ChIP-Seq Workflow Template

Author: Daniela Cassol (danielac@ucr.edu) and Thomas Girke (thomas.girke@ucr.edu)

Last update: 21 November, 2019

Package

systemPipeR 1.21.0

Contents

1 Introduction

Users want to provide here background information about the design of their ChIP-Seq project.

1.1 Background and objectives

This report describes the analysis of several ChIP-Seq experiments studying the DNA binding patterns of the transcriptions factors ... from *organism* ....

1.2 Experimental design

Typically, users want to specify here all information relevant for the analysis of their NGS study. This includes detailed descriptions of FASTQ files, experimental design, reference genome, gene annotations, etc.

2 Workflow environment

2.1 Generate workflow environment

Load workflow environment with sample data into your current working directory. The sample data are described here.

```
library(systemPipeRdata)
genWorkenvir(workflow = "chipseq")
setwd("chipseq")
```

Alternatively, this can be done from the command-line as follows:

```
Rscript -e "systemPipeRdata::genWorkenvir(workflow='chipseq')"
```

In the workflow environments generated by `genWorkenvir` all data inputs are stored in a `data/` directory and all analysis results will be written to a separate `results/` directory, while the `systemPipeChIPseq.Rmd` script and the `targets` file are expected to be located in the parent directory. The R session is expected to run from this parent directory. Additional parameter files are stored under `param/`.

To work with real data, users want to organize their own data similarly and substitute all test data for their own data. To rerun an established workflow on new data, the initial `targets` file along with the corresponding FASTQ files are usually the only inputs the user needs to provide.

2.2 Run workflow

Now open the R markdown script `systemPipeChIPseq.Rmd` in your R IDE (*e.g.* vim-r or RStudio) and run the workflow as outlined below.

2.2.1 Run R session on computer node

After opening the `Rmd` file of this workflow in Vim and attaching a connected R session via the `F2` (or other) key, use the following command sequence to run your R session on a computer node.

```
q("no")  # closes R session on head node
```

```
srun --x11 --partition=short --mem=2gb --cpus-per-task 4 --ntasks 1 --time 2:00:00 --pty bash -l
module load R/3.6.0
R
```

Now check whether your R session is running on a computer node of the cluster and assess your environment.

```
system("hostname")  # should return name of a compute node starting with i or c
getwd()  # checks current working directory of R session
dir()  # returns content of current working directory
```

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (H Backman and Girke 2016).

```
library(systemPipeR)
```

If applicable users can load custom functions not provided by `systemPipeR`. Skip this step if this is not the case.

```
source("systemPipeChIPseq_Fct.R")
```

3 Read preprocessing

3.1 Experiment definition provided by `targets` file

The `targets` file defines all FASTQ files and sample comparisons of the analysis workflow.

```
targetspath <- system.file("extdata", "targets_chip.txt", package = "systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")
targets[1:4, -c(5, 6)]
##                     FileName SampleName Factor SampleLong
## 1 ./data/SRR446027_1.fastq.gz        M1A     M1  Mock.1h.A
## 2 ./data/SRR446028_1.fastq.gz        M1B     M1  Mock.1h.B
## 3 ./data/SRR446029_1.fastq.gz        A1A     A1   Avr.1h.A
## 4 ./data/SRR446030_1.fastq.gz        A1B     A1   Avr.1h.B
##   SampleReference
## 1
## 2
## 3             M1A
## 4             M1B
```

3.2 Read quality filtering and trimming

The following example shows how one can design a custom read preprocessing function using utilities provided by the `ShortRead` package, and then apply it with `preprocessReads` in batch mode to all FASTQ samples referenced in the corresponding `SYSargs2` instance (`trim` object below). More detailed information on read preprocessing is provided in `systemPipeR's` main vignette.

