Analysis of ChIP-Seq Data with R/Bioconductor

...

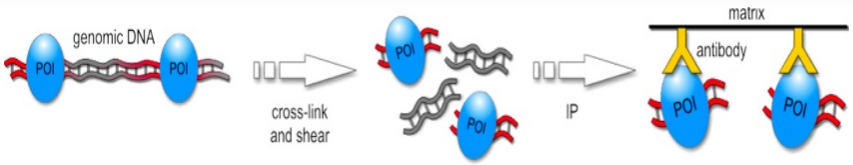
Thomas Girke

December 9, 2012

Outline

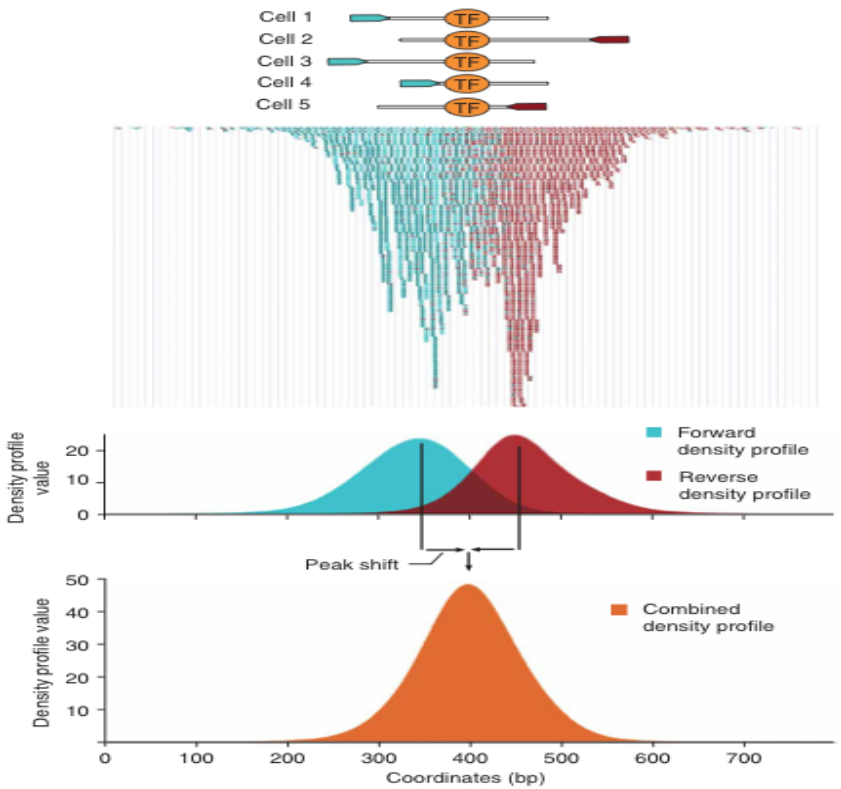
Outline

ChIP-Seq Technology



⇓

ChIP-Seq



ChIP-Seq Workflow

- Read mapping
- Peak calling
- Peak annotation/filtering
- Differential peak analysis
- Motif enrichment analysis in sequences under peaks

Peak Callers (Command-line Tools)

- CisGenome
- ERANGE
- FindPeaks
- F-Seq
- GLITR
- MACS
- PeakSeq
- QuEST
- SICER
- SiSSRs
- spp
- USeq
- ...

Pepke, S, Wold, B, Mortazavi, A (2009) Computation for ChIP-seq and RNA-seq studies. Nat Methods 6, 22-32. Link

Outline

General Purpose Resources for ChIP-Seq Analysis in R

- GenomicRanges Link: high-level infrastructure for range data
- Rsamtools Link: BAM support
- rtracklayer Link: Annotation imports, interface to online genome browsers
- DESeq Link: RNA-Seq analysis
- edgeR Link: RNA-Seq analysis
- chipseq Link: Utilities for ChIP-Seq analysis
- ChIPpeakAnno Link: Annotating peaks with genome context information
- ...

Peak Calling in R

- BayesPeak Link: hidden Markov models (HMM) and Bayesian statistics
- PICS Link: probabilistic inference
- DiffBind Link: Differential binding analysis of ChIP-Seq peak data
- MOSAiCS Link: model-based analysis of ChIP-Seq data
- iSeq Link: Hidden Ising Models
- ChIPseqR Link
- CSAR Link: tests based on Poisson distribution
- ChIP-Seq Link
- SPP Link

Outline

Outline

Data Sets and Experimental Variables

- To make the following sample code work, users can download the sample data Link into the directory of their current R session.
- It contains slimmed down versions of four FASTQ files from the ChIP-Seq experiment published by Kaufman et al (2010, GSE20176 Link), a shortened GFF3 annotation file Link and the corresponding reference genome from *Arabidopsis thaliana*.

