Identification of transcriptional units

Warren Anderson, Mete Civelek, Michael Guertin

August 1, 2019

This guide provides code and documentation of analyses from Anderson et al., 2019, *Defining data-driven gene coordinates with primaryTranscriptAnnotation in R*

Contents

1 Overview

The following analyses were completed to obtain annotated transcriptional unit (TU) coordinates. We performed *de novo* TU dentification using a hidden Markov model (HMM) based method known as *groHMM* (Chae *et al.*, 2015). We used our gene annotations that were based on inferred TSSs/TTSs to annotate the identified TUs.

2 De novo TU identification

We used the R/bioconductor package *groHMM* to implement *de novo* TU identification. This method has been shown to outperform complementary methods in terms of sensitivity and specificity (Chae *et al.*, 2015). Because *groHMM* supports parameter tuning for performance optimization, we varied key analysis parameters according to directions in the package documentation. As described previously (Chae *et al.*, 2015), we varied the log probability of a transition from the transcribed to the untranscribed state and the read count variance in the untranscribed state (*LtProbB* and *UTS*, respectively). For this analysis, we used the gene annotations inferred based on our novel methods to evaluate the performance of each set of TUs identified from a given HMM parameterization. The vignette for our *primaryTranscriptAnnotation* R package contains details on inferring transcript coordinates (`https://github.com/WarrenDavidAnderson/genomicsRpackage/tree/master/primaryTranscriptAnnotation`). In particular, the data processing undertaken to obtain the input file inferredcoords.bed is documented in the vignette (see primaryTranscriptAnnotation-vignette.pdf). We implemented the following code using a server to perform the parameter variations and document some basic performance metrics. The R code for this analysis can also be found in *TUid.R*.

First we merged bam files from all pre-adipogenesis time points using the following unix command. We call samtools merge from a directory with all of the pre-adipogenesis bam files:

```
samtools merge preadip_merged.bam *.bam
```

Here is the code that formats the data necessary for identifying TUs.

```
################################################################
## get data files, perform basic processing
## just run this in R
################################################################
lib.loc = "/h4/t1/users/wa3j/software/R_libs"
.libPaths(lib.loc)
library(dplyr, lib.loc=lib.loc)
library(bigWig, lib.loc=lib.loc)
library(groHMM, lib.loc=lib.loc)
library(GenomicFeatures, lib.loc=lib.loc)

# get merged bam file
bam.file="preadip_merged.bam"
data <- readGAlignments(bam.file)
data_gr <- granges(data)
data_gr <- sort(data_gr)
expr <- keepStandardChromosomes(data_gr, pruning.mode="coarse")

# get gene annotations based on TSS/TSS identification
gene.ann0 = read.table("inferredcoords.bed",sep="\t",header=F,stringsAsFactors=F)
names(gene.ann0) = c("chr","start","end","gene","xy","strand")
gene.ann = makeGRangesFromDataFrame(gene.ann0[,-5], seqnames.field="chr",
                                    start.field="start",end.field="end",
                                    strand.field="strand",
                                    starts.in.df.are.0based=TRUE,
                                    keep.extra.columns=TRUE)
gene.ann <- sort(gene.ann)
gene.ann <- keepStandardChromosomes(gene.ann, pruning.mode="coarse")

# window parameters
Fp <- windowAnalysis(expr, strand="+", windowSize=50)
Fm <- windowAnalysis(expr, strand="-", windowSize=50)

save.image("groHMMdata.RData")
```

Note that the *evaluateHMMInAnnotations()* function uses the inferred gene annotations to document 'merge errors' and 'dissociation errors'. Merge errors occur when a given TU overlaps multiple gene annotations. Dissociation errors occur when multiple TUs overlap a given gene annotation.

We implemented the following analysis to obtain TU coordinates based on distinct parameterizations of the HMM model.

```
################################################################
## HMM parameter testing
## hmm.param.test.R
################################################################

# run code
# nohup Rscript hmm.param.test.R &

lib.loc = "/h4/t1/users/wa3j/software/R_libs"
.libPaths(lib.loc)
library(dplyr, lib.loc=lib.loc)
library(bigWig, lib.loc=lib.loc)
library(groHMM, lib.loc=lib.loc)
library(GenomicFeatures, lib.loc=lib.loc)

# load initial data
load("groHMMdata.RData")

# set the number of cores for grohmm
options(mc.cores=getCores(60))

# specify parameters for variation
vars = c(10,15,20,25,30,40,50,60,80,100,150,200,300)
ltpr = c(-50,-100,-200,-300,-400,-500)
LtProbB = sapply(ltpr,function(x){rep(x,length(vars))}) %>% as.vector
UTS = rep(vars,length(ltpr))
tune <- data.frame(LtProbB=LtProbB, UTS=UTS)

# testing parameter sets
hmm.vars <- mclapply(seq_len(nrow(tune)), function(x) {
  hmm <- detectTranscripts(Fp=Fp, Fm=Fm, LtProbB=tune$LtProbB[x],
                           UTS=tune$UTS[x], threshold=1)
  e <- evaluateHMMInAnnotations(hmm$transcripts, gene.ann)
  return(list(hmm=hmm, eval=e$eval))
}, mc.cores=getOption("mc.cores"),  mc.silent=FALSE)
names(hmm.vars) = apply(tune,1,function(x){paste0(x[2],"_",x[1])})

save(hmm.vars, file="grohmm_vars.RData")
```