First, we construct *SYSargs2* object from *cwl* and *yml* param and *targets* files.

```
dir_path <- system.file("extdata/cwl/preprocessReads/trim-se",
    package = "systemPipeR")
trim <- loadWF(targets = targetspath, wf_file = "trim-se.cwl",
    input_file = "trim-se.yml", dir_path = dir_path)
trim <- renderWF(trim, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
trim
output(trim)[1:2]
```

Next, we execute the code for trimming all the raw data.

```
# args <- systemArgs(sysma='param/trim.param',
# mytargets='targets_chip.txt')
filterFct <- function(fq, cutoff = 20, Nexceptions = 0) {
    qcount <- rowSums(as(quality(fq), "matrix") <= cutoff, na.rm = TRUE)
    fq[qcount <= Nexceptions]
    # Retains reads where Phred scores are >= cutoff with N
    # exceptions
}
preprocessReads(args = trim, Fct = "filterFct(fq, cutoff=20, Nexceptions=0)",
    batchsize = 1e+05)
writeTargetsout(x = trim, file = "targets_chip_trim.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
```

3.3 FASTQ quality report

The following `seeFastq` and `seeFastqPlot` functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The results are written to a PDF file named `fastqReport.pdf`. Parallelization of FASTQ quality report via scheduler (*e.g.* Slurm) across several compute nodes.

```
library(BiocParallel)
library(batchtools)
f <- function(x) {
    library(systemPipeR)
    targets <- system.file("extdata", "targets_chip.txt", package = "systemPipeR")
    dir_path <- system.file("extdata/cwl/preprocessReads/trim-se",
        package = "systemPipeR")
    trim <- loadWorkflow(targets = targets, wf_file = "trim-se.cwl",
        input_file = "trim-se.yml", dir_path = dir_path)
    trim <- renderWF(trim, inputvars = c(FileName = "_FASTQ_PATH1_",
        SampleName = "_SampleName_"))
    seeFastq(fastq = infile1(trim)[x], batchsize = 1e+05, klength = 8)
}

resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
param <- BatchtoolsParam(workers = 4, cluster = "slurm", template = "batchtools.slurm.tmpl",
    resources = resources)
```

```
fqlist <- bplapply(seq(along = trim), f, BPPARAM = param)

pdf("./results/fastqReport.pdf", height = 18, width = 4 * length(fqlist))
seeFastqPlot(unlist(fqlist, recursive = FALSE))
dev.off()
```

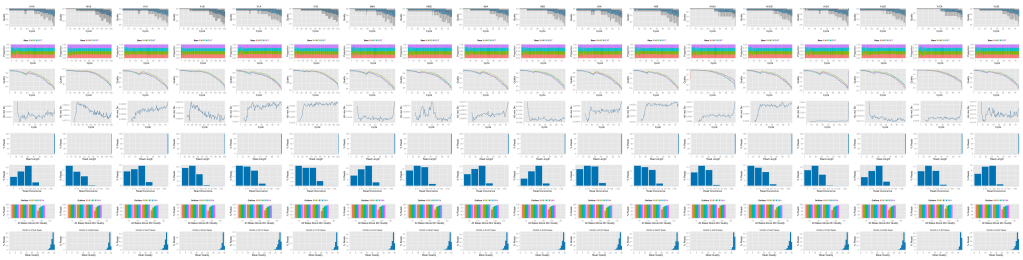

Figure 1: FASTQ quality report for 18 samples

4 Alignments

4.1 Read mapping with `Bowtie2`

The NGS reads of this project will be aligned with `Bowtie2` against the reference genome sequence (Langmead and Salzberg 2012). The parameter settings of the aligner are defined in the `bowtie2-index.cwl` and `bowtie2-index.yml` files. In ChIP-Seq experiments it is usually more appropriate to eliminate reads mapping to multiple locations. To achieve this, users want to remove the argument setting `-k 50 non-deterministic` in the configuration files.

Building the index:

```
dir_path <- system.file("extdata/cwl/bowtie2/bowtie2-idx", package = "systemPipeR")
idx <- loadWorkflow(targets = NULL, wf_file = "bowtie2-index.cwl",
    input_file = "bowtie2-index.yml", dir_path = dir_path)
idx <- renderWF(idx)
idx
cmdlist(idx)

## Run in single machine
runCommandline(idx, make_bam = FALSE)
```