Kaufmann et al (2010) Orchestration of floral initiation by APETALA1. Science 328, 85-89. Link

Outline

Align Reads and Output Indexed Bam Files

Note: Rsubread is Linux only. OS X/Windows users want to skip the mapping step and download the Bam files from here: Link

```
> library(Rsubread); library(Rsamtools)
> dir.create("results") # Note: all output data will be written to directory 'results'
> buildindex(basename="./results/tair10chr.fasta", reference="./data/tair10chr.fasta") # Build indexed refe
> targets <- read.delim("./data/targets.txt") # Import experiment design information
> targets
> input <- paste("./data/", targets[,3], sep="")
> output <- paste("./results/", targets[,3], ".sam", sep="")
> reference <- "./results/tair10chr.fasta"
> for(i in seq(along=targets[,3])) {
+         align(index=reference, readfile1=input[i], output_file=output[i], nthreads=8, indels=1, TH1=2)
+         asBam(file=output[i], destination=gsub(".sam", "", output[i]), overwrite=TRUE, indexDestination=T
+         unlink(output[i])
+ }
```

Outline

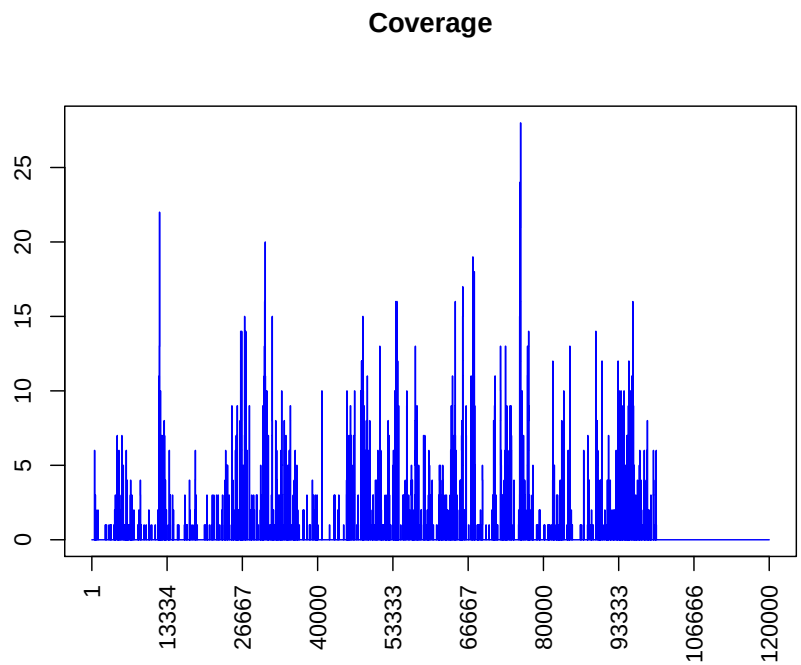
Important Resources for ChIP-Seq Analysis

Coverage and peak slicing

```
> library(rtracklayer); library(GenomicRanges); library(Rsamtools)
> targets <- read.delim("./data/targets.txt") # Import experiment design information
> targets

   Samples Factor           Fastq
1  AP1IND1   sig1 SRR038845.fastq
2  AP1IND2   sig1 SRR038846.fastq
3 AP1UIND1   bgr1 SRR038848.fastq
4 AP1UIND2   bgr1 SRR038850.fastq

> samples <- as.character(targets$Fastq)
> samplespath <- paste("./results/", samples, ".bam", sep="")
> aligns <- readBamGappedAlignments(samplespath[3])
> cov <- coverage(aligns)
> islands <- slice(cov, lower = 15)
> islands[[1]][1:12,]

Views on a 30427671-length Rle subject
views:
     start   end width
 [1] 11998 12003     6 [22 22 22 22 22 22]
 [2] 27064 27077    14 [15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [3] 30619 30623     5 [16 16 16 16 16]
 [4] 30709 30725    17 [16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 20 ...]
 [5] 31956 31975    20 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 ...]
 [6] 48015 48029    15 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [7] 53839 53844     6 [16 16 16 16 16 16]
 [8] 54031 54066    36 [16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 ...]
 [9] 64384 64394    11 [16 16 16 16 16 16 16 16 16 15 15]
[10] 65692 65719    28 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 ...]
[11] 67505 67508     4 [19 19 19 19]
[12] 67743 67770    28 [18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 ...]
```

Coverage Plot

```
> plotCov <- function(mycov=cov, mychr=1, mypos=c(1,1000), mymain="Coverage", ...) {
+     op <- par(mar=c(8,3,6,1))
+     plot(as.numeric(mycov[[mychr]][mypos[1]:mypos[2]]), type="l",
+     lwd=1, col="blue", ylab="", main=mymain, xlab="", xaxt="n", ...)
+     axis(1, las = 2, at=seq(1,mypos[2]-mypos[1], length.out= 10),
+     labels=as.integer(seq(mypos[1], mypos[2], length.out= 10)))
+     par(op)
+ }
> plotCov(mycov=cov, mychr="Chr1", mypos=c(1,120000)) # Remember: read data is trunctated to first 100kbp
```



