovo = rbind(filtered.denovo, ttdat[1,])
}
length(unique(filtered.denovo$Query_ID))
length(unique(filtered.denovo$Target_ID))

# initial annotation for TF clusters
# motif.community
motif.community0 = motif.community

# remove filtered TF clusters
motif.community1 = motif.community0
for(ii in bad.facs){
  comm = motif.community1$community[motif.community1$motif==ii]
  ind = which(motif.community1$community == comm)
  motif.community1 = motif.community1[-ind,]
}
length(motif.community1$community %>% unique)

# remove all of the unclustered clusters
motif.community2 = motif.community1
clustclust = c()
for(ii in clustered){
  comm = motif.community2$community[which(motif.community2$motif==ii)]
  ind = which(motif.community2$community == comm)
  motif.community2 = motif.community2[-ind,]
}
length(motif.community2$community %>% unique)
```

```
# add back cluster representatives
motif.community3 = motif.community2
for(comm in newclust){

  # get new cluster motifs
  ind = which(comm.gAv$membership == comm)
  motifs = comm.gAv$names[ind]

  # aggregate old clusters
  oldcoms = sapply(motifs,function(x){
      motif.community1$community[motif.community1$motif==x]})
  indcoms = sapply(oldcoms,function(x){
      which(motif.community1$community==x)}) %>% unlist
  newcom = motif.community1[indcoms,]
  newcom$community = min(oldcoms)
  motif.community3 = rbind(motif.community3, newcom)
}
length(motif.community3$community %>% unique)
```

We then determine the number of distinct *de novo* motifs associated with each TF cluster/singles and remove those without multiple *de novo* motif associations.

```
#### get the num of de novo hits for each cluster ###
motif.community.summary = c()
for(ii in 1:length(unique(motif.community3$community))){
  comm = unique(motif.community3$community)[ii]
  tf.motifs = motif.community3$motif[motif.community3$community == comm]
  ind.de = sapply(tf.motifs,function(x)
      which(filtered.denovo$Target_ID==x)) %>% unlist
  de.motifs = filtered.denovo$consensus[ind.de] %>% unique
  comps = sapply(de.motifs,function(x){
    all = denovo0$comps[denovo0$motif==x]
    out = strsplit(all,",")[[1]]
    return(out)
  }) %>% unlist %>% unique
  compout = paste(comps,collapse=",")
  center = names(flt.pwm.clust)[names(flt.pwm.clust) %in% tf.motifs]
  id = motif.community3$ids.orig[motif.community3$motif == center]
  new = c(center, id, length(comps), compout)
  motif.community.summary = rbind(motif.community.summary, new)
}
motif.community.summary = as.data.frame(motif.community.summary,
      stringsAsFactors=F)
names(motif.community.summary) = c("id","tf","ncomps","comps")
rownames(motif.community.summary) = 1:nrow(motif.community.summary)

# filter TFs based on those identified by multiple comparisons
onecomp = motif.community.summary %>% filter(ncomps==1)
motif.community.summary = motif.community.summary %>% filter(ncomps>1)
flt.pwm.clust.final = list()
for(ii in 1:nrow(motif.community.summary)){
  motif = motif.community.summary$id[ii]
  ind = which(names(flt.pwm.clust) == motif)
  flt.pwm.clust.final[[motif]] = flt.pwm.clust[[ind]]
}
```

Finally, we plot the resulting set of TF motif signatures, output annotation information for the motifs and clusters, and write PWM files for subsequent analysis with `FIMO`.

```
# plot filtered/clustered TFs
final.plts = list()
p=1
for(ii in names(flt.pwm.clust.final)){
  id = motif.community4$ids.orig[motif.community4$motif==ii]
  mat = flt.pwm.clust[[ii]]
  final.plts[[p]] = logo.plt(pwm=mat, title=id)
  p=p+1
}
pdf("TF_final216.pdf", onefile = TRUE, height=9, width=4.5)
marrangeGrob(grobs=final.plts, nrow=6, ncol=1, top=NULL)
dev.off()

# generate speadsheet format for clusters
motif.clusters.id = list()
motif.clusters.tf = list()
for(ii in 1:length(unique(motif.community4$community))){
  clust = unique(motif.community4$community)[ii]
  ind = which(motif.community4$community == clust)
  namen = paste0("community_",clust)
  motif.clusters.id[[namen]] = motif.community4$motif[ind]
  motif.clusters.tf[[namen]] = motif.community4$ids.orig[ind]
}
maxlen = lapply(motif.clusters.id,function(x)length(x)) %>% unlist %>% max
datTF = as.data.frame(matrix(0,maxlen,length(motif.clusters.tf)))
datID = as.data.frame(matrix(0,maxlen,length(motif.clusters.id)))
for(ii in 1:ncol(datID)){
  len = length(motif.clusters.id[[ii]])
  datID[1:len,ii] = motif.clusters.id[[ii]]
  datTF[1:len,ii] = motif.clusters.tf[[ii]]
}
names(datID) = names(datTF) = names(motif.clusters.tf)
datID = datID[,order(apply(datID,2,function(x)length(which(x!=0))),
   decreasing=T)]
datTF = datTF[,order(apply(datTF,2,function(x)length(which(x!=0))),
   decreasing=T)]
#write.table(dat,"TFclusters.txt",col.names=T,row.names=F,sep="\t")

# generate pwm files for the 216 TF motifs of interest
pwm.out = flt.pwm.clust.final
cdir = getwd()
pwm.dir = paste0(cdir,"/TF216")
setwd(pwm.dir)
for(ii in 1:length(pwm.out)){
  tf = names(pwm.out)[ii]
  mat = t( pwm.out[[ii]] )
  fname = paste0(tf,".txt")
  write.table(mat,fname,sep="\t",quote=F,col.names=F,row.names=F)
}
setwd(cdir)
```

7 Transcription factor motif enrichment analysis

Here we describe the methods for identifying DNA motifs that are enriched in dynamic regions of open chromatin using `FIMO` from the MEME suite(). See above for the methods for defining open chromatin peaks from ATAC-seq data **??**, identifying differentially expressed peaks **??**, and indentifying transcription factor (TF) motifs of interest based on *de novo* motif analysis **??**. UPDATE BASED ON

MOST RECENT CODE and RE-RUN ANALYSES.

7.1 FIMO analysis of TF motifs in the murine genome

Here we use R to establish 216 .txt files with PWM identifiers to facilitate the submission of a par-ralelized job submission to a SLURM system (array job, e.g., see `https://arcs.virginia.edu/slurm`). The code for the following analyses can be found in TFfimo.sh.