The following submits 18 alignment jobs via a scheduler to a computer cluster.

```
targets <- system.file("extdata", "targets_chip.txt", package = "systemPipeR")
dir_path <- system.file("extdata/cwl/bowtie2/bowtie2-se", package = "systemPipeR")
args <- loadWF(targets = targets, wf_file = "bowtie2-mapping-se.cwl",
    input_file = "bowtie2-mapping-se.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
args
cmdlist(args)[1:2]
output(args)[1:2]
```

```
moduleload(modules(args))  # Skip if a module system is not used
resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
reg <- clusterRun(args, FUN = runCommandline, more.args = list(args = args,
    dir = FALSE), conffile = ".batchtools.conf.R", template = "batchtools.slurm.tmpl",
    Njobs = 18, runid = "01", resourceList = resources)
getStatus(reg = reg)
waitForJobs(reg = reg)
args <- output_update(args, dir = FALSE, replace = TRUE, extension = c(".sam",
    ".bam"))  ## Updates the output(args) to the right location in the subfolders
output(args)
```

Alternatively, one can run the alignments sequentially on a single system.

```
args <- runCommandline(args, force = F)
```

Check whether all BAM files have been created and write out the new targets file.

```
writeTargetsout(x = args, file = "targets_bam.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
file.exists(outpaths)
```

4.2 Read and alignment stats

The following provides an overview of the number of reads in each sample and how many of them aligned to the reference.

```
read_statsDF <- alignStats(args = args)
write.table(read_statsDF, "results/alignStats.xls", row.names = FALSE,
    quote = FALSE, sep = "\t")
read.delim("results/alignStats.xls")
```

4.3 Create symbolic links for viewing BAM files in IGV

The `symLink2bam` function creates symbolic links to view the BAM alignment files in a genome browser such as IGV without moving these large files to a local system. The corresponding URLs are written to a file with a path specified under `urlfile`, here `IGVurl.txt`. Please replace the directory and the user name.

```
symLink2bam(sysargs = args, htmldir = c("~/.html/", "somedir/"),
    urlbase = "http://cluster.hpcc.ucr.edu/~tgirke/", urlfile = "./results/IGVurl.txt")
```

5 Utilities for coverage data

The following introduces several utilities useful for ChIP-Seq data. They are not part of the actual workflow.

5.1 Rle object stores coverage information

```
library(rtracklayer)
library(GenomicRanges)
library(Rsamtools)
library(GenomicAlignments)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
aligns <- readGAlignments(outpaths[1])
cov <- coverage(aligns)
cov
```

5.2 Resizing aligned reads

```
trim(resize(as(aligns, "GRanges"), width = 200))
```

5.3 Naive peak calling

```
islands <- slice(cov, lower = 15)
islands[[1]]
```

5.4 Plot coverage for defined region

```
library(ggbio)
myloc <- c("Chr1", 1, 1e+05)
ga <- readGAlignments(outpaths[1], use.names = TRUE, param = ScanBamParam(which = GRanges(myloc[1],
    IRanges(as.numeric(myloc[2]), as.numeric(myloc[3])))))
autoplot(ga, aes(color = strand, fill = strand), facets = strand ~
    seqnames, stat = "coverage")
```

6 Peak calling with MACS2

6.1 Merge BAM files of replicates prior to peak calling

Merging BAM files of technical and/or biological replicates can improve the sensitivity of the peak calling by increasing the depth of read coverage. The `mergeBamByFactor` function merges BAM files based on grouping information specified by a `factor`, here the `Factor` column of the imported targets file. It also returns an updated `SYSargs2` object containing the paths to the merged BAM files as well as to any unmerged files without replicates. This step can be skipped if merging of BAM files is not desired.

```
dir_path <- system.file("extdata/cwl/mergeBamByFactor", package = "systemPipeR")
args <- loadWF(targets = "targets_bam.txt", wf_file = "merge-bam.cwl",
    input_file = "merge-bam.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_BAM_PATH_",
    SampleName = "_SampleName_"))
```

```
args_merge <- mergeBamByFactor(args = args, overwrite = TRUE)
writeTargetsout(x = args_merge, file = "targets_mergeBamByFactor.txt",
    step = 1, new_col = "FileName", new_col_output_index = 1,
    overwrite = TRUE)
```

```
# Skip if a module system is not used
module("list")
module("unload", "miniconda2")
module("load", "python/2.7.14")  # Make sure to set up your enviroment variable for MACS2
```

