Working with CYCLeR

Stefan Stefanov

8/20/2021

CYCLeR is a pipeline for reconstruction of circRNA transcripts from RNA-seq data and their subsequent quantification. The algorithm relies on comparison between control total RNA-seq samples and circRNA enriched samples to identify circRNA specific features. Then the selected circRNA features are used to infer the transcripts through a graph-based algorithm. Once the predicted transcript set is assembled, the transcript abundances are estimated through an EM algorithm with **kallisto** [1]. **CYCLeR** takes as an input BAM files and back-splice junction (BSJ) files and outputs transcript infomation in different formats and a transcript abundance file.

Installation of CYCLeR

Command line tools needed

```
**STAR** - https://github.com/alexdobin/STAR
**samtools** - https://sourceforge.net/projects/samtools/files/samtools/
**kallisto** - http://pachterlab.github.io/kallisto/download
#We also suggest RStudio to supplement the interactive experience
**RStudio** - https://rstudio.com/products/rstudio/download/
```

R packages installation

```
library(devtools)
install_github("stiv1n/CYCLeR")
```

Pre-processing the data

Mapping with STAR

The STAR [2] mapping parameters are up to a personal preference. It is imperative to include the **intronMotif** tag. My suggestion would be 2-pass mapping with these parameters:

```
#first pass
STAR --alignSJoverhangMin 5 --outSAMstrandField intronMotif --outFilterMismatchNmax 3
--outFilterMismatchNoverLmax 0.1
--chimSegmentMin 15 --chimScoreMin 1 --chimJunctionOverhangMin 15
--outFilterMultimapNmax 50 --alignIntronMax 100000 --alignIntronMin 15
#second pass
STAR --outSAMstrandField intronMotif --outFilterMismatchNmax 7 --outFilterMismatchNoverLmax 0.3
--alignSJoverhangMin 15 --alignSJDBoverhangMin 3
--chimSegmentMin 15 --chimScoreMin 1 --chimJunctionOverhangMin 15
--outFilterMultimapNmax 50 --alignIntronMax 100000 --alignIntronMin 15
#converting to sorted BAM
samtools view -u -h Aligned.out.sam | samtools sort <name>_sorted.bam
```

Processing the BAM info in R

We need the information for read length, fragment length and library sizes from the BAM files.

```
#load the BSJ files
bam_file_prefix<-system.file("extdata", package = "CYCLeR")
filenames<-c("sample1_75","sample2_75","sample3_75","sample4_75")
BSJ_files_ciri<-paste0(BSJ_files_prefix,"/",filenames)
bam_files<-paste0(bam_file_prefix,filenames,".bam")
#mark the samples control and enriched or bare the consequences
sample_table<-data.frame(filenames,c("control","control","enriched","enriched"),bam_files,stringsAsFactors = F)
colnames(sample_table)<-c("sample_name","treatment","file_bam")
si<- DataFrame(sample_table[,c("sample_name","file_bam")])
si$file_bam <-BamFileList(si$file_bam, asMates = F)
#this holds all the needed info of the bam files for downstream processing
sc <- getBamInfo(si)
sample_table$lib_size<-sc@listData$lib_size
sample_table$read_len<-sc@listData$read_length
```

Use the provided sample table template.

```
##   sample_name treatment                                           file_bam
## 1  SRR1191323   control /home/sstefan/data/wt_vs_RR/SRR1191323_sorted.bam
## 2  SRR1191331   control /home/sstefan/data/wt_vs_RR/SRR1191331_sorted.bam
## 3  SRR1191327  enriched /home/sstefan/data/wt_vs_RR/SRR1191327_sorted.bam
## 4  SRR1191335  enriched /home/sstefan/data/wt_vs_RR/SRR1191335_sorted.bam
##   lib_size read_len
## 1 19255420      101
## 2 15068605      101
## 3 17144585      101
## 4 18667943      101
```

