Working with CYCLeR

Stefan Stefanov

9/14/2022

Contents

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

The computation steps prior and post **CYCLeR** run are most efficiently run on HPC. It is very likely that any HPC in biological institute already has most of those tools installed. Just in case, a **Docker** image containing all the tools is provided.
NOTE: prior to running **Docker** image make sure that ***Docker** is indeed installed and working: https://docs.docker.com/get-started/

- **STAR** - https://github.com/alexdobin/STAR
- **samtools** - https://sourceforge.net/projects/samtools/files/samtools/
- **kallisto** - http://pachterlab.github.io/kallisto/download
- **bwa** (needed for CIRI2) - http://bio-bwa.sourceforge.net/bwa.shtml
- **CIRI2** - https://sourceforge.net/projects/ciri/files/CIRI2/
- **CIRCexplorer2** - https://circexplorer2.readthedocs.io/en/latest/

```
#Docker image with all command line tools
sudo docker pull stiv1n/cycler.prerequisites
```

R packages installation

Option 1: Local installation from GitHub Standard GitHub installation. The dependencies might have compilation issues. For Ubuntu, the issues should be resolved with installation of a few libraries. NOTE: you need a local **samtools** binary for an optional step.

```
#sudo apt update && apt install -y libcurl4-openssl-dev libxml2 libssl-dev \
#libbz2-dev liblzma-dev pkg-config build-essential libglpk40
library(devtools)
install_github("stiv1n/CYCLeR")
```

Option 2: Docker installation The **Docker** use requires you to mount a volume - a working directory () where the output and input would be stored. This container uses **RStudio server** and required login. In this case, the username is *rstudio* the password is *guest*.

```
sudo docker pull stiv1n/cycler
sudo docker run --rm -ti -e PASSWORD=guest -v <local_dir>:/usr/workdir -p 8787:8787 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. Sorting of the file can be performed via **STAR** or **samtools**. NOTE: STAR requires an index and works better with provided GTF. The parameters of STAR index are dependent on the sequencing, so better to read the manual.
An example run for the **Docker** container is shown for **samtools**. My preferred parameters for **STAR**:

```
#STAR parameters
STAR --alignSJoverhangMin 8 --outSAMstrandField intronMotif
--outFilterMismatchNmax 2 --outFilterMismatchNoverLmax 0.1 --chimSegmentMin 15
--chimScoreMin 1 --chimJunctionOverhangMin 15 --chimOutType WithinBAM
--outSAMtype BAM SortedByCoordinate --limitBAMsortRAM 9664623958
--outFilterMultimapNmax 50 --alignIntronMax 100000 --alignIntronMin 15
--seedSearchStartLmax 5 --winAnchorMultimapNmax 200
#Docker run
sudo docker run -v <local_dir>:/usr/local stiv1n/cycler.prerequisites STAR
```

Again, the **Docker** use requires you to mount a volume - a working directory () where the output and input would be stored.

```
#converting the default Aligned.out.sam to a sorted BAM
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  samtools view -u -h /usr/local/Aligned.out.sam | samtools sort \
  -o /usr/local/<name>_sorted.bam
```

BSJ identification

It is advantageous to have input from BSJ identification tools that use different aligners. I suggest **CIRI2** with **bwa** and **CIRCexplorer2** with **STAR**. We have already discussed **STAR** mapping. **CIRI2** requires **bwa** mapping. NOTE: For safety always use full path.

```
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  bwa index -a bwtsw reference.fa
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  bwa mem -T 19 reference.fa read_1.fq read_2.fq > <sample_name>.sam
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  perl /usr/src/myapp/CIRI_v2.0.6/CIRI2.pl -I <sample_name>.sam \
  -O CIRI_<sample_name> -F reference.fa -A annotation.gtf
```

CYCLeR needs just 2 steps of the **CIRCexplorer2** pipeline. NOTE: **CIRCexplore2** uses a flat annotation file

```
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  CIRCexplorer2 parse -t STAR /usr/local/Chimeric.out.junction \
  -b /usr/local/back_spliced_junction.bed
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  CIRCexplorer2 annotate -r annotation.txt -g reference.fa \
  -b /usr/local/back_spliced_junction.bed -o /usr/local/CE_<sample_name>
```

CYCLeR

After all the pre-processing, all the files should preferably be in one folder.

Processing the BAM info in R

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

```
#load BAM files
bam_file_prefix<-system.file("extdata", package = "CYCLeR")
filenames<-c("sample1_75","sample2_75","sample3_75","sample4_75")
BSJ_files_ciri<-paste0(bam_file_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
## 1  sample1_75   control
## 2  sample2_75   control
## 3  sample3_75  enriched
## 4  sample4_75  enriched
##                                                                        file_bam
## 1 /home/stefan/miniconda3/envs/cycler/lib/R/library/CYCLeR/extdata/sample1_75.bam
## 2 /home/stefan/miniconda3/envs/cycler/lib/R/library/CYCLeR/extdata/sample2_75.bam
## 3 /home/stefan/miniconda3/envs/cycler/lib/R/library/CYCLeR/extdata/sample3_75.bam
## 4 /home/stefan/miniconda3/envs/cycler/lib/R/library/CYCLeR/extdata/sample4_75.bam
##   lib_size read_len
## 1    13884       75
```