6.2 Peak calling without input/reference sample

MACS2 can perform peak calling on ChIP-Seq data with and without input samples (Zhang et al. 2008). The following performs peak calling without input on all samples specified in the corresponding `args` object. Note, due to the small size of the sample data, MACS2 needs to be run here with the `nomodel` setting. For real data sets, users want to remove this parameter in the corresponding `*.param` file(s).

```
dir_path <- system.file("extdata/cwl/MACS2/MACS2-noinput/", package = "systemPipeR")
args <- loadWF(targets = "targets_mergeBamByFactor.txt", wf_file = "macs2.cwl",
    input_file = "macs2.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))

runCommandline(args, make_bam = FALSE, force = T)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
file.exists(outpaths)
writeTargetsout(x = args, file = "targets_macs.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
```

6.3 Peak calling with input/reference sample

To perform peak calling with input samples, they can be most conveniently specified in the `SampleReference` column of the initial `targets` file. The `writeTargetsRef` function uses this information to create a `targets` file intermediate for running MACS2 with the corresponding input samples.

```
writeTargetsRef(infile = "targets_mergeBamByFactor.txt", outfile = "targets_bam_ref.txt",
    silent = FALSE, overwrite = TRUE)
dir_path <- system.file("extdata/cwl/MACS2/MACS2-input/", package = "systemPipeR")
args_input <- loadWF(targets = "targets_bam_ref.txt", wf_file = "macs2-input.cwl",
    input_file = "macs2.yml", dir_path = dir_path)
args_input <- renderWF(args_input, inputvars = c(FileName1 = "_FASTQ_PATH1_",
    FileName2 = "_FASTQ_PATH2_", SampleName = "_SampleName_"))
cmdlist(args_input)[1]
## Run
args_input <- runCommandline(args_input, make_bam = FALSE, force = T)
outpaths_input <- subsetWF(args_input, slot = "output", subset = 1,
    index = 1)
file.exists(outpaths_input)
```

```
writeTargetsout(x = args_input, file = "targets_macs_input.txt",
    step = 1, new_col = "FileName", new_col_output_index = 1,
    overwrite = TRUE)
```

The peak calling results from MACS2 are written for each sample to separate files in the `results` directory. They are named after the corresponding files with extensions used by MACS2.

6.4 Identify consensus peaks

The following example shows how one can identify consensus preaks among two peak sets sharing either a minimum absolute overlap and/or minimum relative overlap using the `subsetByOverlaps` or `olRanges` functions, respectively. Note, the latter is a custom function imported below by sourcing it.

```
# source('http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/rangeoverlapper.R')
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)  ## escolher um dos outputs index
peak_M1A <- outpaths["M1A"]
peak_M1A <- as(read.delim(peak_M1A, comment = "#")[, 1:3], "GRanges")
peak_A1A <- outpaths["A1A"]
peak_A1A <- as(read.delim(peak_A1A, comment = "#")[, 1:3], "GRanges")
(myol1 <- subsetByOverlaps(peak_M1A, peak_A1A, minoverlap = 1))
# Returns any overlap
myol2 <- olRanges(query = peak_M1A, subject = peak_A1A, output = "gr")
# Returns any overlap with OL length information
myol2[values(myol2)["OLpercQ"][, 1] >= 50]
# Returns only query peaks with a minimum overlap of 50%
```

7 Annotate peaks with genomic context

7.1 Annotation with `ChIPpeakAnno` package

The following annotates the identified peaks with genomic context information using the `ChIPpeakAnno` and `ChIPseeker` packages, respectively (Zhu et al. 2010; Yu, Wang, and He 2015).

```
library(ChIPpeakAnno)
library(GenomicFeatures)
dir_path <- system.file("extdata/cwl/annotate_peaks", package = "systemPipeR")
args <- loadWF(targets = "targets_macs.txt", wf_file = "annotate-peaks.cwl",
    input_file = "annotate-peaks.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))

txdb <- makeTxDbFromGFF(file = "data/tair10.gff", format = "gff",
    dataSource = "TAIR", organism = "Arabidopsis thaliana")
ge <- genes(txdb, columns = c("tx_name", "gene_id", "tx_type"))
for (i in seq(along = args)) {
    peaksGR <- as(read.delim(infile1(args)[i], comment = "#"),
        "GRanges")
```

```
    annotatedPeak <- annotatePeakInBatch(peaksGR, AnnotationData = genes(txdb))
    df <- data.frame(as.data.frame(annotatedPeak), as.data.frame(values(ge[values(annotatedPeak)$feature,
        ])))
    outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
    write.table(df, outpaths[i], quote = FALSE, row.names = FALSE,
        sep = "\t")
}
writeTargetsout(x = args, file = "targets_peakanno.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
```