```
#####################################################
## generate motif lists for array analysis
#####################################################

# get ids
ls *.txt | awk -F".txt" '{print $1}' > motif.ids

# read motif key into R
library(dplyr)
motifs = read.table("motif.ids",header=F,stringsAsFactors=F)[,1]

# R code to generate a set of motif lists
# each list will be processed with an individual job
narray = 216
nmotif = length(motifs)
motif.per = floor( nmotif / narray )
for(ii in 1:narray){
     ind1 = (ii-1) * motif.per + 1
     ind2 = ind1 + motif.per - 1
     ind = ind1:ind2
     out = motifs[ind]
     fname = paste0("motifs_",ii,".txt")
     write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
     if(ii == narray & ind2 != nmotif){
        ind = (ind2+1):nmotif
        ii = ii + 1
        out = motifs[ind]
        fname = paste0("motifs_",ii,".txt")
        write.table(out,fname,col.names=F,row.names=F,quote=F,sep="\t")
     }
}
```

We generate a set of PWM files with formats compatible with the FIMO analysis.

NEW ADDITION: For this analysis, we evaluated the frequencies of each nucleotide in the genome (see), as follows, using the murine genome (mm10 build) .fa file (ADD CODE).

```
# shell code
awk -F a '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 345823800
awk -F c '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 253024277
awk -F g '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 253137432
awk -F t '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 346530271

awk -F A '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 427456346
awk -F C '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 299623606
awk -F G '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 299552717
awk -F T '!/^>"/ {cnt+=NF-1}END{print cnt}' mm10.fa # 427635170
```

```
# R code
a = 345823800
c = 253024277
g = 253137432
t = 346530271

A = 427456346
C = 299623606
G = 299552717
T = 427635170

# repeat
a / (a+g+c+t) # 0.2885434
c / (a+g+c+t) # 0.2111147
g / (a+g+c+t) # 0.2112091
t / (a+g+c+t) # 0.2891328

# non
A / (A+C+T+G) # 0.2939323
C / (A+C+T+G) # 0.2060306
G / (A+C+T+G) # 0.2059818
T / (A+C+T+G) # 0.2940553

# both
(A+a) / (A+C+T+G+a+g+c+t) # 0.2914976
(C+c) / (A+C+T+G+a+g+c+t) # 0.2083275
(G+g) / (A+C+T+G+a+g+c+t) # 0.2083435
(T+t) / (A+C+T+G+a+g+c+t) # 0.2918314
```

Based on this analysis, we applied the following background in our `FIMO` analyses: A 0.29, C 0.21, G 0.21, T 0.29 (MODIFY FIMO CODE ACCORDINGLY). We generate a set of PWM files with formats compatible with the `FIMO` analysis as follows.

```
####################################################
## generate meme format pwm files
####################################################

pwmdir=~/TFfimo
motifs=$(ls | awk -F".txt" '{print $1}')

# create header file (header)
printf '%s\n' 'MEME version 5' '' 'ALPHABET= ACGT' '' 'strands: + -'  '' \
> header1
printf '%s\n' 'Background letter frequencies (from uniform background):' \
> header2
printf '%s\n' 'A 0.25000 C 0.25000 G 0.25000 T 0.25000' '' > header3
cat header1 header2 header3 > header
rm header1 header2 header3

# loop through all motifs and create meme files
for ii in ${motifs}
do
cat ${pwmdir}/${ii}.txt > pwm
echo MOTIF ${ii} > line1
echo letter-probability matrix: > line2
cat header line1 line2 pwm > ${ii}.txt
rm pwm line1 line2
done
```

Here is the main file for the SLURM array job submission. The code can be saved in a file named fimo_slurm.sh.

```
###### slurm script (fimo_slurm.sh)

#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH -p largemem
#SBATCH --time=96:00:00
#SBATCH --mem=128G
#SBATCH --partition=standard

dir=~/motifs/TFfimo
cd ${dir}

# sbatch --array=1-216 fimo_slurm.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/slurm_fimorun.sh ${cnt}
```

The main code for the parallelized `FIMO` analysis can be included in a file named slurm_fimorun.sh. For this analysis we use a permissive p-value threshold of 0.01 and we apply stringent significance filtering during downstream analysis as described below.