Import Aligned Read Data

Import aligned reads (bam files) and extend to 200bp

```
> chip_signal_list <- sapply(samplespath, list)
> for(i in seq(along=samplespath)) {
+         aligns <- readBamGappedAlignments(samplespath[i])
+         chip_signal_list[[i]] <- as(aligns, "GRanges")
+ }
> chip_signal_list[["./results/SRR038845.fastq.bam"]][1:4,]

GRanges with 4 ranges and 0 metadata columns:
      seqnames     ranges strand
         <Rle>  <IRanges>  <Rle>
  [1]     Chr1 [121, 156]      -
  [2]     Chr1 [121, 156]      -
  [3]     Chr1 [216, 251]      +
  [4]     Chr1 [295, 330]      +
  ---
  seqlengths:
       Chr1     Chr2     Chr3     Chr4     Chr5     ChrC     ChrM
   30427671 19698289 23459830 18585056 26975502   154478   366924

> chip_signal_list <- sapply(names(chip_signal_list), function(x) resize(chip_signal_list[[x]], width = 200
```

Outline

Naive Peak Calling by Coverage Value

Compute coverage and call peaks

```
> Nreads <- sapply(names(chip_signal_list), function(x)  length(chip_signal_list[[x]]))
> normfactor <- 10^6/Nreads
> chip_signal_list <- sapply(names(chip_signal_list), function(x) coverage(chip_signal_list[[x]]) * normfac
> chip_signal_list[["./results/SRR038845.fastq.bam"]][1:2,]

SimpleRleList of length 2
$Chr1
numeric-Rle of length 30427671 with 4702 runs
  Lengths:              139               17 ...             1248
  Values : 6.42287571414349  12.845751428287 ...                0
$Chr2
numeric-Rle of length 19698289 with 21348 runs
  Lengths:              834                1 ...           265395
  Values :                0 6.42287571414349 ...                0

> chip_peak_list <- sapply(names(chip_signal_list), function(x) slice(chip_signal_list[[x]], lower=5))
> chip_peak_list[[1]][[1]][1:3]

Views on a 30427671-length Rle subject
views:
    start  end width
[1]     1  689   689 [6.422876 6.422876 6.422876 6.422876 6.422876 ...]
[2]   764  963   200 [9.634314 9.634314 9.634314 9.634314 9.634314 ...]
[3]  1463 1697   235 [12.84575 12.84575 12.84575 12.84575 12.84575 ...]
```