The peak annotation results are written for each peak set to separate files in the `results` directory. They are named after the corresponding peak files with extensions specified in the `annotate_peaks.param` file, here `*.peaks.annotated.xls`.

7.2 Annotation with `ChIPseeker` package

Same as in previous step but using the `ChIPseeker` package for annotating the peaks.

```
library(ChIPseeker)
for (i in seq(along = args)) {
    peakAnno <- annotatePeak(infile1(args)[i], TxDb = txdb, verbose = FALSE)
    df <- as.data.frame(peakAnno)
    outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
    write.table(df, outpaths[i], quote = FALSE, row.names = FALSE,
        sep = "\t")
}
writeTargetsout(x = args, file = "targets_peakanno.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
```

Summary plots provided by the `ChIPseeker` package. Here applied only to one sample for demonstration purposes.

```
peak <- readPeakFile(infile1(args)[1])
covplot(peak, weightCol = "X.log10.pvalue.")
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
peakHeatmap(outpaths[1], TxDb = txdb, upstream = 1000, downstream = 1000,
    color = "red")
plotAvgProf2(outpaths[1], TxDb = txdb, upstream = 1000, downstream = 1000,
    xlab = "Genomic Region (5'->3')", ylab = "Read Count Frequency")
```

8 Count reads overlapping peaks

The `countRangeset` function is a convenience wrapper to perform read counting iteratively over serveral range sets, here peak range sets. Internally, the read counting is performed with the `summarizeOverlaps` function from the `GenomicAlignments` package. The resulting count tables are directly saved to files, one for each peak set.

```
library(GenomicRanges)
dir_path <- system.file("extdata/cwl/count_rangesets", package = "systemPipeR")
args <- loadWF(targets = "targets_macs.txt", wf_file = "count_rangesets.cwl",
```

```
    input_file = "count_rangesets.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))

## Bam Files
targets <- system.file("extdata", "targets_chip.txt", package = "systemPipeR")
dir_path <- system.file("extdata/cwl/bowtie2/bowtie2-se", package = "systemPipeR")
args_bam <- loadWF(targets = targets, wf_file = "bowtie2-mapping-se.cwl",
    input_file = "bowtie2-mapping-se.yml", dir_path = dir_path)
args_bam <- renderWF(args_bam, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
args_bam <- output_update(args_bam, dir = FALSE, replace = TRUE,
    extension = c(".sam", ".bam"))
outpaths <- subsetWF(args_bam, slot = "output", subset = 1, index = 1)

bfl <- BamFileList(outpaths, yieldSize = 50000, index = character())
countDFnames <- countRangeset(bfl, args, mode = "Union", ignore.strand = TRUE)
writeTargetsout(x = args, file = "targets_countDF.txt", step = 1,
    new_col = "FileName", new_col_output_index = 1, overwrite = TRUE)
```

9 Differential binding analysis

The `runDiff` function performs differential binding analysis in batch mode for several count tables using `edgeR` or `DESeq2` (Robinson, McCarthy, and Smyth 2010; Love, Huber, and Anders 2014). Internally, it calls the functions `run_edgeR` and `run_DESeq2`. It also returns the filtering results and plots from the downstream `filterDEGs` function using the fold change and FDR cutoffs provided under the `dbrfilter` argument.

```
dir_path <- system.file("extdata/cwl/rundiff", package = "systemPipeR")
args_diff <- loadWF(targets = "targets_countDF.txt", wf_file = "rundiff.cwl",
    input_file = "rundiff.yml", dir_path = dir_path)
args_diff <- renderWF(args_diff, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))

cmp <- readComp(file = args_bam, format = "matrix")
dbrlist <- runDiff(args = args_diff, diffFct = run_edgeR, targets = targets.as.df(targets(args_bam)),
    cmp = cmp[[1]], independent = TRUE, dbrfilter = c(Fold = 2,
        FDR = 1))
writeTargetsout(x = args_diff, file = "targets_rundiff.txt",
    step = 1, new_col = "FileName", new_col_output_index = 1,
    overwrite = TRUE)
```