```
###### analysis script (slurm_fimorun.sh)

#!/bin/bash

dir=~/motifs/TFfimo
cd ${dir}

# threshold
thresh=0.01

# count indicator
cnt="$1"

# navigate to a new directory for each factor
mkdir condit_id_${cnt}
cp -t condit_id_${cnt} motifs_${cnt}.txt
cd condit_id_${cnt}

# loop through each unmatched motif
motif=$(cat motifs_${cnt}.txt)

# get the appropriate pwm file
memedir=${dir}/TF216
for ii in ${motif}
do
cp ${memedir}/${ii}.txt ${dir}/condit_id_${cnt}
done

# modules, paths
module load bedtools
module load ucsc-tools
fimo=/home/wa3j/meme-5.0.3/bin/fimo

# fimo params
mm10=~/genomes/mm10/mm10.fa
```

```
max=100000000
outdir=out

for ii in ${motif}
do

# fimo header - initiate results file
outfile=fimo_${ii}.txt
printf '%s\n' 'motif_id chr start end' | tr ' ' '\t' > ${outfile}

# impliment fimo
${fimo} -thresh ${thresh} --max-stored-scores ${max} \
-motif ${ii} -o ${outdir} ${ii}.txt ${mm10}

# process fimo results
cd ${outdir}
cp fimo.tsv ..
cd ..
cat fimo.tsv | grep -v sequence_name | cut -f1,3-5 |  \
sort -k1,1 -k2,2n | awk '!seen[$0]++' |  \
tail -n +5 > fimo0.bed
mv ${outfile} tmp
cat tmp fimo0.bed > ${outfile}
rm fimo0.bed

done
```

The analysis can be implemented as follows.

```
chmod u+x *.sh
sbatch --array=1-216 org_main.sh
```

7.2 Filtering the **FIMO** results

We filter the FIMO results for a given motif by specifying a number of motif occurances in the genome, identifying the p-value at which the specified number of occurences is observed, and taking all motif occurences at or below the identified p-value. For this analysis, we identified the p-values corresponding to 500,000 motif occurances. This number was empirically determined based on xyz analyses. We implement the analysis by submitting a SLURM array job. The main SLURM file was structured as follows.

```
####### org_main.sh #######
#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=96:00:00
#SBATCH --partition=standard

dir=~/motifs/TFfimo/res
cd ${dir}

# sbatch --array=1-216 org_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/org_run.sh ${cnt}
```

Here is the file for implementing the analysis in parallel for the FIMO results for each TF motif.

```
####### org_run.sh #######
#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=~/motifs/TFfimo/res
cd ${dir}

# param
nmatch=500000

# data directory
datdir=condit_id_${cnt}

# filter the data
cd ${datdir}
motif=$(cat motifs_*)
echo ${motif}
fname=fimo_${motif}.txt
pval=$(head -${nmatch} fimo.tsv | tail | tail -n +10 | cut -f8)
pval=$(expr ${pval})
nline=$(cut -f8 fimo.tsv | LC_ALL=C fgrep -n ${pval} | tail | \
tail -n +10 | awk -F":" '{print $1}')
head -${nline} fimo.tsv > ${fname}
mv ${fname} ..
cd ${dir}
```

We document the p-value at which the specified number of motif occurances is observed.

```
####################################################
## get the number of sites and highest pval for each motif
####################################################

dir=~/motifs/TFfimo/res/summary500k
cd ${dir}

# set space separated data file
echo motif nsites highp > fimo.summary

# loop through and collect information
for ii in *fimo_motif*
do
motif=$(echo ${ii} | awk -F"fimo_" '{print $2}' | awk -F".txt" '{print $1}')
nsites=$(wc -l ${ii} | awk -F" " '{print $1}')
highp=$(tail ${ii} | tail -n +10 | cut -f8)
echo ${motif} ${nsites} ${highp} > new
mv fimo.summary tmp
cat tmp new > fimo.summary
rm new tmp
done
```

We use R to plot the relation between between identified p-values corresponding to the specified number of sites and the total number of motif occurances at the identified p-values across all TF motifs.

```
# import data into R
dat = read.table("fimo.summary",header=T,sep=" ",stringsAsFactors=F)

# plot
plot(-log10(dat$highp), log10(dat$nsites)); abline(h=6.5)
plot(dat$highp, dat$nsites)
```

We use `Bedtools` and `UCSC tools` to generate .bed and .bigWig files for the the motif corrdinates identified as described above.

```
####################################################
## generate bw and bed files for fimo data
####################################################

# set id files for the array job
cnt=1
for ii in *fimo_motif*
do
echo ${ii} > array${cnt}
cnt=$(expr ${cnt} + 1)
done

# get mouse chromosome annotation
wget http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes

####### org_main.sh #######
#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=8:00:00
#SBATCH --partition=standard

dir=~/motifs/TFfimo/res/summary500k
cd ${dir}

# sbatch --array=1-215 org_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/org_run.sh ${cnt}

##########################
####### org_run.sh #######
#!/bin/bash

# chmod u+x *.sh

cnt="$1"

# modules
module load bedtools
module load ucsc-tools/3.7.4

dir=~/motifs/TFfimo/res/summary500k
cd ${dir}

# data directory
fimodat=$(cat array${cnt})
motifid=$(echo ${fimodat} | awk -F"fimo_" '{print $2}' | \
awk -F".txt" '{print $1}')
```

```
# data processing
tail -n +2 ${fimodat} | cut -f3-5 | sort -k1,1 -k2,2n | \
awk '!seen[$0]++' > fimo0_${motifid}.bed
bedtools merge -i fimo0_${motifid}.bed | \
awk '{OFS="\t";} {print $1, $2, $3, 1}' > fimo_${motifid}.bed
bedGraphToBigWig fimo_${motifid}.bed mm10.chrom.sizes fimo_${motifid}.bigWig
rm fimo0_${motifid}.bed
```

8 Evaluating motif enrichment in differentially expressed open chromatin regions

Given the corrdinates of TF motif occurances, out next task is to determine which motifs are enriched in dynamic regions of open chromatin. For this analysis we us the .bigWig files containing motif coordinates (see above). We also use previously generated data, including...REFERENCE SECTIONS ABOVE. ALSO, I THINK WE SHOULD AUGMENT THE ANALYSES TO CONSIDER MOTIF ENRICHMENT IN DREG REGIONS AND DYNAMIC PEAK CLUSTER REGIONS FOR COMPREHENSIVNESS AND SYMMETRY WITH THE DE NOVO MOTIF ANALYSES.

IN GENERAL, THIS WHOLE SECTION NEEDS TO BE REFINED AND UPDATED, ALL ROUGH DRAFT CODES ARE ON GIT HOWEVER.

8.1 `FIMO` analysis of TF motif enrichment in differentially expressed open chromatin

We load key data into R and save a .RData image with requesite information (Motif_in_peak.R) that will be used in a shell script that will be executed through a parallel SLURM job.