```
## 2    13959       75
## 3     8494       75
## 4     8637       75
```

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(sample_table$sample_name,BSJ_files_prefix,"CIRI")
colnames(ciri_table)<-c("circ_id", "sample1_75","sample2_75","sample3_75","sample4_75")
ciri_bsjs<-process_BSJs(ciri_table,sample_table)
# i would suggest combine the output of pipelines using different mapping tools
BSJ_files_prefix_CE<-paste0(system.file("extdata", package = "CYCLeR"),"/CE_")
ce_table<-parse_files(sample_table$sample_name,BSJ_files_prefix_CE,"CE")
colnames(ce_table)<-c("circ_id", "sample1_75","sample2_75","sample3_75","sample4_75")
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 3L    24725824 24726292 435115.
## 2 3L    24725824 24728508   9150.
## 3 3L    24728297 24734187 171930.
## 4 3L    24728297 24741000   7992.
## 5 3R    4622509  4628349   64579.
## 6 3R    4626973  4628349  160551.
```

If you use *parse_files* with `input_type=="tsv"`, you can just edit the table with:

```
table_circ<-table_circ%>%separate(circ_id, into=c("chr","start","end","strand"),sep = "_")
```

Transcript assembly

BSJ loci extraction (optional) 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 processed with **samtools** [6]. The trimmed files are also useful for long term local storage.

```
#####################################################
samtools_prefix<-""
trimmed_bams<-filter_bam(BSJ_gr,sample_table,samtools_prefix)
sc@listData[["file_bam"]]<-trimmed_bams
```

Annotation info (optional) The use of the TxDb package is to annotate the identified features. The annotation step is not mandatory, but it does provide useful information. I can also be used to avoid *de novo* feature detection. In the **Docker** container the annotation library is provided.

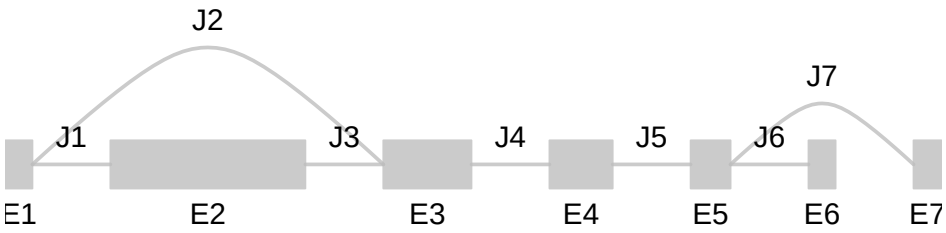
```
#####################################################
#get the gene/transcript info
library("TxDb.Dmelanogaster.UCSC.dm6.ensGene")
#restoreSeqlevels(txdb)
txdb <- TxDb.Dmelanogaster.UCSC.dm6.ensGene
txdb <- keepSeqlevels(txdb, c("chr2L","chr2R","chr3R","chr3L","chr4","chrX","chrY"))
seqlevelsStyle(txdb) <- "Ensembl"
gene_ranges <- genes(txdb)
txf <- convertToTxFeatures(txdb)
#asnnotation as sg-object
sgf <- convertToSGFeatures(txf)
```