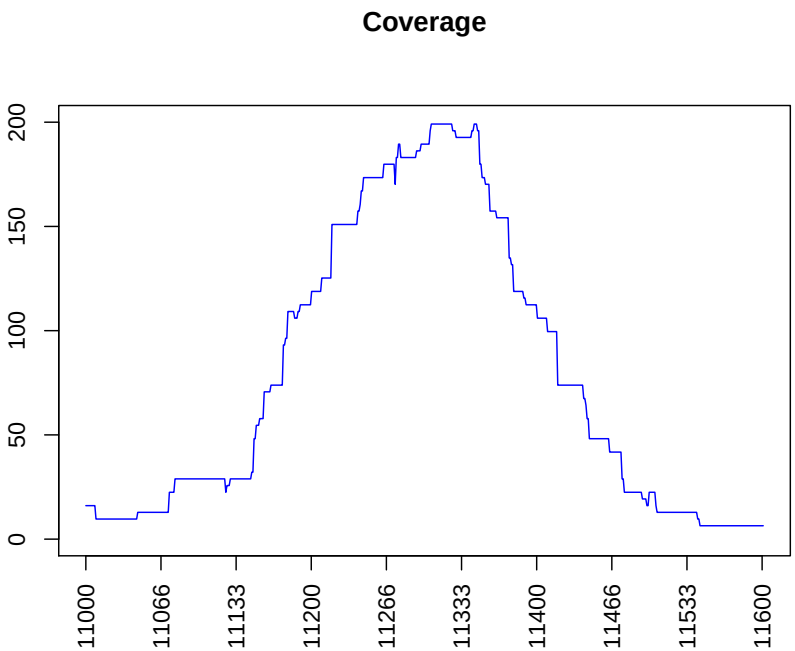
Peak Calling with BayesPeak

Compute coverage and call peaks

```
> library(BayesPeak)
> sig <- readBamGappedAlignments(samplespath[1])
> bgr <- readBamGappedAlignments(samplespath[3])
> sig <- as(as(sig, "GRanges"), "RangedData")
> bgr <- as(as(bgr, "GRanges"), "RangedData")
> raw.output <- bayespeak(treatment=sig, control=bgr, start = 1, end = 100000)

..............

> # unreliable.jobs <- log(raw.output$QC$lambda1) < 1.5 # Removal of false positives due to overfitting.
> # bpeaks <- as.data.frame(summarise.peaks(raw.output, method = "lowerbound", exclude.jobs = unreliable.jo
> bpeaks <- as.data.frame(summarise.peaks(raw.output, method = "lowerbound"))
> source("./data/Fct/chipseqFct.R") # Imports the rangeCoverage function.
> sigcovDF <- rangeCoverage(summaryFct=viewMeans, myname="sig_", peaksIR=bpeaks[,1:3], sig=sig, readextend=
> bgrcovDF <- rangeCoverage(summaryFct=viewMeans, myname="bgr_", peaksIR=bpeaks[,1:3], sig=bgr, readextend=
> bpeaksDF <- cbind(bpeaks, sigcovDF[,-1], bgrcovDF[,-1])
> bpeaksDF[1:4,]

  space start   end width        PP   sig_cov sig_cov.pos sig_cov.neg
1  Chr1  8301  8401   101 0.5537269 0.1907785   0.1271857  0.06359283
2  Chr1 11151 11351   201 0.9998927 0.6390921   0.5911602  0.04793191
3  Chr1 13601 13751   151 0.9998833 0.4040882   0.2339458  0.17014240
4  Chr1 16601 17051   451 0.9999917 0.4699665   0.1922590  0.27770749
      bgr_cov bgr_cov.pos bgr_cov.neg
1 0.039603881 0.039603881 0.000000000
2 0.009950229 0.009950229 0.000000000
3 0.092715046 0.079470040 0.013245007
4 0.008869162 0.004434581 0.004434581
```

Coverage Plot

```
> plotCov(mycov=chip_signal_list[[1]], mychr="Chr1", mypos=c(11000, 11600), ylim=c(0,200))
```



Identify Common Peaks Among Two Methods

Compares results from simple cutoff method with BayesPeak results

```
> simple_peak <- as.data.frame(as(chip_peak_list[[1]], "IRangesList"))
>    # simple_peak <- as.data.frame(chip_peak_list[[1]])
> commonpeaks <- subsetByOverlaps(as(bpeaks, "RangedData"), as(simple_peak, "RangedData"), minoverlap=100)
> bpeaksDF[bpeaksDF$start %in% start(commonpeaks),][1:4,]

  space start   end width        PP   sig_cov sig_cov.pos sig_cov.neg
1  Chr1  8301  8401   101 0.5537269 0.1907785   0.1271857  0.06359283
2  Chr1 11151 11351   201 0.9998927 0.6390921   0.5911602  0.04793191
3  Chr1 13601 13751   151 0.9998833 0.4040882   0.2339458  0.17014240
4  Chr1 16601 17051   451 0.9999917 0.4699665   0.1922590  0.27770749
      bgr_cov bgr_cov.pos bgr_cov.neg
1 0.039603881 0.039603881 0.000000000
2 0.009950229 0.009950229 0.000000000
3 0.092715046 0.079470040 0.013245007
4 0.008869162 0.004434581 0.004434581
```

Exercise 1: Compare Results with Published Peaks

Task 1 Import peaks predicted by Kaufmann et al (2010).

Task 2 Determine how many of the published peaks have at least a 50% length overlap with the results from the BayesPeak and the naive peak calling methods.

Required information:

```
> pubpeaks <- read.delim("./data/Kaufmann_peaks100k.txt") # Published peaks for first 100kbp on chromosomes
> pubpeaks <- pubpeaks[order(pubpeaks$space, pubpeaks$start),]
> pubpeaks[1:4,]

              PeakID space start   end score_position     score length
chr1_3132   chr1_3132  Chr1  3094  3172           3132  4.656515     79
chr1_8365   chr1_8365  Chr1  8222  8425           8365 11.046212    204
chr1_11298 chr1_11298  Chr1 11149 11440          11298 52.109592    292
chr1_13686 chr1_13686  Chr1 13602 13756          13686  5.341907    155

> # Import olRanges function, which accepts two GRranges (IRanges) objects
> source("./data/Fct/rangeoverlapper.R")
```

Outline

Import Annotation Data from GFF

Annotation data from GFF

```
> library(rtracklayer); library(GenomicRanges); library(Rsamtools)
> gff <- import.gff("./data/TAIR10_GFF3_trunc.gff", asRangedData=FALSE)
> seqlengths(gff) <- end(ranges(gff[which(elementMetadata(gff)[,"type"]=="chromosome"),]))
> subgene_index <- which(elementMetadata(gff)[,"type"] == "gene")
> gffsub <- gff[subgene_index,] # Returns only gene ranges
> strand(gffsub) <- "*" # For strand insensitive analysis
> gffsub[1:4,1:2]

GRanges with 4 ranges and 2 metadata columns:
      seqnames         ranges strand |   source     type
         <Rle>      <IRanges>  <Rle> | <factor> <factor>
  [1]     Chr1 [ 3631,  5899]      * |   TAIR10     gene
  [2]     Chr1 [ 5928,  8737]      * |   TAIR10     gene
  [3]     Chr1 [11649, 13714]      * |   TAIR10     gene
  [4]     Chr1 [23146, 31227]      * |   TAIR10     gene
  ---
  seqlengths:
       Chr1     Chr2     Chr3     Chr4     Chr5     ChrC     ChrM
   30427671 19698289 23459830 18585056 26975502   154478   366924

> ids <- elementMetadata(gffsub)[, "group"]
> gffgene <- gffsub
> gffsub <- split(gffsub, ids) # Coerce to GRangesList
```

