Starr: Simple Tiling ARRay analysis for Affymetrix ChIP-chip data

Benedikt Zacher, Achim Tresch

October 29, 2019

1 Introduction

Starr is an extension of the *Ringo* [6] package for the analysis of ChIP-chip projects. Whereas the latter is specialized to the processing of Nimblegen and Agilent arrays, the former provides all corresponding features for Affymetrix arrays. Data can be read in from Affymetrix CEL-files, or from text files in gff format. Standard quality assessment and data normalization tools are available. *Starr* uses the Bioconductor *ExpressionSet* class for data storage. The *probeAnno* class from *Ringo* serves as a mapping of the ChIP signals onto the genomic position. Consequently, all functions from *Ringo* that operate on either *ExpressinoSet* or *probeAnno* can be used without modification. These include smoothing operations, peak-finding, and quality control plots. *Starr* adds new options for high-level analysis of ChIP-chip data. We demonstrate *Starr's* facilities at the example of an experiment that compares DNA binding under two different conditions. Three chips have been produced, two contain the actual immunoprecipitated DNA, and the other one is a control experiment.

```
> library(Starr)
```

2 Reading the data

To read Affymetrix tiling array data, two different file types are required. The bpmap file contains the mapping of the physical position on the array to the genomic position of the probe sequences. The CEL file delivers the measured intensities from the scanner. The data included in this package contains the first 80000 bp from chromosome 1 of a real ChIP-chip experiment in yeast. Artificial bpmap and CEL files were constructed for demonstration purposes. Two ChIP-chip experiments were performed with TAP-tagged RNA Polymerase II subunit Rpb3. For the control experiment, the ChIP-chip protocol has exactly been reproduced with wild type cells (i.e. with no TAP-tag to Rpb3). The *readBpmap()* function from the *affxparser* package reads the bpmap file.

```
> dataPath <- system.file("extdata", package="Starr")
> bpmapChr1 <- readBpmap(file.path(dataPath, "Scerevisiae_tlg_chr1.bpmap"))
```

The function *readCelFile()* reads one or more CEL files and stores them in an *ExpressionSet*. Additionally to the path to the CEL files, experiment names and the type of experiment must be specified. An optional experimentData object can be included. This is a "MIAME" object, which includes information about the experiment (e.g. the investigator or lab where the experiment was done, an overall title, etc.).

```
> cels <- c(file.path(dataPath,"Rpb3_IP_chr1.cel"), file.path(dataPath,"wt_IP_chr1.cel"),
+         file.path(dataPath,"Rpb3_IP2_chr1.cel"))
```

```
> names <- c("rpb3_1", "wt_1","rpb3_2")
> type <- c("IP", "CONTROL", "IP")
> rpb3Chr1 <- readCelFile(bpmapChr1, cels, names, type, featureData=T, log.it=T)
```

Now we give a very short introduction to the *ExpressionSet* class. For a more detailed view, please refer to "An Introduction to Bioconductor's ExpressionSet Class" [4]. A summary of the *ExpressionSet* can be shown with:

```
> rpb3Chr1

ExpressionSet (storageMode: lockedEnvironment)
assayData: 20000 features, 3 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: rpb3_1 wt_1 rpb3_2
  varLabels: type CEL
  varMetadata: labelDescription
featureData
  featureNames: 1 2 ... 20000 (20000 total)
  fvarLabels: chr seq pos
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

The *ExpressionSet* in this case consists of three different objects. Optinally, a *MIAME* object can be added as just described. In the following, a short description of the components of `rpb3Chr1` is given:

1. The *assayData* is a matrix with the measured signals.

   ```
   > head(exprs(rpb3Chr1))

       rpb3_1     wt_1   rpb3_2
   1 14.51089 14.42134 15.39158
   2 12.67154 14.12533 12.88817
   3 14.49791 14.09515 15.21496
   4 11.89860 13.72238 12.61816
   5 13.75071 13.73682 15.05078
   6 11.35590 13.00545 12.09935
   ```

2. Phenotypic data summarizes information about the samples (e.g., information about type of experiment, such as tagged IP, raw IP, input DNA, ...). The rownames of the phenotypic data are equal to the colnames of the assayData. The information about the type of experiment is needed for the normalization.

   ```
   > pData(rpb3Chr1)