```
library(dplyr)
library(DESeq2)
library(ggplot2)
library(bigWig)
library(prodlim)
library(lattice)
library(stringr)

#############################################################
# load data from differential peak analysis
# load data from de novo motif analysis
#############################################################

# load annotation for all peaks (summits.bed)
# see empiricalSummits_20190325.R
load("ATACsummits_20190325.RData")
all.peaks = paste0(summits.bed[,1],":",summits.bed[,2],"-",summits.bed[,3])
all.summits = paste0(summits.bed[,1],":",summits.bed[,2],
                     "-",summits.bed[,3],",",summits.bed[,4])

# load differential peak data
# see atac_time_deg_20190214.R
load("pairwise.deg.RData")

# load annotation for all peaks (bed.map)
# see atac_time_deg_20190214.R
load("bed.map.RData")
all.peaks = paste0(bed.map[,1],":",bed.map[,2],"-",bed.map[,3])
```

```
# functions to load and plot atac data
code.dir = paste0("~/ATAC_time_DEG/peak_enrich")
source(paste0(code.dir,"/motifEnrich_functions.R"))

# load atac deseq2 data object with sizefactors, deseq_obj_preadip
# see atac_time_deg_20190214.R
load("atacDeseq.RData")
sizefac = sizeFactors(deseq_obj_preadip)

# upload mapping between TF motif and database names
pathToFile <- paste0(getwd(),"/motif.id.key.txt")
f <- file(pathToFile, "rb")
rawText <- readLines(f)
close(f)
motifs0 <- sapply(rawText, str_split, "\t")
motifs.raw = unlist(motifs0)[-c(1)] %>% unname
ids.new = sapply(motifs.raw,function(x){
  strsplit(x," ",fixed=TRUE)[[1]][1]}) %>% unname
ids.orig = sapply(motifs.raw,function(x){
  id = strsplit(x," ",fixed=TRUE)[[1]]
  out = paste(id[2:length(id)], collapse=" ")
}) %>% unlist
id.map = as.data.frame(cbind(ids.new,ids.orig),stringsAsFactors=F)
rownames(id.map) = 1:nrow(id.map)

############################################################
# analysis parameters
############################################################

# preadipogenesis times
t.order = cbind(c("t0","20min","40min","60min","2hr","3hr","4hr"),
                c(0,0.33,0.67,1,2,3,4)) %>%
  as.data.frame(stringsAsFactors=FALSE)
names(t.order) = c("condition","time")


# bigwig import params
file.prefix = "3T3_"
file.suffix = ".bigWig"
min.length = 1

# fdr threshold for filtering all fimo results per factor
fdr.thresh = 0.001
maxdiff.thresh = 10

# peak count threshold for considering optimal chi-sq result
pk.cnt.thresh = 100

# parameters for designating dynamic peaks
sig.thresh = 0.001
fc.thresh = 1

# parameters for designating non-dynamic peaks
sig.un = 0.5
fc.un = 0.25

# motif enrichment params
step = 20
```

```
half.win = 600
delbp = 60

# smoothing param
loess.span = 0.3

# save.image("enrichdat_600bp_step20.RData")
# see enrichSummary.sh for analysis
```

We run the analysis on an HPC system using a SLURM job.

```
####################################################
## main analysis scripts
####################################################

####### org_main.sh #######
#!/bin/bash
#SBATCH -A CivelekLab
#SBATCH --ntasks=1
#SBATCH --time=8:00:00
#SBATCH --partition=standard

dir=~/motifs/TFfimo/summary/Renrich
cd ${dir}


# sbatch --array=1-215 org_main.sh
cnt=${SLURM_ARRAY_TASK_ID}
${dir}/org_run.sh ${cnt}


##########################
####### org_run.sh #######
#!/bin/bash

# chmod u+x *.sh

cnt="$1"

dir=~/motifs/TFfimo/summary/Renrich
cd ${dir}

# modules
module purge
module load gcc R/3.5.1_test

mkdir run_${cnt}
cp -t run_${cnt} enrichdat_600bp_step20.RData getenrichdat.R array$cnt
cd run_${cnt}

# call the R script
Rscript --vanilla getenrichdat.R ${cnt}

#########################################
####### R script - getenrichdat.R #######

library(dplyr)
library(DESeq2)
library(bigWig)

# input
args = commandArgs(trailingOnly=TRUE)

# load data
load("enrichdat_600bp_step20.RData")

# region size for evaluating the peak index
delbp = 60

# fimo data dir
fname = paste0("array",args[1])
id = read.table(fname,stringsAsFactors=F)[1,1]
id = gsub(".txt", ".bigWig", id)
fimo.dir = "/nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/motifs/TFfimo/res/s
fimo.dir = paste0(fimo.dir,id)

# get factor id and load fimo data
fac = id
mot = strsplit(fac,"fimo_")[[1]][2]
mot = strsplit(mot,".bigWig")[[1]][1]
id = id.map$ids.orig[id.map$ids.new == mot]
bw.fimo = load.bigWig(fimo.dir)

# loop through each pairwise comparison
fimo.frame = c()
for(comp in names(res.pairs))

    # differential peak analysis significant indices
    res_ii = res.pairs[[comp]]
    ind.up = which(res_ii$padj<sig.thresh & res_ii$log2FoldChange>fc.thresh)
    ind.dn = which(res_ii$padj<sig.thresh & res_ii$log2FoldChange<(-1)*fc.thresh)
    ind.un = which(res_ii$padj>sig.un & abs(res_ii$log2FoldChange)<fc.un)
    if(length(ind.up)==0)next