3 Selection of the optimal HMM-based TU set

Distinct *groHMM* parameterizations produced varying degrees of merge and dissociation error. In addition to evaluating merge and dissociation errors, we evaluated HMM sensitivity by looking at how many reads mapped to regions of the genome that were not identified as transcribed. First we established a file documenting the respective chromosome sizes for the entire mouse genome (available here: `http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes`).

```
################################################################
## mouse genome annotation
################################################################

# set mm10 genome bed
# wget http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes
gnme0 = read.table("mm10.chrom.sizes",header=F,stringsAsFactors=F)
names(gnme0) = c("chr","len")
rem = c(grep("random",gnme0$chr), grep("chrUn",gnme0$chr))
mm10.bed = gnme0[-rem,]
names(mm10.bed) = c("chr","end")
write.table(mm10.bed,"mm10.bed",sep="\t",quote=F,col.names=F,row.names=F)

# sort the file
command2=paste('sort -k1,1 -k2,2n', 'mm10.bed', '> mm10.sorted.bed')
system(command2)
```

We next implemented the following analysis to obtain parameter sensitivity information for further evaluation. The .bigWig files for this analysis can be obtained from CyVerse.

```
wget https://de.cyverse.org/dl/d/068D0BC4-89F1-4A55-AE27-694A2679AD6E/preadip_minus_merged.bigWig
wget https://de.cyverse.org/dl/d/026727C7-36EF-435E-9AA1-7A6F9E79C514/preadip_plus_merged.bigWig
```

Here is the code documenting TU identification performance with respect to HMM parameterization.

```
################################################################
## get sensitivity information
## hmm.param.sens.R
################################################################

# run code
# nohup Rscript hmm.param.sens.R &

# load libraries
lib.loc = "/h4/t1/users/wa3j/software/R_libs"
.libPaths(lib.loc)
library(bigWig, lib.loc=lib.loc)
library(groHMM, lib.loc=lib.loc)
library(GenomicFeatures, lib.loc=lib.loc)
library(dplyr, lib.loc=lib.loc)

# load hmm data
load("grohmm_vars.RData")

# load bigwigs
bw.plus = load.bigWig("preadip_plus_merged.bigWig")
bw.minus = load.bigWig("preadip_minus_merged.bigWig")

# bed dir
bed.bin = "/h4/t1/apps/seqanal/bedtools/bin/"
```

```
# loop through hmms, check reads in complement, document metrics

eval.dat = c()

for(ii in 1:length(hmm.vars)){

  # get complement to hmm regions
  hmm.test = hmm.vars[[ii]]$hmm
  transcripts = hmm.test[["transcripts"]]
  if(is.null(transcripts)==TRUE){next}
  hmm.bed0 <- data.frame(chr=seqnames(transcripts),
                         start=start(transcripts)-1,
                         end=end(transcripts),
                         names=c(rep(".", length(transcripts))),
                         scores=c(rep(".", length(transcripts))),
                         strand=strand(transcripts))
  hmmp = hmm.bed0 %>% filter(strand=="+")
  hmmm = hmm.bed0 %>% filter(strand=="-")
  write.table(hmmp,"hmmp.bed",sep="\t",quote=F,col.names=F,row.names=F)
  write.table(hmmm,"hmmm.bed",sep="\t",quote=F,col.names=F,row.names=F)
  command1=paste('sort -k1,1 -k2,2n', 'hmmp.bed', '> hmmp.sorted.bed')
  command2=paste('sort -k1,1 -k2,2n', 'hmmm.bed', '> hmmm.sorted.bed')
  system(command1); system(command2)
  comm1 = paste0(bed.bin,"complementBed -i hmmp.sorted.bed -g mm10.sorted.bed
         > compp.bed")
  comm2 = paste0(bed.bin,"complementBed -i hmmm.sorted.bed -g mm10.sorted.bed
         > compm.bed")
  system(comm1); system(comm2)
  complplus = read.table("compp.bed",stringsAsFactors=F)
  complminus = read.table("compm.bed",stringsAsFactors=F)
  system(paste0("rm hmmp.bed hmmm.bed hmmp.sorted.bed hmmm.sorted.bed
         compp.bed compm.bed"))

  # map reads to the complement regions
  plus.bed = complplus %>% mutate(gene=".",xy=".",strand="+")
  minus.bed = complplus %>% mutate(gene=".",xy=".",strand="-")
  names(plus.bed)[1:3] = names(minus.bed)[1:3] = c("chr","start","end")
  cnt.plus = bed.region.bpQuery.bigWig(bw.plus, plus.bed)
  cnt.minus = bed.region.bpQuery.bigWig(bw.minus, minus.bed)

  # get count density metrics
  len.p = plus.bed$end - plus.bed$start
  len.m = minus.bed$end - minus.bed$start
  density.plus = cnt.plus / len.p
  density.minus = cnt.minus / len.m
  counts = c(cnt.plus, cnt.minus)
  densities = c(density.plus, density.minus)
  max_cnt = max(counts);    max_den = max(densities)
  mean_cnt = mean(counts);  mean_den = mean(densities)
  med_cnt = median(counts); med_den = median(densities)
  sum_cnt = sum(counts);    sum_den = sum(densities)

  # combine evaluation metrics
  pars0 = strsplit(names(hmm.vars)[ii],"_")[[1]]
  LtProbB = pars0[2]
  UTS = pars0[1]
  hmm.eval = hmm.vars[[ii]]$eval
  new = data.frame(LtProbB, UTS, hmm.eval,
                   max_cnt, mean_cnt, med_cnt, sum_cnt,
                   max_den, mean_den, med_den, sum_den)
  eval.dat = rbind(eval.dat, new)

} # ii, hmm loop

save(eval.dat, file="hmmEval.RData")
```