             type
   rpb3_1      IP
   wt_1   CONTROL
   rpb3_2      IP
                                                                        CEL
   rpb3_1  /tmp/Rtmp5C0Scj/Rinst511375f4042e/Starr/extdata/Rpb3_IP_chr1.cel
   wt_1      /tmp/Rtmp5C0Scj/Rinst511375f4042e/Starr/extdata/wt_IP_chr1.cel
   rpb3_2 /tmp/Rtmp5C0Scj/Rinst511375f4042e/Starr/extdata/Rpb3_IP2_chr1.cel
   ```

3. The *featureData* in this case contains information from the bpmap file. The featureNames correspond to the rownames of the *assayData* of the *ExpressionSet*. With the *featureData*, each ChIP-signal from the expression matrix can be mapped to the chromosome and its position on it, as well as its genomic sequence. This information can be used for sequence specific normalization methods.

```
> featureData(rpb3Chr1)

An object of class 'AnnotatedDataFrame'
  featureNames: 1 2 ... 20000 (20000 total)
  varLabels: chr seq pos
  varMetadata: labelDescription

> head(featureData(rpb3Chr1)$chr)

[1] chr1 chr1 chr1 chr1 chr1 chr1
Levels: chr1

> head(featureData(rpb3Chr1)$seq)

[1] "GTGTGGGTGTGTGGGTGTGGTGTGG" "ACCACACCCACACACCCACACACCA"
[3] "GGTGTGGTGTGTGGGTGTGTGGGTG" "CACACACCCACACACCACACCACAC"
[5] "TGGTGTGTGGTGTGGTGTGTGGGTG" "CACACACCACACCACACACCACACC"

> head(featureData(rpb3Chr1)$pos)

[1]  1  5  9 13 17 21
```

3 Diagnostic plots

Since the probes are placed on the array in a randomized way, localized signal distortions are most likely due to technical artefacts. A reconstruction of the array image helps to identify these defects. The *plotImage()* function constructs a reconstruction of the artificial array, used in this example (see figure 1).

```
> plotImage(file.path(dataPath,"Rpb3_IP_chr1.cel"))
```

Besides that, *Starr* provides different diagnostic plots for the visual inspection of the data. These plots should help to find an appropriate normalization method. The densityplots and the boxplots show the distribution of the measured intensities (see figure 2).

```
> par(mfcol=c(1,2))
> plotDensity(rpb3Chr1, oneDevice=T, main="")
> plotBoxes(rpb3Chr1)
```

To compare the different experiments, the *plotScatter()* function can be applied. This produces a matrix of pairwise scatterplots in the upper panel and pearson correlation in the lower panel. The density of the data points can be visualized with a color gradient (see figure 3).