Selecting a BSJ set

Selecting a BSJ set is very important, because the algorithm assumes that the provided set of BSJ is *correct*. I suggest BSJ identification with **CIRI2** [3] and **CIRCexplorer2** [4], but the choice is up to a personal preference. I have provided some useful functions for parsing the output from BSJ identification software.

```
#load the BSJ files
BSJ_files_prefix<-paste0(system.file("extdata", package = "CYCLeR"),"/ciri_")
ciri_table<-parse.files(BSJ_files_ciri,file_path="","CIRI")
ciri_bsjs<-process.BSJs(ciri_table,sample_table)
# i would suggest combine the output of pipelines using different mapping tools
BSJ_files_prefix_CE<-paste0(system.file("extdata", package = "CYCLeR"),"/CE_")
ce_table<-parse.files(sample_table$sample_name,BSJ_files_prefix_CE,"CE")
ce_bsjs<-process.BSJs(ce_table,sample_table)
#we need to unify the results from the BSJ identification and counting
table_circ<-combine.two.BSJ.tables(ce_bsjs,ciri_bsjs)
#further downstream we need just the mean values for enriched samples
table_circ<-table_circ[,c("chr","start","end","meanRR")]
colnames(table_circ)<-c("chr","start","end","count")
#combine
BSJ_set<-union(ciri_bsjs$circ_id,ce_bsjs$circ_id)
BSJ_set<-BSJ_set[!grepl("caffold",BSJ_set)]#just in case
BSJ_set<-BSJ_set[!grepl("mitochondrion",BSJ_set)]
#################################################################
#converting the BSJ set into a GRanges object
BSJ_gr<-make.BSJ.gr(BSJ_set)
```

The parse.files can work with **CIRI2**, **CIRCexplorer2** or **TSV** file. Naturally a person may have different criterion for *correct* BSJs based on different criteria. It is not an issue as long as the data is presented in the following template:

```
head(table_circ)
```

```
## # A tibble: 6 x 4
##   chr   start    end      count
##   <chr> <chr>    <chr>    <dbl>
## 1 2L    10036818 10037279 1.18
## 2 2L    10080375 10081946 0.923
## 3 2L    10096914 10098010 0.332
## 4 2L    10096914 10137559 0.292
## 5 2L    10104792 10114895 0.268
## 6 2L    10120720 10122038 0.260
```

Transcript assembly

BSJ loci extraction

Prior to the feature detection the files need to be trimmed to speed up the process. Afterwards the transcript features (e.g. exons, junctions) are identified with **SGSeq** [5]. The files are convereted with **samtools** [6].

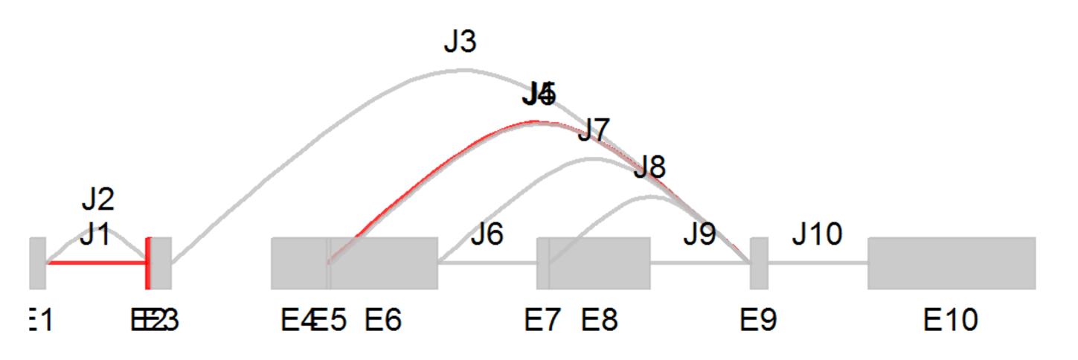
```
#####################################################
#get the gene/transcript info
#restoreSeqlevels(txdb)
txdb <- TxDb.Dmelanogaster.UCSC.dm6.ensGene
txdb <- keepSeqlevels(txdb, c("chr2L","chr2R","chr3R","chr3L","chr4","chrX","chrY"))
seqlevelsStyle(txdb) <- "Ensembl"
gene_ranges <- genes(txdb)
txf <- convertToTxFeatures(txdb)
#asnnotation as sg-object
sgf <- convertToSGFeatures(txf)
####################################################
samtools_prefix<-"/home/sstefan/software/samtools-1.10/bin/"
trimmed_bams<-filter.bam(BSJ_gr,sample_table,samtools_prefix)
sc@listData[["file_bam"]]<-trimmed_bams
#sc@listData[["sample_name"]]<-sample_table_ce$sample_name
```

