A simple tutorial using R/Bioconductor to analyze RNA sequencing data

Chiachun Chiu

Oct 26, 2015

Download reference genome

Using mouse genome as an example

- NCBI
- Ensembl

Align reads to reference genome (using STAR as aligner)

Downloading and installation of STAR

- Download STAR from github
- Requirement of STAR:
  - x86-64 compatible processors
  - 64 bit Linux/Mac OS X
  - More than 30 GB of RAM

```
git clone https://github.com/alexdobin/STAR.git
cd STAR
make STAR
# To include STAR-fucion
git submodule update --init --recursive
```

Generating genome indices

- The option `--runMode` directs **STAR** to run genome indices generation job.
- The option `--genomeDir` specifies the path to the directory where the genome indices are stored. This directory has to be created before **STAR** run.
- The option `--sjdbOverhang` specifies the length of the genomic sequence around the annotated junction to be used in constructing the splice junction database, where `ReadLength` is the length of the reads.
- Including chromosome/scaffolds/patches is strongly recommended.
- Using annotations:
  - **GTF**
  - **GFF**
  - **user-defined format**
- For very small genome, the parameter `--genomeSAindexNbases` could be set according to the result: `min(14, log2(GenomeLength)/2) - 1`
- For the genome with a large number of references (>5,000), the parameter `--genomeChrBinNbots` should be reduced (set to `min(18, log2(GenomeLength/NumberOfReferences))`) to reduce RAM consumption.

```
refGenomeInfoDir="RefGenome/Ensembl"
genomeVersion="Mus_musculus.GRCm38.82"
annotFile=${genomeVersion}.gtf
refgenomeFile=${genomeVersion}.fa
numThreads=12
ReadLength=75
ReadLength_1=`expr $ReadLength - 1`
runMode="genomeGenerate"
refgenomeParameterDir="ENSEMBL.Mus_musculus.release-82"
readDirs="Whole_Transcriptome"

if [[ ! -d "./$refgenomeParameterDir" ]]; then
  mkdir $refgenomeParameterDir
fi

STAR --runThreadN $numThreads \
    --runMode $runMode \
    --genomeDir ./$refgenomeParameterDir \
    --genomeFastaFiles ./$refGenomeInfoDir/$refgenomeFile \
    --sjdbGTFfile ./$refGenomeInfoDir/$annotFile \
    --sjdbOverhang $ReadLength_1
```

Running mapping jobs

- The option `--readFilesIn` specifies the names of the files containing the sequences to be mapped. **STAR** supports both **FASTA** and **FASTQ** files. If the read files are compressed, use the option `--readFilesCommand UncompressionCommand`.

```
for f in `cat $filelist`;
do
    echo "Now, STAR is processing ${f}.fastq......"
    if [[ $f =~ 'LPS' ]]; then
        exp=`echo $f | awk 'BEGIN {FS="_"} {print $1"_"$2}'`
    else
        exp=`echo $f | awk 'BEGIN {FS="_"} {print $1}'`
    fi

    STAR --genomeDir ./$refgenomeParameterDir \
         --readFilesIn ./$readDirs/$exp/${f}.fastq \
         --runThreadN 12 \
         --outFileNamePrefix Aligned/$f
done
```

Converting SAM files into BAM files using samtools

```
for f in `cat $filelist`;
do
    samtools view -bS Aligned/${f}Aligned.out.sam -o Aligned/${f}.bam
done
```

Locating alignment files

Using Rsamtools to bridge BAM files and R

```
library(Rsamtools)
bamfilenames <- list.files("~/Datasets/NGS_raw_data/Aligned", pattern = "bam")
bamfiles <- BamFileList(bamfilenames)
```

Defining gene models

```
library("GenomicFeatures")
gtffile <- file.path("~/Datasets/NGS_raw_data/RefGenome/Ensembl","Mus_musculus.GRCm38.82.gtf")
txdb <- makeTxDbFromGFF(gtffile, format="gtf", circ_seqs=character())
# Defining a gene by exon
ebg <- exonsBy(txdb, by="gene")
```

Reads counting

```
library("GenomicAlignments")
library("BiocParallel")
# Specifying the parameters of parallel package
snowparam <- SnowParam(workers = 12, type = "SOCK")
register(snowparam)
se <- summarizeOverlaps(feature=ebg, reads=bamfiles, mode="Union", ignore.strand=FALSE)
```

Constructing a experimental design table & performing differential expression analysis

```
library(DESeq2)
sampleData <- data.frame(Treatment=rep("LPS_trt", 24),
                         Time=paste(rep(c(0,1,3,5,7,9), each=4), "h", sep=""),
                         Replicate=paste("r",rep(c(1:4),6),sep=""))
rownames(sampleData) <- rownames(colData(se))
colData(se) <- DataFrame(sampleData)
dds <- DESeqDataSet(se, ~ Time)
dds <- dds[rowSums(counts(dds)) > 1, ]
dds.deseq <- DESeq(dds, test="LRT", reduced= ~1)
results.deseq <- results(dds.deseq)
results.sig.deseq <- subset(results.deseq, padj < 0.05)
```