```
> plotScatter(rpb3Chr1, density=T, cex=0.5)
```

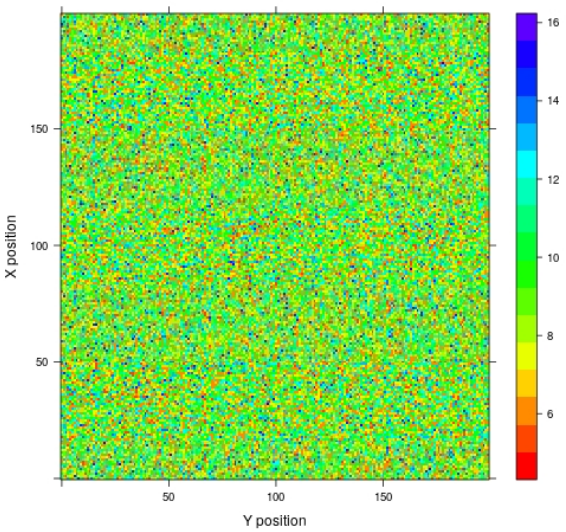

Figure 1: Spatial distribution of raw reporter intensities of the artificial array

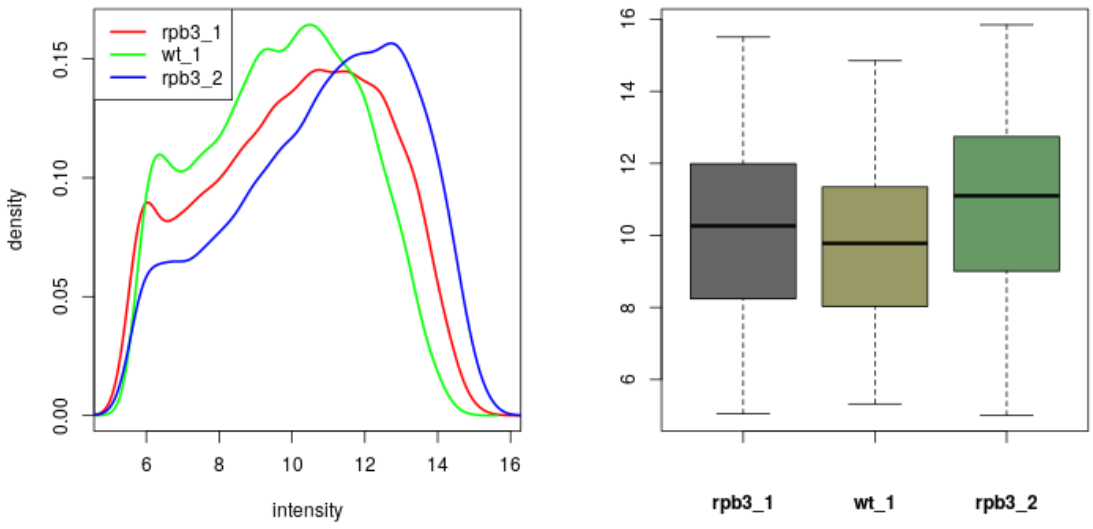

Figure 2: Density- and boxplots of the logged intensities

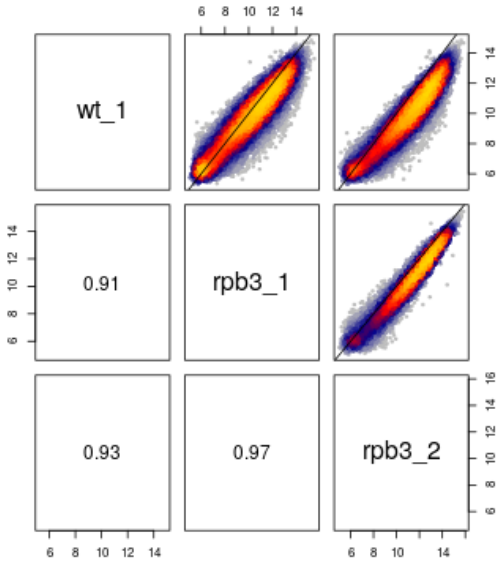

Figure 3: A scatterplot matrix, showing the correlation in the lower panel. In the scatterplots, the density of the points is illustrated with a color gradient.

MA-plots are a classic and important quality control plot to spot and correct saturation-dependent effects in the log enrichment. For each probe, the log enrichment M is plotted versus the average log intensities of signal and reference (A-value). Ideally, the measured enrichment should be independent of the mean intensity A of signal and reference. But if e.g. the signal and the reference measurements have different saturation characteristics, then e.g. M will show a dependence on A. *plotMA()* constructs MA plots of all pairs of Immunoprecipitation and control experiments (see figure 4).

```
> ips <- rpb3Chr1$type == "IP"
> controls <- rpb3Chr1$type == "CONTROL"
> plotMA(rpb3Chr1, ip=ips, control=controls)
```

The last diagnostic plot shown here is about the sequence depedent bias of the probe intensities (see figure 5). The raw logged intensity depends on the GC-content of the probe sequence. But there is also a remarkable dependency on base position within the sequence.