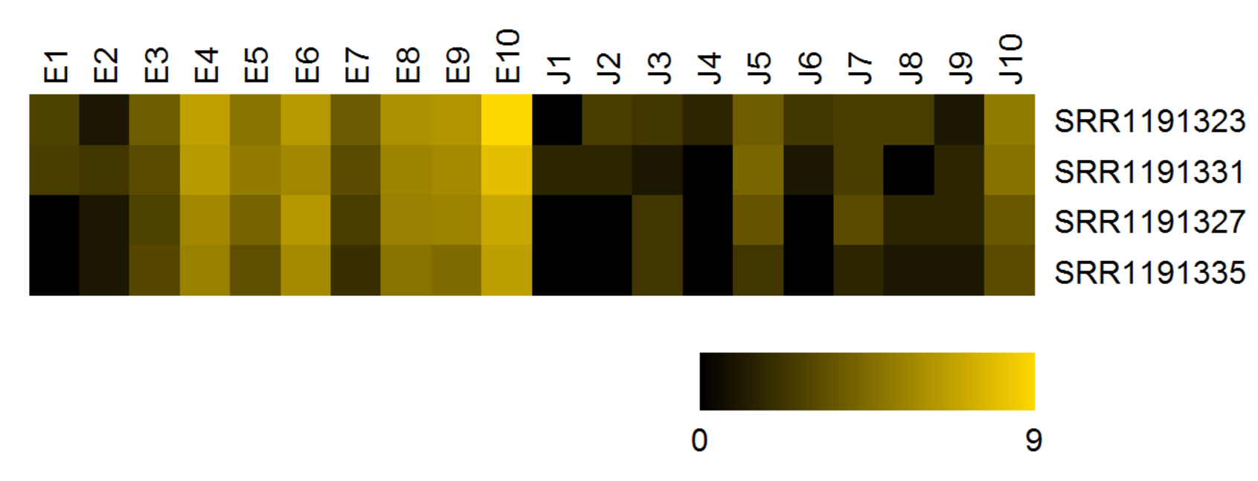
Feature identification with SGSeq

```
sgfc_pred <- analyzeFeatures(sc, min_junction_count=2, beta =0.1 , min_n_sample=1,cores=1,verbose=F)
sgfc_pred <- SGSeq::annotate(sgfc_pred, txf)
```

SGSeq feature plotting function can be used for visual representation of the control VS enriched difference

```
plotFeatures(sgfc_pred,  geneID = "1",assay = "counts",color_novel = "red", include = "both",tx_view=F,Rowv=NA, s
quare=T)
```