10 GO term enrichment analysis

The following performs GO term enrichment analysis for each annotated peak set.

```
dir_path <- system.file("extdata/cwl/annotate_peaks", package = "systemPipeR")
args <- loadWF(targets = "targets_bam_ref.txt", wf_file = "annotate-peaks.cwl",
    input_file = "annotate-peaks.yml", dir_path = dir_path)
```

```
args <- renderWF(args, inputvars = c(FileName1 = "_FASTQ_PATH1_",
    FileName2 = "_FASTQ_PATH2_", SampleName = "_SampleName_"))

args_anno <- loadWF(targets = "targets_macs.txt", wf_file = "annotate-peaks.cwl",
    input_file = "annotate-peaks.yml", dir_path = dir_path)
args_anno <- renderWF(args_anno, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))
annofiles <- subsetWF(args_anno, slot = "output", subset = 1,
    index = 1)
gene_ids <- sapply(names(annofiles), function(x) unique(as.character(read.delim(annofiles[x])[,
    "geneId"])), simplify = FALSE)
load("data/GO/catdb.RData")
BatchResult <- GOCluster_Report(catdb = catdb, setlist = gene_ids,
    method = "all", id_type = "gene", CLSZ = 2, cutoff = 0.9,
    gocats = c("MF", "BP", "CC"), recordSpecGO = NULL)
```

11 Motif analysis

11.1 Parse DNA sequences of peak regions from genome

Enrichment analysis of known DNA binding motifs or *de novo* discovery of novel motifs requires the DNA sequences of the identified peak regions. To parse the corresponding sequences from the reference genome, the `getSeq` function from the `Biostrings` package can be used. The following example parses the sequences for each peak set and saves the results to separate FASTA files, one for each peak set. In addition, the sequences in the FASTA files are ranked (sorted) by increasing p-values as expected by some motif discovery tools, such as `BCRANK`.

```
library(Biostrings)
library(seqLogo)
library(BCRANK)
dir_path <- system.file("extdata/cwl/annotate_peaks", package = "systemPipeR")
args <- loadWF(targets = "targets_macs.txt", wf_file = "annotate-peaks.cwl",
    input_file = "annotate-peaks.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_",
    SampleName = "_SampleName_"))

rangefiles <- infile1(args)
for (i in seq(along = rangefiles)) {
    df <- read.delim(rangefiles[i], comment = "#")
    peaks <- as(df, "GRanges")
    names(peaks) <- paste0(as.character(seqnames(peaks)), "_",
        start(peaks), "-", end(peaks))
    peaks <- peaks[order(values(peaks)$X.log10.pvalue., decreasing = TRUE)]
    pseq <- getSeq(FaFile("./data/tair10.fasta"), peaks)
    names(pseq) <- names(peaks)
    writeXStringSet(pseq, paste0(rangefiles[i], ".fasta"))
}
```

11.2 Motif discovery with BCRANK

The Bioconductor package BCRANK is one of the many tools available for *de novo* discovery of DNA binding motifs in peak regions of ChIP-Seq experiments. The given example applies this method on the first peak sample set and plots the sequence logo of the highest ranking motif.

```
set.seed(0)
BCRANKout <- bcrank(paste0(rangefiles[1], ".fasta"), restarts = 25,
    use.P1 = TRUE, use.P2 = TRUE)
toptable(BCRANKout)
topMotif <- toptable(BCRANKout, 1)
weightMatrix <- pwm(topMotif, normalize = FALSE)
weightMatrixNormalized <- pwm(topMotif, normalize = TRUE)
pdf("results/seqlogo.pdf")
seqLogo(weightMatrixNormalized)
dev.off()
```