    # get peak data in bed format and document peak counts
    up.bed = get.map.from.res( rownames(res_ii[ind.up,]) )
    dn.bed = get.map.from.res( rownames(res_ii[ind.dn,]) )
    un.bed = get.map.from.res( rownames(res_ii[ind.un,]) )
    len.up = length(ind.up)
    len.dn = length(ind.dn)
    len.un = length(ind.un)

    # load bigwig factor mapping data into peak coordinates
    # number of TFBS in each peak region
    up = bed.region.bpQuery.bigWig(bw.fimo, up.bed)
    dn = bed.region.bpQuery.bigWig(bw.fimo, dn.bed)
    un = bed.region.bpQuery.bigWig(bw.fimo, un.bed)
    names(up) = rownames(up.bed)
    names(dn) = rownames(dn.bed)
    names(un) = rownames(un.bed)

    # match atac peaks with motif coordinates
    # get numbers of peaks with >0 motifs
    # chi square test result
    result = get.xsquare.table(up=up, dn=dn, un=un)
    resultn = apply(result,2,function(x)100*x/(sum(x)))
    chi = chisq.test(result)

    # get read mapping coords with half.win around each summit
    up.ind = sapply(rownames(res_ii)[ind.up],function(x)which(all.peaks==x))
    dn.ind = sapply(rownames(res_ii)[ind.dn],function(x)which(all.peaks==x))
    un.ind = sapply(rownames(res_ii)[ind.un],function(x)which(all.peaks==x))
    bed.up = bed.window(summits=all.summits[up.ind], half.win=half.win)
    bed.dn = bed.window(summits=all.summits[dn.ind], half.win=half.win)
    bed.un = bed.window(summits=all.summits[un.ind], half.win=half.win)

    # matching peak data with the motif and computing features of the trace
bps = seq(-half.win, half.win, length.out=2*half.win/step)
    updat = motif.map(bigwig=bw.fimo,coords=bed.up,step=step,bps=bps)
    dndat = motif.map(bigwig=bw.fimo,coords=bed.dn,step=step,bps=bps)
    undat = motif.map(bigwig=bw.fimo,coords=bed.un,step=step,bps=bps)
    var.up = var(updat$mean)
    var.dn = var(dndat$mean)
    var.un = var(undat$mean)
    cv.up = var(updat$mean) / mean(updat$mean)
    cv.dn = var(dndat$mean) / mean(dndat$mean)
    cv.un = var(undat$mean) / mean(undat$mean)

# compute peak index
    indL = which(updat$bp < -(half.win-delbp))
    indM = which(updat$bp >= -delbp/2 & updat$bp <= delbp/2)
    indR = which(updat$bp > (half.win-delbp))
    pk.index.up = mean(updat$mean[indM]) / mean(c(updat$mean[indL],updat$mean[indR]))
    pk.index.dn = mean(dndat$mean[indM]) / mean(c(dndat$mean[indL],dndat$mean[indR]))
    pk.index.un = mean(undat$mean[indM]) / mean(c(undat$mean[indL],undat$mean[indR]))

    # summary metrics
    IU = resultn[1,2] - resultn[1,1] # inc with motif - un with motif
    DU = resultn[1,3] - resultn[1,1] # dec with motif - un with motif
    ID = resultn[1,2] - resultn[1,3] # inc with motif - dec with motif
    DI = resultn[1,3] - resultn[1,2] # dec with motif - inc with motif
    new = c(id, mot, comp, chi$p.value, chi$statistic, IU, DU, ID, DI,
            var.up, var.dn, var.un, cv.up, cv.dn, cv.un, len.up, len.dn, len.un,
pk.index.up, pk.index.dn, pk.index.un)
    names(new) = c("TF", "id", "tcomp","pval","xsq", "INCminusUN", "DECminusUN",
                   "INCminusDEC", "DECminusINC", "varINC", "varDEC", "varUN",
                   "cvINC", "cvDEC", "cvUN", "npeakINC", "npeakDEC", "npeakUN",
"pkindINC", "pkindDEC", "pkindUN")
    fimo.frame = rbind(fimo.frame, new)

 # pairwise comparison

# data processing
dat = as.data.frame(fimo.frame,stringsAsFactors=F)
dat[,4:18] = apply(dat[,4:18],2,function(x)data.matrix(x) %>% as.numeric)
rownames(dat) = 1:nrow(dat)
dat$pval = p.adjust(dat$pval, method="BH")

# write data out
fname = paste0("fimoSummary_",mot,".RData")
save(dat,file=fname)
```

We aggregate the fimo data as follow.

```
#####################################################
## aggregation of the results
#####################################################

dir=/nv/vol192/civeleklab/warren/MGlab/ATAC_WAFD/3T3_ATAC1-3/motifs/TFfimo/summary/Renri
cd $dir

# R script, aggregate.R
files = list.files()
dat.file = files[grep("fimoSummary_",files)]
motif = strsplit(dat.file,"fimoSummary_")[[1]][2]
motif = strsplit(motif,".RData")[[1]][1]
load(dat.file)
fname = paste0("fimoSummary_",motif,".txt")
write.table(dat,fname,sep="",quote=F,col.names=T,row.names=F)

# loop through all data folders and collect R data in .txt format
folders=$(ls -d *run_*)
for fold in $folders
do
cp aggregate.R $fold
cd $fold
module purge
module load gcc R/3.5.1_test
Rscript --vanilla aggregate.R
cp *.txt* ..
cd ..
done

# isolate and aggregate text files into a list
mkdir Rtxt
mv *fimoSummary_motif* Rtxt
cd Rtxt

# R code
files = list.files()
fac.chisq = list()
for(ii in files)
motif = strsplit(ii,"fimoSummary_")[[1]][2]
motif = strsplit(motif,".txt")[[1]][1]
dat = read.table(ii,stringsAsFactors=F,header=T,sep="")
fac.chisq[[motif]] = dat

save(fac.chisq, file="enrichdatProcessed2M.RData")
```

Next we evaluate enrichment in differentially expressed ATAC peaks (Motif_in_peak_20190401.R).