```
> par(mfcol=c(1,2))
> plotGCbias(exprs(rpb3Chr1)[,1], featureData(rpb3Chr1)$seq, main="")
> plotPosBias(exprs(rpb3Chr1)[,1], featureData(rpb3Chr1)$seq)
```

4 Normalization of the data

After quality assessment, we perform normalization of the raw data. Here we use the cyclic loess normalization.

```
> rpb3_loess <- normalize.Probes(rpb3Chr1, method="loess")
```

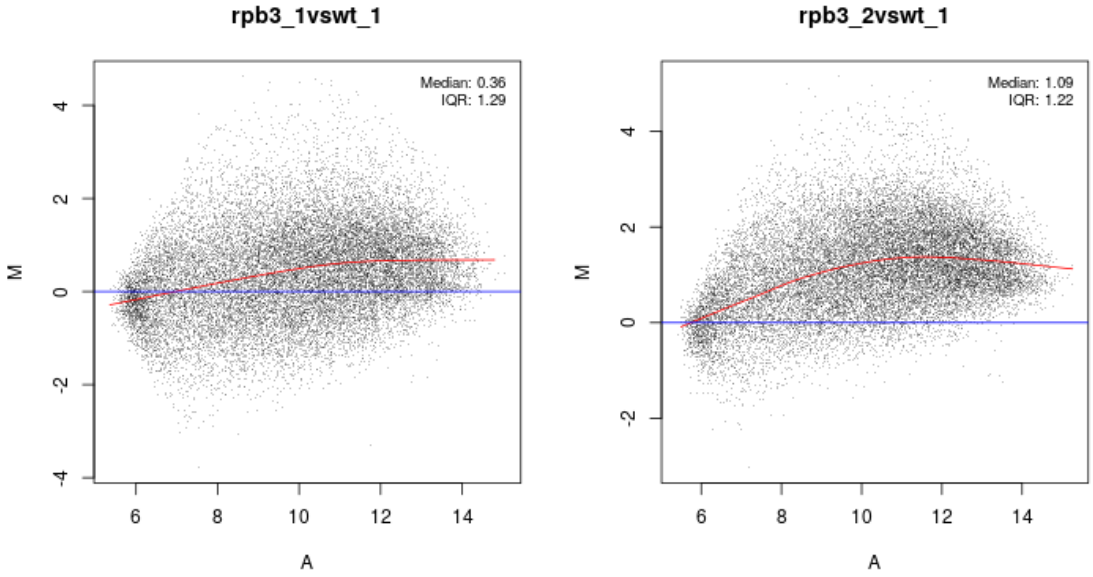

Figure 4: Pairwise MA-plots of all pairs of Immunoprecipitation and control experiments. Both plots show a dependency between A and M value.

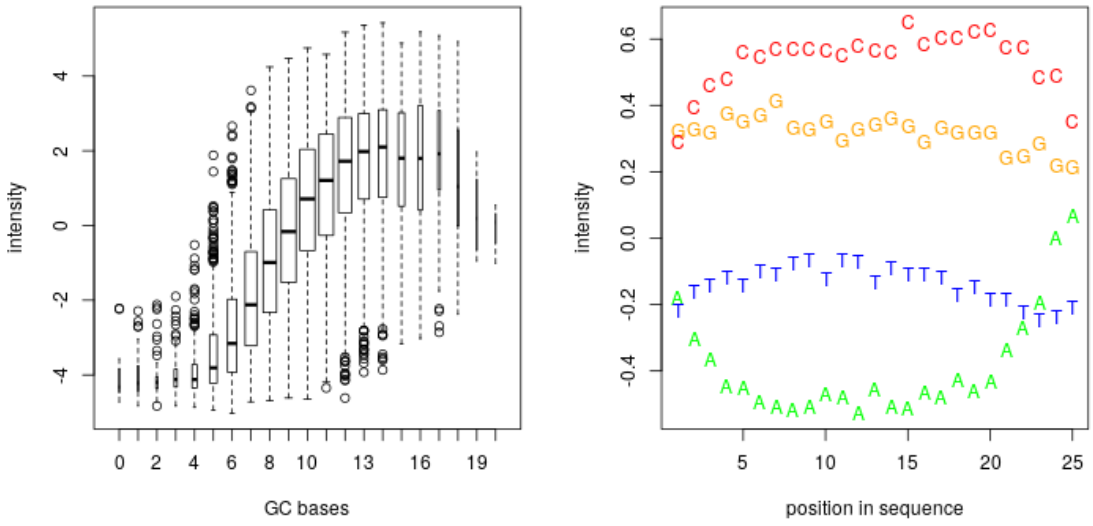

Figure 5: Sequence-specific hybridization bias (raw data). The raw logged intensity depends on the GC-content of the probe sequence. But there is also a remarkable dependency on base position within the sequence.

Besides this normalization method, there are e.g. median rank percentile [2], scale, quantile, vsn, MAT and some others available. After normalization, we perform again diagnositc plots to assert that the normalization was appropriate for the correction of the systematic measurement errors. The MA-plot of the normalized data does not show any dependence of the M and A values (see figure 6).