Re-couting and Processing the features

I prefer the **RSubread** [7] counting method, thus I re-count the identified exon features. I use the **SGseq** counted junctions. The features that are depleted in circRNA enriched samples need to be removed. **CYCLeR** provides 2 approaches for identifying depleted features: DEU strategy and simple comparison of normalized coverage values.

```
#extract BSJ-corrected splice graphs  (sg)
full_sg<-overlap.SG.BSJ(sgfc_pred,BSJ_gr) #includes linear and circular features
# we have made new feature set so we need to recount
full_fc<-recount.features(full_sg,sample_table)#fc==feature counts
#removing super low coverage features
full_sg<-full_sg[rowSums(as.data.frame(full_fc[,sample_table$treatment=="enriched"]))>15]
full_fc<-full_fc[rowSums(as.data.frame(full_fc[,sample_table$treatment=="enriched"]))>15,]
circ_sg<-full_sg[full_sg%over%BSJ_gr] #includes features within BSJ enclosed region
lin_sg<-full_sg[full_sg%outside%BSJ_gr] #includes features outside of BSJ enclosed region
#annotate the BSJ with the corresponding geneIDs
BSJ_sg<-make.BSJ.sg(circ_sg,BSJ_gr)
#full_fc<-count_matrix[full_sg@featureID,]
#get the correct genome for sequence info
bs_genome=Dmelanogaster
#RPKM calculation for exons
full_rpkm<-RPKM.calc(full_fc, full_sg, BSJ_gr, bs_genome=bs_genome , sample_table=sample_table, feature_type =
"e", gc_correction = T)
lin_rpkm<-full_rpkm[full_sg%outside%BSJ_gr,]
#extracting circ specific counts
circ_fc_adj<-full_rpkm[full_sg%over%BSJ_gr,]
depleted_exons<-find.depleted.features(circ_fc_adj,sample_table,circ_sg)
#making sure that the circ edge exons remian in the mix; they coudl be depleted in case of very low levels of the
circle
edge_features<-union(full_sg@featureID[start(full_sg)%in%start(BSJ_gr)],full_sg@featureID[end(full_sg)%in%end(BSJ
_gr)])
depleted_exons<-setdiff(depleted_exons,edge_features)
circ_exons<-circ_sg[!circ_sg@featureID%in%depleted_exons]# the final set of circRNA exons
circ_exons_counts<-circ_fc_adj[!circ_sg@featureID%in%depleted_exons,]
########################################################################
#now for junctions
#we need to normalize the junction read counts to the exon counts
count_matrix<-as.data.frame(counts(sgfc_pred))
count_matrix <- apply (count_matrix, c (1, 2), function (x) {(as.integer(x))})
###########################################
sg_gr<-rowRanges(sgfc_pred)
sg_gr_j<-sg_gr[sg_gr@type=="J"]
#circ_sg_j<-sg_gr_j[sg_gr_j%over%BSJ_gr]
circ_sg_j<-sg_gr_j[unique(queryHits(findOverlaps(sg_gr_j,BSJ_gr,type = "within")))]
count_matrix_j<-count_matrix[circ_sg_j@featureID,]
#get the relative sequences of around a junction
seqs_j<-paste0(seqs[match(start(circ_sg_j),end(full_sg))],seqs[match(end(circ_sg_j),start(full_sg))])
#calculate the scaled read count for junction
junc_rpkm<-RPKM.calc(count_matrix=count_matrix_j, sg=circ_sg_j, bsj_granges =  BSJ_gr, sample_table =  sample_tab
le, feature_type = "j")
deplted_j<-find.depleted.features(junc_rpkm,sample_table,circ_sg_j)
circ_junc<-circ_sg_j[!circ_sg_j@featureID%in%deplted_j]
circ_junc_counts<-junc_rpkm[!circ_sg_j@featureID%in%deplted_j,]
circ_junc_counts[circ_junc_counts==0]<-1
colnames(circ_junc_counts)<-sample_table$sample_name
```