```
library(dplyr)
library(DESeq2)
library(ggplot2)
library(bigWig)
library(prodlim)
library(lattice)
library(stringr)
```

```
dir = paste0("/media/wa3j/Seagate2/Documents/PRO/",
             "adipogenesis/July2018/atac_time_deg/motifEnrich")
setwd(dir)

############################################################
# load data from differential peak analysis
# load data from de novo motif analysis
############################################################

# load annotation for all peaks (summits.bed)
# see empiricalSummits_20190325.R
load("ATACsummits_20190325.RData")
all.peaks = paste0(summits.bed[,1],":",summits.bed[,2],"-",summits.bed[,3])
all.summits = paste0(summits.bed[,1],":",summits.bed[,2],
                     "-",summits.bed[,3],",",summits.bed[,4])

# load differential peak data
# see atac_time_deg_20190214.R
load("pairwise.deg.RData")

# load annotation for all peaks (bed.map)
# see atac_time_deg_20190214.R
load("bed.map.RData")
all.peaks = paste0(bed.map[,1],":",bed.map[,2],"-",bed.map[,3])

# functions to load and plot atac data
code.dir = paste0("/run/user/1001/gvfs/smb-share:server=home1.virginia.edu,share=cphgdes
                  "users/wa3j/MGlab/Analysis/Adipogenesis/ATAC_time_DEG/peak_enrich")
source(paste0(code.dir,"/motifEnrich_functions.R"))

# load atac deseq2 data object with sizefactors, deseq_obj_preadip
# see atac_time_deg_20190214.R
load("atacDeseq.RData")
sizefac = sizeFactors(deseq_obj_preadip)

# upload mapping between TF motif and database names
pathToFile <- paste0(getwd(),"/motif.id.key.txt")
f <- file(pathToFile, "rb")
rawText <- readLines(f)
close(f)
motifs0 <- sapply(rawText, str_split, "")
motifs.raw = unlist(motifs0)[-c(1)] %>% unname
ids.new = sapply(motifs.raw,function(x)strsplit(x," ",fixed=TRUE)[[1]][1]) %>% unname
ids.orig = sapply(motifs.raw,function(x)
  id = strsplit(x," ",fixed=TRUE)[[1]]
  out = paste(id[2:length(id)], collapse=" ")
) %>% unlist
id.map = as.data.frame(cbind(ids.new,ids.orig),stringsAsFactors=F)
rownames(id.map) = 1:nrow(id.map)

############################################################
# analysis parameters
############################################################

# preadipogenesis times
t.order = cbind(c("t0","20min","40min","60min","2hr","3hr","4hr"),
                c(0,0.33,0.67,1,2,3,4)) %>%
  as.data.frame(stringsAsFactors=FALSE)
```

```
names(t.order) = c("condition","time")


# bigwig import params
file.prefix = "3T3_"
file.suffix = ".bigWig"
min.length = 1

# fdr threshold for filtering all fimo results per factor
fdr.thresh = 0.001
maxdiff.thresh = 10

# peak count threshold for considering optimal chi-sq result
pk.cnt.thresh = 100

# parameters for designating dynamic peaks
sig.thresh = 0.001
fc.thresh = 1

# parameters for designating non-dynamic peaks
sig.un = 0.5
fc.un = 0.25

# motif enrichment params
step = 20
half.win = 600
delbp = 60

# smoothing param
loess.span = 0.3

# save.image("enrichdat_600bp_step20.RData")
# see enrichSummary.sh for analysis

############################################################
# perform enrichment analyses for each factor
# see enrichSummary.sh for the implementation of this analysis on Rivanna
# enrich1-8.R does this analysis, too
# note that p-values are BH adjusted within factor in the .sh script in Rivanna
############################################################


dir = paste0("/media/wa3j/Seagate2/Documents/PRO/",
             "adipogenesis/July2018/atac_time_deg/motifEnrich")
setwd(dir)

# data location for fimo bigWigs
# bigwigs produced in TFfimo216.sh
fimo.bigWig.path = paste0("/media/wa3j/Seagate2/Documents/PRO/adipogenesis",
                          "/July2018/atac_time_deg/motifEnrich/fimo_bigWig750k")

# load fac.chisq
# load("enrichdatProcessed500k.RData")


############################################################
# format, sort, and filter enrichment data
############################################################
```

```
# decide which time comparisons to consider based on peak counts
# note that nmotifUN is the number of unchanged peaks for the given time comparison
fac1 = fac.chisq[[1]]
fac1 = fac1[,c(3,16:18)]
indlo1 = which(fac1$npeakINC < 200)
indlo2 = which(fac1$npeakDEC < 200)
indlo = union(indlo1, indlo2)
fac1[indlo,]
fac1[-indlo,]

# look at relationship between variability and motif percentages
plt.dat = c()
for(ii in 1:length(fac.chisq))
  new = fac.chisq[[ii]]
  plt.dat = rbind(plt.dat, new[-indlo,])

plot(plt.dat$INCminusUN, plt.dat$varINC)
plot(plt.dat$DECminusUN, plt.dat$varDEC)
hist(plt.dat$varINC %>% log)
hist(plt.dat$varDEC %>% log)