Annotate Peaks with ChIPpeakAnno

```
> library(ChIPpeakAnno)
> annoRD <- unlist(gffsub)
> names(annoRD) <- gsub(".*=", "", elementMetadata(annoRD)[, "group"])
> annoRD <- as(annoRD, "RangedData")
> peaksRD <- RangedData(space=bpeaksDF$space, IRanges(bpeaksDF$start, bpeaksDF$end))
> annotatedPeak <- annotatePeakInBatch(peaksRD, AnnotationData = annoRD)
> as.data.frame(annotatedPeak)[1:4,1:11]

  space start   end width        names peak strand   feature start_position
1  Chr1  8301  8401   101 01 AT1G01020   01      + AT1G01020           5928
2  Chr1 11151 11351   201 02 AT1G01030   02      + AT1G01030          11649
3  Chr1 13601 13751   151 03 AT1G01030   03      + AT1G01030          11649
4  Chr1 16601 17051   451 04 AT1G01030   04      + AT1G01030          11649
  end_position insideFeature
1         8737        inside
2        13714      upstream
3        13714    overlapEnd
4        13714    downstream

> bpeaksDF[1:4,]

  space start   end width        PP   sig_cov sig_cov.pos sig_cov.neg
1  Chr1  8301  8401   101 0.5537269 0.1907785   0.1271857  0.06359283
2  Chr1 11151 11351   201 0.9998927 0.6390921   0.5911602  0.04793191
3  Chr1 13601 13751   151 0.9998833 0.4040882   0.2339458  0.17014240
4  Chr1 16601 17051   451 0.9999917 0.4699665   0.1922590  0.27770749
      bgr_cov bgr_cov.pos bgr_cov.neg
1 0.039603881 0.039603881 0.000000000
2 0.009950229 0.009950229 0.000000000
3 0.092715046 0.079470040 0.013245007
4 0.008869162 0.004434581 0.004434581
```

Alternative Peak Annotation Approach

Alternative approach using `olRanges` function

```
> source("./data/Fct/rangeoverlapper.R")
> olRanges(query=gffgene, subject=as(as(bpeaks, "RangedData"), "GRanges"), output="df")[1:2,]

  space Qindex Sindex Qstart  Qend Sstart  Send OLstart OLend OLlength  OLpercQ
1  Chr1      2      1   5928  8737   8301  8401    8301  8401      101 3.594306
2  Chr1      3      3  11649 13714  13601 13751   13601 13714      114 5.517909
    OLpercS OLtype
1 100.00000 inside
2  75.49669 oldown

> as.data.frame(annotatedPeak)[c(2,5),1:11] # Corresponding result from ChIPpeakAnno

  space start   end width        names peak strand   feature start_position
2  Chr1 11151 11351   201 02 AT1G01030   02      + AT1G01030          11649
5  Chr1 20901 21101   201 05 AT1G01040   05      + AT1G01040          23146
  end_position insideFeature
2        13714      upstream
5        31227      upstream
```

Outline

Import Annotation Data from GFF

Annotation data from GFF

```
> peakranges <- GRanges(seqnames = Rle(bpeaksDF$space), ranges = IRanges(bpeaksDF$start, bpeaksDF$end),
+                strand = Rle(strand("*")), peakIDs=paste("peak", seq(along=bpeaksDF[,1]), sep="_"))
> countDF <- data.frame(row.names=elementMetadata(peakranges)[,"peakIDs"])
> peakranges <- split(peakranges, 1:length(peakranges)) # Coerce to GRangesList
> for(i in samplespath) {
+         aligns <- readBamGappedAlignments(i) # Substitute next two lines with this one.
+         counts <- countOverlaps(peakranges, aligns)
+         countDF <- cbind(countDF, counts)
+ }
> colnames(countDF) <- samples
> rownames(countDF) <- gsub(".*=", "", rownames(countDF))
> countDF[1:4,]
```

	SRR038845.fastq	SRR038846.fastq	SRR038848.fastq	SRR038850.fastq
peak_1	9	36	2	5
peak_2	54	69	1	9
peak_3	31	13	7	3
peak_4	75	101	2	13

```
> write.table(countDF, "./results/countDF", quote=FALSE, sep="\t", col.names = NA)
> countDF <- read.table("./results/countDF")
```

Simple RPKM Normalization

RPKM: here defined as reads per kilobase of sequence range per million mapped reads