Note that this analysis generates information releated to counts and densities in regions defined as untranscribed according to a given implementation of *groHMM*. We now can process the information on HMM performance to select an optimal set of TUs for downstream analysis.

```
################################################################
## process error information to select the best hmm
################################################################

lib.loc = "/h4/t1/users/wa3j/software/R_libs"
.libPaths(lib.loc)
library(dplyr, lib.loc=lib.loc)
library(groHMM, lib.loc=lib.loc)
library(GenomicFeatures, lib.loc=lib.loc)

load("hmmEval.RData")
load("grohmm_vars.RData")

# sort data based on merge errors, dissoc errors, and sums of reads outside of TUs
eval.dat.sorted1 = eval.dat[with(eval.dat, order(merged, dissociated, sum_cnt)),] %>%
  dplyr::select(merged, dissociated, sum_cnt)
eval.dat.sorted2 = eval.dat[with(eval.dat, order(dissociated, merged, sum_cnt)),] %>%
  dplyr::select(merged, dissociated, sum_cnt)
eval.dat.sorted3 = eval.dat[with(eval.dat, order(sum_cnt, dissociated, merged)),] %>%
  dplyr::select(merged, dissociated, sum_cnt)
head(eval.dat.sorted1)
head(eval.dat.sorted2)
head(eval.dat.sorted3)

# look at quartiles for metrics of interest
quantile(eval.dat$merged)
quantile(eval.dat$dissociated)
quantile(eval.dat$sum_cnt)

# select the best hmm: third lowest merge error (2106, lowest quartile)
# lowest sum count (7968454), lowest quartile for dissociation error (114)
eval.dat[68,]
hmm.best = hmm.vars[["20_-500"]]$hmm

# convert to bed
transcripts = hmm.best[["transcripts"]]
hmm.bed0 <- data.frame(chr=seqnames(transcripts),
                       start=start(transcripts)-1,
                       end=end(transcripts),
                       names=c(rep(".", length(transcripts))),
                       scores=c(rep(".", length(transcripts))),
                       strand=strand(transcripts))

save(hmm.bed0, file="bestHMM.bed")
```

We evaluated HMM performance based on merge errors, dissociation errors, and total reads mapped to regions that were not identified by *groHMM*. We selected the HMM with the lowest read count outside of HMM-defined untranscribed regions. This HMM had the second lowest merge error and a relatively low dissociation error (both within the lowest error quartile).

4 References

Chae M, Danko CG, Kraus WL (2015). "groHMM: a computational tool for identifying unannotated and cell type-specific transcription units from global run-on sequencing data." *BMC bioinformatics*, **16**.