The circRNA exon features are stored in SGRanges format with a corresponding matrix

```
circ_exons[geneID(circ_exons)==1]
```

```
## SGFeatures object with 2 ranges and 0 metadata columns:
##       seqnames      ranges strand     type  splice5p  splice3p featureID
##          <Rle>   <IRanges>  <Rle> <factor> <logical> <logical> <integer>
##   [1]       2L 74903-75018      +        E     FALSE     FALSE     70542
##   [2]       2L 75078-75366      +        E     FALSE     FALSE     70543
##          geneID                                txName        geneName
##       <integer>                       <CharacterList> <CharacterList>
##   [1]         1 FBtr0306540,FBtr0078101,FBtr0302164,...     FBgn0031213
##   [2]         1 FBtr0306540,FBtr0078101,FBtr0302164,...     FBgn0031213
##   -------
##   seqinfo: 1870 sequences from an unspecified genome
```

```
circ_exons_counts[geneID(circ_exons)==1,]
```

```
##       SRR1191323 SRR1191331 SRR1191327 SRR1191335
## 70542         31         29         19          9
## 70543         23         16          8          6
```

Transcript prediction

Transcript prediction is processed one samples at a time. The transcript sets from different samples are then merged.

```
qics_out1<-transcripts.per.sample(sample3_75)
qics_out2<-transcripts.per.sample(sample3_75)
qics_out_final<-merge.qics(qics_out1,qics_out2)
```

Output and Quantification

CYCLeR transcript output

CYCLeR provides 3 forms of output of the annotated transcript: a comprehensive flat file, a GTF-like file, and FASTA file.

```
gtf.table<-prep.output.gtf(qics_out_final,circ_exons)
write.table(qics_out_final[,-9],file = "dm_circles.txt", sep = "\t",row.names = F, col.names = T,quote=F)
qics_out_fa<-DNAStringSet(qics_out_final$seq)
names(qics_out_fa)<-qics_out_final$circID
#if you have a known set of circRNA in FASTA format the CYCLeR output can be combined with it
fasta_circ<-readDNAStringSet("...")
final_ref_fa<-merge.fasta(qics_out_fa,fasta_circ)
writeXStringSet(qics_out_fa,'...')
```

CYCLeR quantification

The final transcript abundance estimation is performed with **kallisto**. An extended and padded circRNA reference sequences are build and combined with linear RNA sequences *kallisto index* is created to be used for any desired sample quantification.

```
#extanding the sequence
sed  '/^[^>]/s/.*\color{#fff}{.*}.*.{30}\color{#fff}{.\{30\}}.{30}/\2\1\2/' circles_seq.fa > circles_seq_extende
d.fa
#Padding the fasta
perl -pe 's/^[^>].*/"N"x159 . "$&"/e' circles_seq_extended.fa > circles_seq_extended_padded.fa
#merging linear and circular sequences
cat linear_transcripts.fa circles_seq_extended_padded.fa > for_kallisto.fa
#Kallisto comands
kallisto index -i kallisto_index -k 31 for_kallisto.fa
kallisto quant -i kallisto_index -o ./ sample1_75_R1.fastq sample2_75_2.fastq
```


1. Bray NL, Pimentel H, Melsted P, Pachter L. Near-optimal probabilistic RNA-seq quantification. Nature Biotechnology. 2016;34:525–7. doi:10.1038/nbt.3519.

2. Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, et al. STAR: Ultrafast universal RNA-seq aligner. Bioinformatics. 2013;29:15–21.

3. Gao Y, Zhang J, Zhao F. Circular RNA identification based on multiple seed matching. Briefings in bioinformatics. 2018;19:803–10.

4. Zhang X-o, Dong R, Zhang Y, Zhang J-l, Luo Z, Zhang J, et al. Diverse alternative back-splicing and alternative splicing landscape of circular RNAs. Genome Research. 2016;1277–87.

5. Goldstein LD, Cao Y, Pau G, Lawrence M, Wu TD, Seshagiri S, et al. Prediction and quantification of splice events from RNA-seq data. PLoS ONE. 2016;11:1–18.

6. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25:2078–9.

7. Liao Y, Smyth GK, Shi W. The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. Nucleic Acids Research. 2019;47.