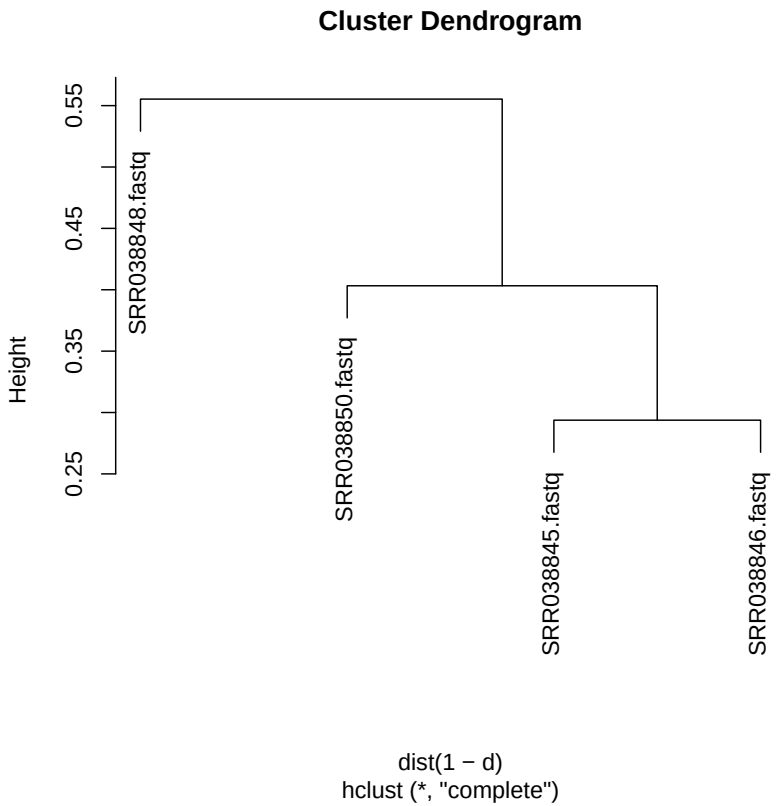
```
> returnRPKM <- function(counts, ranges) {
+         geneLengthsInKB <- sum(width(ranges))/1000 # Number of bases per sequence range in kbp
+         millionsMapped <- sum(counts)/1e+06 # Factor for converting to million of mapped reads.
+         rpm <- counts/millionsMapped # RPK: reads per kilobase of sequence range.
+         rpkm <- rpm/geneLengthsInKB # RPKM: reads per kilobase of sequence range per million mapped reads
+         return(rpkm)
+ }
> countDFrpkm <- apply(countDF, 2, function(x) returnRPKM(counts=x, ranges=peakranges))
> countDFrpkm[1:4,]
```

	SRR038845.fastq	SRR038846.fastq	SRR038848.fastq	SRR038850.fastq
peak_1	3377.000	11753.467	521.6813	1758.1132
peak_2	10181.404	11319.778	131.0692	1590.1740
peak_3	7780.271	2838.908	1221.2871	705.5739
peak_4	6302.237	7384.646	116.8289	1023.6819

QC Check

QC check by computing a sample correlating matrix and plotting it as a tree

```
> d <- cor(countDFrpkm, method="spearman")
> plot(hclust(dist(1-d))) # Sample tree
```



Identify DiffPeaks with Simple Fold Change Method

Compute mean values for replicates

```
> source("./data/Fct/colAg.R")
> countDFrpkm_mean <- colAg(myMA=countDFrpkm, group=c(1,1,2,2), myfct=mean)
> countDFrpkm_mean[1:4,]
```

```
       SRR038845.fastq_SRR038846.fastq SRR038848.fastq_SRR038850.fastq
peak_1                        7565.234                        1139.8973
peak_2                       10750.591                         860.6216
peak_3                        5309.589                         963.4305
peak_4                        6843.441                         570.2554
```

Log2 fold changes

```
> countDFrpkm_mean <- cbind(countDFrpkm_mean, log2ratio=log2(countDFrpkm_mean[,1]/countDFrpkm_mean[,2]))
> countDFrpkm_mean <- countDFrpkm_mean[is.finite(countDFrpkm_mean[,3]), ]
> degs2fold <- countDFrpkm_mean[countDFrpkm_mean[,3] >= 1 | countDFrpkm_mean[,3] <= -1,]
> degs2fold[1:4,]
```

```
       SRR038845.fastq_SRR038846.fastq SRR038848.fastq_SRR038850.fastq
peak_1                        7565.234                        1139.8973
peak_2                       10750.591                         860.6216
peak_3                        5309.589                         963.4305
peak_4                        6843.441                         570.2554
       log2ratio
peak_1  2.730481
peak_2  3.642893
peak_3  2.462348
peak_4  3.585042
```

```
> write.table(degs2fold, "./results/degs2fold", quote=FALSE, sep="\t", col.names = NA)
> degs2fold <- read.table("./results/degs2fold")
```