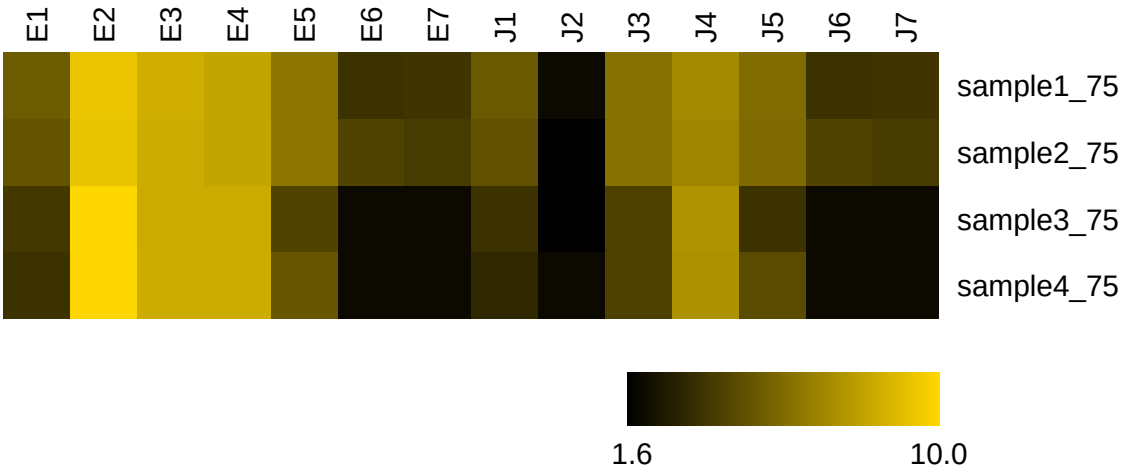
Feature identification with SGSeq The feature detection is based on the SGSeq package. There are 3 options to approach the problem. The default function parameters requires a lot of RAM and processing time. The second option allows using **Rsamtools** `which=BSJ_gr` function, which focuses the reconstruction solely on the pre-selected regions. This significantly speeds up the feature detection and lowers RAM requirements. It is less reliable than the default parameters, however, it is very convenient for a quick test. Additionally, `features=txf` can be provided to indicate that no *de novo* assembly should be done. Naturally, that is the fastest approach, but obviously lacking.

```
#option 1: for fast computer, no RAM limitations, time is not a factor
sgfc_pred <- analyzeFeatures(sc, min_junction_count=2, beta =0.1 ,
                             min_n_sample=1,cores=1,verbose=F)

#option 2: for moderate computer, limited RAM, speed is of the essence
sgfc_pred <- analyzeFeatures(sc, which=BSJ_gr, min_junction_count=2, beta =0.1 ,
                             min_n_sample=1,cores=1,verbose=F)

#option 3: for a toaster with attached monitor
sgfc_pred <- analyzeFeatures(sc, which=BSJ_gr, features=txf, min_junction_count=2,
                             beta =0.1 , min_n_sample=1,cores=1,verbose=F)
#annotation (optional)
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, square=T)
```


J2
J7
J1
J3
J4
J5
J6
E1
E2
E3
E4
E5
E6
E7


E1
E2
E3
E4
E5
E6
E7
J1
J2
J3
J4
J5
J6
J7
sample1_75
sample2_75
sample3_75
sample4_75
1.6
10.0

Re-couting and Processing the features

I prefer the **RSubread** [7] counting method, thus I re-count the identified exon features. Later the **SGseq** counted junctions are used. 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. The simple comparison turns on automatically only due to the lack of replicates.

```
#extract BSJ-corrected splice graphs (sg)
full_sg<-overlap_SG_BSJ(sgfc_pred,BSJ_gr,sgf) #includes linear and circular features
# we have made new feature set so we need to recount the exons
full_fc<-recount_features(full_sg,sample_table)#fc==feature counts
# time to prepare the circular splice graph
```

The sequences of the exons are needed for the subsequent steps. The genome sequence is provided with a **BSgenome** package. For the tutorial the **Docker** image has the needed library provided.

```
#get the correct genome for sequence info
#requires the appropriate BSgenome library
library(BSgenome.Dmelanogaster.UCSC.dm6)
bs_genome=Dmelanogaster
circ_sgfc<-prep_circular_sg(full_sg, full_fc,sgfc_pred, bs_genome, BSJ_gr, th=15)
```

The circRNA exon features are stored in SummarizedExperiment format

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(sgfc=circ_sgfc,BSJ_gr = BSJ_gr,"sample3_75")
qics_out2<-transcripts_per_sample(sgfc=circ_sgfc,BSJ_gr = BSJ_gr,"sample4_75")
qics_out_final<-merge_qics(qics_out1,qics_out2,sgfc_pred)
```

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_sgfc)
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

#prepping the circRNA sequences for quantification
extended_seq<-paste0(qics_out_final$seq,substr(qics_out_final$seq,1,30),
      strrep("N",mean(sc@listData$frag_length[sample_table$treatment=="enriched"])))
qics_out_fa_extended<-DNAStringSet(extended_seq)
names(qics_out_fa_extended)<-qics_out_final$circID
writeXStringSet(qics_out_fa_extended,'circles_seq_extended_padded.fa')
```

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(final_ref_fa,'...')
```

The same function can be used for merging with known linear transcript sequences for the quantification step.

```
fasta_lin<-readDNAStringSet("...")
final_ref_fa<-merge_fasta(qics_out_fa_extended,fasta_lin)
writeXStringSet(final_ref_fa,'for_kallisto.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.

```
#alternative way of merging linear and circular sequences
cat linear_transcripts.fa circles_seq_extended_padded.fa > for_kallisto.fa
#Kallisto comands
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  kallisto index -i kallisto_index -k 31 for_kallisto.fa
sudo docker run  -v <local_dir>:/usr/local stiv1n/cycler.prerequisites \
  kallisto quant -i kallisto_index -o ./ <sample_name>_1.fastq <sample_name>_2.fastq
```

1. Bray NL, Pimentel H, Melsted P, Pachter L. Near-optimal probabilistic RNA-seq quantification. Nature Biotechnology. 2016;34:525–7.
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, Dong R, Zhang Y, Zhang J, 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;11:1–8.
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.