# separate into enrichment direction classes
enrich.thresh = 5
pkind.thresh = 0.5
enrich.inc = plt.dat %>% filter( (INCminusUN>enrich.thresh & (pkindINC-pkindUN)>pkind.th
                                    (INCminusDEC>enrich.thresh & (pkindINC-pkindDEC)>pkin
enrich.dec = plt.dat %>% filter((DECminusUN>enrich.thresh & (pkindDEC-pkindUN)>pkind.thr
                                   (DECminusINC>enrich.thresh & (pkindDEC-pkindINC)>pkind
enrich.un = plt.dat %>% filter((DECminusUN < -enrich.thresh & (pkindUN-pkindDEC)>pkind.t
                                  (INCminusUN < -enrich.thresh & (pkindUN-pkindINC)>pkind

# filter based on enrichment peak index
enrich.pk.thresh = 1.8
enrich.inc = enrich.inc %>% filter(pkindINC > enrich.pk.thresh) %>% mutate(class="inc")
enrich.dec = enrich.dec %>% filter(pkindDEC > enrich.pk.thresh) %>% mutate(class="dec")
enrich.un = enrich.un %>% filter(pkindUN > enrich.pk.thresh) %>% mutate(class="un")

# check fdrs
max(enrich.inc$pval) # 2.762924e-11
max(enrich.dec$pval) # 0.002106126
max(enrich.un$pval)  # 0.001951505

############################################################
# aggregate and summarize enrichment data
# filter for plotting enrichment data for each factor
############################################################

# aggregate all data
processed.enrich.dat = rbind(enrich.inc, enrich.dec, enrich.un)
length(unique(processed.enrich.dat$id))
unique(processed.enrich.dat$TF)

# summarize data for each factor
enrich.summary = c()
for(ii in unique(processed.enrich.dat$id))
  motif = ii
  ind = which(processed.enrich.dat$id == motif)
  TF = processed.enrich.dat$TF[ind][1]
  dat = processed.enrich.dat[ind,]
  minp = min(dat$pval)
  ind.inc = grep("inc",dat$class)
  ind.dec = grep("dec",dat$class)
  ind.unc = grep("un",dat$class)
  ninc = length(ind.inc)
  ndec = length(ind.dec)
  nunc = length(ind.unc)
  comp.inc = paste(dat$tcomp[ind.inc],collapse=",")
  comp.dec = paste(dat$tcomp[ind.dec],collapse=",")
  comp.unc = paste(dat$tcomp[ind.unc],collapse=",")
  new = c(TF,motif,minp,ninc,ndec,nunc,comp.inc,comp.dec,comp.unc)
  enrich.summary = rbind(enrich.summary, new)

enrich.summary = as.data.frame(enrich.summary,stringsAsFactors=F)
rownames(enrich.summary) = 1:nrow(enrich.summary)
names(enrich.summary) = c("factor","id","minFDR","ninc","ndec","nunc",
                          "compsINC","compsDEC","compsUNC")
#write.table(enrich.summary,"enrichSummary750k.txt",col.names=T,row.names=F,quote=F,sep=

# prioritize data for plotting
# select the time comparison with the largest peak index
enrich.inc.plt = enrich.plot.select(enrich.dat=enrich.inc, comp.mode="inc")
enrich.dec.plt = enrich.plot.select(enrich.dat=enrich.dec, comp.mode="dec")
enrich.unc.plt = enrich.plot.select(enrich.dat=enrich.un, comp.mode="un")


############################################################
# plot enrichment data
############################################################

plot.enrich(comp.mode="inc",fname="enrichINCfiltered500k.pdf",mfrow=c(3,3),
            enrich.dat=enrich.inc.plt[order(-enrich.inc.plt$pkindINC),],
            res.pairs=res.pairs, delbp=delbp, loess.span=loess.span,
            sig.thresh=sig.thresh, fc.thresh=fc.thresh,
            sig.un=sig.un, fc.un=fc.un,
            half.win=half.win, step=step,
            fimo.bigWig.path=fimo.bigWig.path,plt.trace=F)

plot.enrich(comp.mode="dec",fname="enrichDECfiltered500k.pdf",mfrow=c(3,3),
            enrich.dat=enrich.dec.plt[order(-enrich.dec.plt$pkindDEC),],
            res.pairs=res.pairs, delbp=delbp, loess.span=loess.span,
            sig.thresh=sig.thresh, fc.thresh=fc.thresh,
            sig.un=sig.un, fc.un=fc.un,
            half.win=half.win, step=step,
            fimo.bigWig.path=fimo.bigWig.path,plt.trace=F)

plot.enrich(comp.mode="un",fname="enrichUNCfiltered500k.pdf",mfrow=c(3,3),
            enrich.dat=enrich.unc.plt[order(-enrich.unc.plt$pkindUN),],
            res.pairs=res.pairs, delbp=delbp, loess.span=loess.span,
            sig.thresh=sig.thresh, fc.thresh=fc.thresh,
            sig.un=sig.un, fc.un=fc.un,
            half.win=half.win, step=step,
            fimo.bigWig.path=fimo.bigWig.path,plt.trace=F)


############################################################
```

```
# plot motifs and summarize motif clusters
############################################################

dir = paste0("/media/wa3j/Seagate2/Documents/PRO/",
             "adipogenesis/July2018/atac_time_deg/motifEnrich")
setwd(dir)

# load data from motif_communities_TF_20190317.R
# load("TFanalysis_20190320.RData")

require(ggplot2)
require(ggseqlogo)
library(gridExtra)
library(dplyr)
library(seqLogo)
library(prodlim)
library(lattice)

# load motif plotting functions
code.dir = paste0("/run/user/1001/gvfs/smb-share:server=home1.virginia.edu,share=cphgdes
          "/users/wa3j/MGlab/Analysis/Adipogenesis/ATAC_time_DEG/TFcluster/motif_clust_f
source(code.dir)