Identify DiffPeaks with DESeq Library

Raw count data are expected here!

```
> library(DESeq)
> countDF <- read.table("./results/countDF")
> conds <- targets$Factor
> cds <- newCountDataSet(countDF, conds) # Creates object of class CountDataSet derived from eSet class
> counts(cds)[1:4, ] # CountDataSet has similar accessor methods as eSet class.

       SRR038845.fastq SRR038846.fastq SRR038848.fastq SRR038850.fastq
peak_1               9              36               2               5
peak_2              54              69               1               9
peak_3              31              13               7               3
peak_4              75             101               2              13

> cds <- estimateSizeFactors(cds) # Estimates library size factors from count data. Alternatively, one can
> cds <- estimateDispersions(cds, fitType="local") # Estimates the variance within replicates
> res <- nbinomTest(cds, "bgr1", "sig1") # Calls DEGs with nbinomTest
> res <- na.omit(res)
> res2fold <- res[res$log2FoldChange >= 1 | res$log2FoldChange <= -1,]
> res2foldpadj <- res2fold[res2fold$padj <= 0.2, ] # Here padj set very high for demo purpose
> res2foldpadj[1:2,1:8]

      id baseMean baseMeanA baseMeanB foldChange log2FoldChange        pval
2 peak_2 25.35138  6.807272  43.89548   6.448322       2.688924 0.002709185
4 peak_4 36.52068 10.211262  62.83010   6.153020       2.621295 0.001821145
        padj
2 0.03142452
4 0.02959360
```

Identify DiffPeaks with edgeR Library

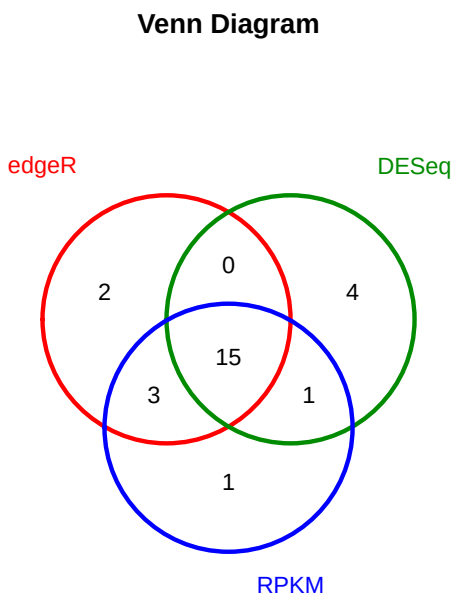
Raw count data are expected here!

```
> library(edgeR)
> countDF <- read.table("./results/countDF")
> y <- DGEList(counts=countDF, group=conds) # Constructs DGEList object
> y <- estimateCommonDisp(y) # Estimates common dispersion
> y <- estimateTagwiseDisp(y) # Estimates tagwise dispersion
> et <- exactTest(y, pair=c("bgr1", "sig1")) # Computes exact test for the negative binomial distribution.
> topTags(et, n=4)

Comparison of groups:  sig1-bgr1
           logFC   logCPM       PValue          FDR
peak_38 3.909219 10.39669 1.405486e-09 9.135660e-08
peak_49 3.950497 10.78386 5.627941e-08 1.829081e-06
peak_4  3.750096 10.69630 1.620723e-07 3.511566e-06
peak_35 3.308416 11.05086 1.034947e-05 1.681789e-04

> edge <- as.data.frame(topTags(et, n=50000))
> edge2fold <- edge[edge$logFC >= 1 | edge$logFC <= -1,]
> edge2foldpadj <- edge2fold[edge2fold$FDR <= 0.01, ]
```

Merge Results and Compute Overlaps Among Methods

Here overlaps for 20 best ranking peaks of each method!

```
> bothDF <- merge(res, countDFrpkm_mean, by.x=1, by.y=0, all=TRUE); bothDF <- na.omit(bothDF)
> cor(bothDF[,"log2FoldChange"], bothDF[,"log2ratio"], method="spearman")
[1] 0.9934441
> source("./data/Fct/overLapper.R")
> setlist <- list(edgeR=rownames(edge[order(edge$FDR),][1:20,]),
+                 DESeq=res[order(res$padj),][1:20,"id"],
+                 RPKM=rownames(degs2fold[order(-degs2fold$log2ratio),][1:20,]))
> OLlist <- overLapper(setlist=setlist, sep="_", type="vennsets")
> counts <- sapply(OLlist$Venn_List, length)
> vennPlot(counts=counts)
```



Outline

Inspect Results in IGV

View peak_3 in IGV

- Download and open IGV Link
- Select in menu in top left corner `A. thaliana (TAIR10)`
- Upload the following indexed/sorted Bam files with `File -> Load from URL...`

http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038845.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038846.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038848.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038850.fastq.bam

- To view peak_3, enter its coordinates `Chr1:16656-16956` in position menu on top.