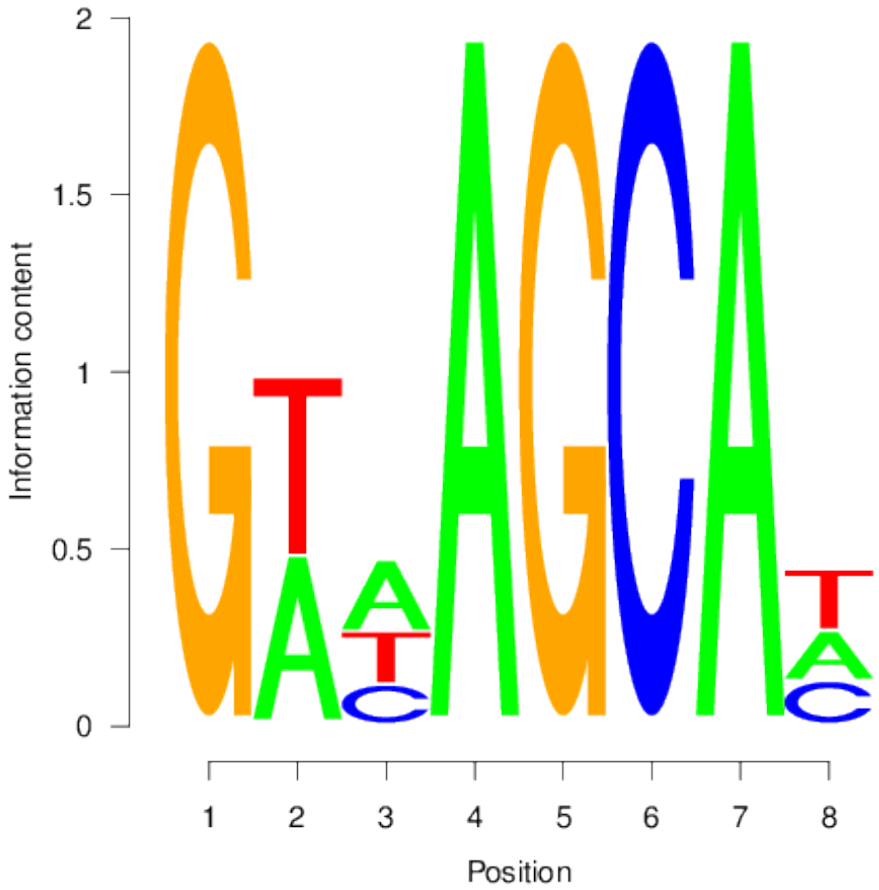

Figure 2: One of the motifs identified by BCRANK

12 Version Information

```
sessionInfo()
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Pop!_OS 19.04
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] systemPipeR_1.21.0          ShortRead_1.44.0
##  [3] GenomicAlignments_1.22.1    SummarizedExperiment_1.16.0
##  [5] DelayedArray_0.12.0         matrixStats_0.55.0
##  [7] Biobase_2.46.0              BiocParallel_1.20.0
##  [9] Rsamtools_2.2.1             Biostrings_2.54.0
## [11] XVector_0.26.0              GenomicRanges_1.38.0
## [13] GenomeInfoDb_1.22.0         IRanges_2.20.1
## [15] S4Vectors_0.24.0            BiocGenerics_0.32.0
## [17] BiocStyle_2.14.0
##
## loaded via a namespace (and not attached):
##  [1] Category_2.52.1          bitops_1.0-6
##  [3] bit64_0.9-7              RColorBrewer_1.1-2
##  [5] progress_1.2.2           httr_1.4.1
##  [7] Rgraphviz_2.30.0         tools_3.6.1
##  [9] backports_1.1.5          R6_2.4.1
## [11] DBI_1.0.0                lazyeval_0.2.2
## [13] colorspace_1.4-1         withr_2.1.2
## [15] tidyselect_0.2.5         prettyunits_1.0.2
## [17] bit_1.1-14               curl_4.2
## [19] compiler_3.6.1           graph_1.64.0
## [21] formatR_1.7              rtracklayer_1.46.0
## [23] bookdown_0.15            checkmate_1.9.4
## [25] scales_1.1.0             genefilter_1.68.0
## [27] RBGL_1.62.1              askpass_1.1
## [29] rappdirs_0.3.1           stringr_1.4.0
```

```
## [31] digest_0.6.22            rmarkdown_1.17
## [33] AnnotationForge_1.28.0   pkgconfig_2.0.3
## [35] htmltools_0.4.0          BSgenome_1.54.0
## [37] dbplyr_1.4.2             limma_3.42.0
## [39] rlang_0.4.1              RSQLite_2.1.2
## [41] GOstats_2.52.0           hwriter_1.3.2
## [43] dplyr_0.8.3              VariantAnnotation_1.32.0
## [45] RCurl_1.95-4.12          magrittr_1.5
## [47] GO.db_3.10.0             GenomeInfoDbData_1.2.2
## [49] Matrix_1.2-17            Rcpp_1.0.3
## [51] munsell_0.5.0            lifecycle_0.1.0
## [53] stringi_1.4.3            yaml_2.2.0
## [55] edgeR_3.28.0             zlibbioc_1.32.0
## [57] BiocFileCache_1.10.2     grid_3.6.1
## [59] blob_1.2.0               crayon_1.3.4
## [61] lattice_0.20-38          splines_3.6.1
## [63] GenomicFeatures_1.38.0   annotate_1.64.0