# plot the motif signatures
final.plts = list()
p=1
for(id in enrich.summary$id)
  fac = enrich.summary$factor[enrich.summary$id==id]
  mat = flt.pwm.clust[[id]]
  final.plts[[p]] = logo.plt(pwm=mat, title=fac)
  p=p+1

pdf("TFs_filtered500k.pdf", onefile = TRUE, height=9, width=4.5)
marrangeGrob(grobs=final.plts, nrow=6, ncol=1, top=NULL)
dev.off()

# isolate filtered factors
motif.community5 = motif.community4 %>% mutate(comm = paste0("community_",community))
motif.community5 = motif.community5[motif.community5$motif %in% enrich.summary$id,]
cols = sapply(motif.community5$comm,function(x)which(names(datID)==x))
TFspreadsheet = datTF[,cols]
# write.table(TFspreadsheet,"TFclusters750k.txt",col.names=T,row.names=F,sep="")
# write.table(motif.community5,"TFclusterAnnotation750k.txt",col.names=T,row.names=F,sep

# generate lists of pwm ids for each community
TF.list = list()
for(ii in 1:ncol(datTF))
  mtfs = datID[,ii]
  TF.list[[names(datTF)[ii]]] = mtfs[mtfs != 0]


############################################################
# plot selected pwms
############################################################

# specify communities of interest
plt.comms = c(5,94,122,137,378)
plt.comms = c(25)
plt.comms = c(61,83)
plt.comms = paste0("community_",plt.comms)

# directory for all pwms
pwm.dir = paste0("/media/wa3j/Seagate2/Documents/PRO/",
                 "adipogenesis/July2018/atac_time_deg/communities/TF/TF679")

# function to get pwm
get.pwm = function(pwm.dir=NULL,motif=NULL)
  fname = paste0(pwm.dir,"/",motif,".txt")
  mat = read.table(fname,sep="",header=F,stringsAsFactors=F,fill=T) %>% t
  nas = apply(mat,1,function(x)all(is.na(x)))
  ind = which(nas==TRUE)
  if(length(ind)>0)mat = mat[-ind,]
  rownames(mat) = c("A","C","G","T")
  colnames(mat) = c(1:ncol(mat))
  return(mat)


# plot the PWMs for all community members
comm.plts = list()
p=1
for(ii in 1:length(plt.comms))
  comm = plt.comms[ii]
  mtfs = TF.list[[comm]]
  for(id in mtfs)
    print(paste(comm,", ",id))
    mat = get.pwm(pwm.dir=pwm.dir, motif=id)
    comm.plts[[p]] = logo.plt(pwm=mat, title=id)
    p=p+1


pdf("pwm_mixed3.pdf", onefile = TRUE, height=9, width=4.5)
marrangeGrob(grobs=comm.plts, nrow=6, ncol=1, top=NULL)
dev.off()
```

9 References

Bailey TL, Boden M, Buske FA, Frith M, Grant CE, Clementi L, Ren J, Li WW, Noble WS (2009). "MEME SUITE: tools for motif discovery and searching." *Nucleic Acids Research*, **37**, W202–208.

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. URL `http://igraph.sf.net`.

Ernst J, Bar-Joseph Z (2006). "STEM: a tool for the analysis of short time series gene expression data." *BMC Bioinformatics*, **7**, 191.

Heinz S, Benner C, Spann N, Bertolino E, Lin YC, Laslo P, Cheng JX, Murre C, Singh H, Glass CK (2010). "Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities." *Molecular Cell*, **38**, 576–589.

Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D (2002). "The human genome browser at UCSC." *Genome Research*, **12**, 996–1006.

Kent WJ, Zweig AS, Barber G, Hinrichs AS, Karolchik D (2010). "BigWig and BigBed: enabling browsing of large distributed datasets." *Bioinformatics (Oxford, England)*, **26**, 2204–2207.

Khan A, Fornes O, Stigliani A, Gheorghe M, Castro-Mondragon JA, van der Lee R, Bessy A, Chèneby J, Kulkarni SR, Tan G, Baranasic D, Arenillas DJ, Sandelin A, Vandepoele K, Lenhard B, Ballester B, Wasserman WW, Parcy F, Mathelier A (2018). "JASPAR 2018: update of the open-access database of transcription factor binding profiles and its web framework." *Nucleic Acids Research*, **46**, D260–D266.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, 1000 Genome Project Data Processing Subgroup (2009). "The Sequence Alignment/Map format and SAMtools." *Bioinformatics (Oxford, England)*, **25**, 2078–2079.

Liu Y, Walavalkar NM, Dozmorov MG, Rich SS, Civelek M, Guertin MJ (2017). "Identification of breast cancer associated variants that modulate transcription factor binding." *PLoS Genetics*, **13**, e1006761.

Love MI, Huber W, Anders S (2014). "Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2." *Genome Biology*, **15**, 550.

Spivak AT, Stormo GD (2012). "ScerTF: a comprehensive database of benchmarked position weight matrices for Saccharomyces species." *Nucleic Acids Research*, **40**, D162–168.

Wang Z, Chu T, Choate LA, Danko CG (2019). "Identification of regulatory elements from nascent transcription using dREG." *Genome Research*, **29**, 293–303.

Weirauch MT, Yang A, Albu M, Cote AG, Montenegro-Montero A, Drewe P, Najafabadi HS, Lambert SA, Mann I, Cook K, Zheng H, Goity A, van Bakel H, Lozano JC, Galli M, Lewsey MG, Huang E, Mukherjee T, Chen X, Reece-Hoyes JS, Govindarajan S, Shaulsky G, Walhout AJM, Bouget FY, Ratsch G, Larrondo LF, Ecker JR, Hughes TR (2014). "Determination and inference of eukaryotic transcription factor sequence specificity." *Cell*, **158**, 1431–1443.