Outline

Sequence Motifs Enriched in Peak Sequences I

Extract peak sequences and predict enriched motifs with BCRANK library

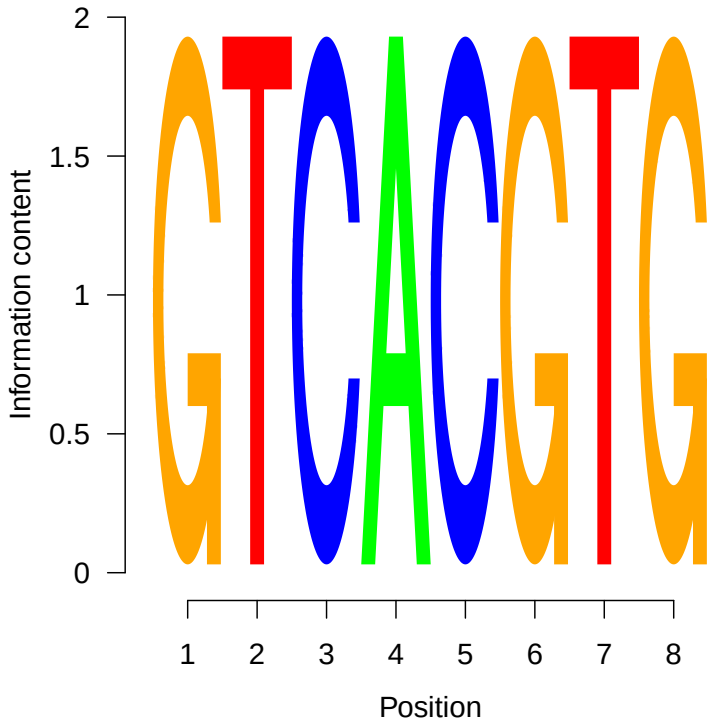
```
> library(Biostrings); library(seqLogo); library(BCRANK)
> pseq <- getSeq(FaFile("./data/tair10chr.fasta"), as(as(bpeaksDF, "RangedData"), "GRanges"))
> names(pseq) <- paste(bpeaksDF$space, bpeaksDF$start, sep="_")
> writeXStringSet(pseq[1:8], "./results/pseq.fasta") # Note: reduced to 8 sequences to run quickly.
> set.seed(0)
> BCRANKout <- bcrank("./results/pseq.fasta", restarts=25, use.P1=TRUE, use.P2=TRUE)
```

```
**** Running BCRANK on 8 regions, starting from HGRMHGHVSS ****
Iteration 1 - HGRMHGHVSS: 0
Scanning sequences............
Computing scores..............

Iteration 2 - AGRMHGHVSS: 7.673544
Scanning sequences............
Computing scores..............

Iteration 3 - AGAMHGHVSS: 10.23139
Scanning sequences............
Computing scores..............

Iteration 4 - AGAAHGHVSS: 12.78924
Scanning sequences............
Computing scores..............

Iteration 5 - AGAATGHVSS: 15.34709
Scanning sequences............
Computing scores..............

Iteration 6 - AGAATGTVSS: 17.90494
Scanning sequences............
Computing scores..............

Iteration 7 - AGAATGTASS: 20.46278
```

Sequence Motifs Enriched in Peak Sequences II

Plot `BCRANK` result

```
> topMotif <- toptable(BCRANKout, 1)
> weightMatrix <- pwm(topMotif, normalize = FALSE)
> weightMatrixNormalized <- pwm(topMotif, normalize = TRUE)
> seqLogo(weightMatrixNormalized)
```



Exercise 2: Motif Enrichment Analysis

Task 1 Extract from the BCRANK result stored in `weightMatrix` the motif occurrence patterns and generate with them a position weight matrix using the `PWM` function from Biostrings.

Task 2 Enumerate the motif matches in the peak sequences and the entire genome using Biostring's `countPWM` function.

Task 3 Determine which sequence type, peak or genome, shows more matches per 1kbp sequence for this motif.

Task 4 Homework: write a function for computing enrichment p-values for motif matches based on the hypergeometric distribution.

Session Information

```
> sessionInfo()

R version 2.15.2 (2012-10-26)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)
locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] BCRANK_1.20.0                       seqLogo_1.24.0
 [3] edgeR_3.0.4                         DESeq_1.10.1
 [5] locfit_1.5-8                        ChIPpeakAnno_2.6.0
 [7] limma_3.14.3                        org.Hs.eg.db_2.8.0
 [9] GO.db_2.8.0                         RSQLite_0.11.2
[11] DBI_0.2-5                           AnnotationDbi_1.20.3
[13] BSgenome.Ecoli.NCBI.20080805_1.3.17 multtest_2.14.0
[15] Biobase_2.18.0                      biomaRt_2.14.0
[17] VennDiagram_1.5.1                   chipseq_1.8.0
[19] BSgenome_1.26.1                     ShortRead_1.16.3
[21] latticeExtra_0.6-24                 RColorBrewer_1.0-5
[23] lattice_0.20-10                     BayesPeak_1.10.0
[25] Rsamtools_1.10.2                    Biostrings_2.26.2
[27] rtracklayer_1.18.1                  GenomicRanges_1.10.5
[29] IRanges_1.16.4                      BiocGenerics_0.4.0

loaded via a namespace (and not attached):
 [1] annotate_1.36.0    bitops_1.0-4.2     genefilter_1.40.0  geneplotter_1.36.0
 [5] hwriter_1.3        MASS_7.3-22        parallel_2.15.2    RCurl_1.95-3
 [9] splines_2.15.2     stats4_2.15.2      survival_2.36-14   tools_2.15.2
[13] XML_3.95-0.1       xtable_1.7-0       zlibbioc_1.4.0
```