## [65] hms_0.5.2                batchtools_0.9.11
## [67] locfit_1.5-9.1           zeallot_0.1.0
## [69] knitr_1.26               pillar_1.4.2
## [71] rjson_0.2.20             base64url_1.4
## [73] codetools_0.2-16         biomaRt_2.42.0
## [75] XML_3.98-1.20            glue_1.3.1
## [77] evaluate_0.14            latticeExtra_0.6-28
## [79] data.table_1.12.6        BiocManager_1.30.10
## [81] vctrs_0.2.0              gtable_0.3.0
## [83] openssl_1.4.1            purrr_0.3.3
## [85] assertthat_0.2.1         ggplot2_3.2.1
## [87] xfun_0.11                xtable_1.8-4
## [89] survival_2.44-1.1        pheatmap_1.0.12
## [91] tibble_2.1.3             AnnotationDbi_1.48.0
## [93] memoise_1.1.0            brew_1.0-6
## [95] GSEABase_1.48.0
```

13 Funding

This project was supported by funds from the National Institutes of Health (NIH) and the National Science Foundation (NSF).

References

H Backman, Tyler W, and Thomas Girke. 2016. "systemPipeR: NGS workflow and report generation environment." *BMC Bioinformatics* 17 (1): 388. https://doi.org/10.1186/s12859-016-1241-0.

Langmead, Ben, and Steven L Salzberg. 2012. "Fast Gapped-Read Alignment with Bowtie 2." *Nat. Methods* 9 (4). Nature Publishing Group: 357–59. https://doi.org/10.1038/nmeth.1923.

Love, Michael, Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for RNA-seq Data with DESeq2." *Genome Biol.* 15 (12): 550. https://doi.org/10.1186/s13059-014-0550-8.

Robinson, M D, D J McCarthy, and G K Smyth. 2010. "EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40. https://doi.org/10.1093/bioinformatics/btp616.

Yu, Guangchuang, Li-Gen Wang, and Qing-Yu He. 2015. "ChIPseeker: An R/Bioconductor Package for ChIP Peak Annotation, Comparison and Visualization." *Bioinformatics* 31 (14): 2382–3. https://doi.org/10.1093/bioinformatics/btv145.

Zhang, Y, T Liu, C A Meyer, J Eeckhoute, D S Johnson, B E Bernstein, C Nussbaum, et al. 2008. "Model-Based Analysis of ChIP-Seq (MACS)." *Genome Biol.* 9 (9). https://doi.org/10.1186/gb-2008-9-9-r137.

Zhu, Lihua J, Claude Gazin, Nathan D Lawson, Hervé Pagès, Simon M Lin, David S Lapointe, and Michael R Green. 2010. "ChIPpeakAnno: A Bioconductor Package to Annotate ChIP-seq and ChIP-chip Data." *BMC Bioinformatics* 11: 237. https://doi.org/10.1186/1471-2105-11-237.