```
> plotMA(rpb3_loess, ip=ips, control=controls)
```

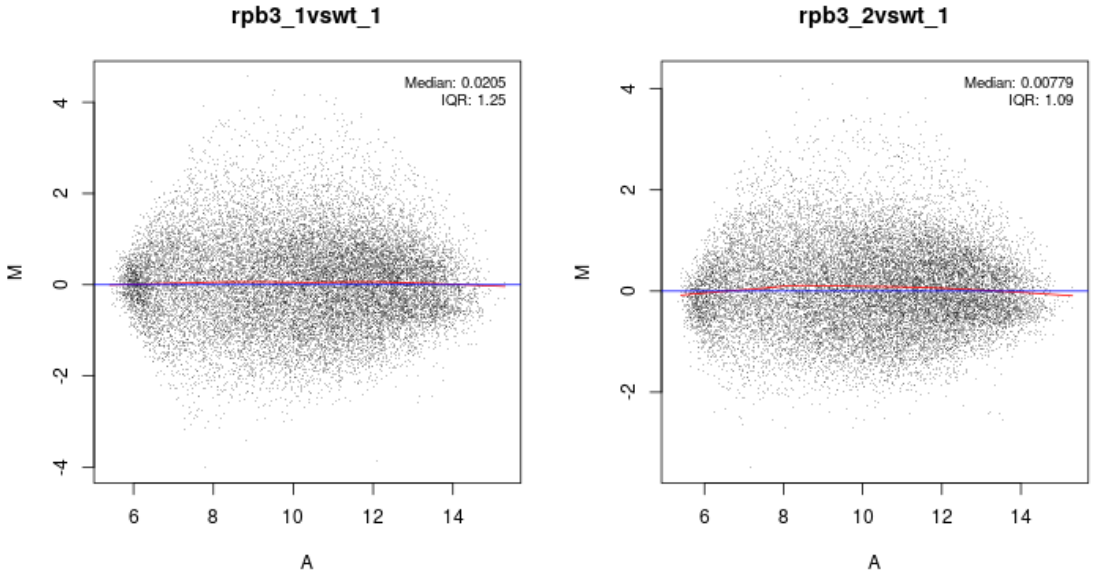

Figure 6: MA-plot of normalized data. There is no dependency between A- and M-value observed any more.

Now we calculate the ratio of the probe intensities. Median values over replicates are taken.

```
> description <- c("Rpb3vsWT")
> rpb3_loess_ratio <- getRatio(rpb3_loess, ips, controls, description, fkt=median, featureData=F)
```

It is very important that the control or reference experiment is able to correct the sequence-dependent bias on probe intensity, which is shown in figure 5. In this case the normalization and reference experiment was adequate to correct all systematic biases in the data (see figure 7).

```
> par(mfcol=c(1,2))
> plotGCbias(exprs(rpb3_loess_ratio)[,1], featureData(rpb3_loess)$seq, main="")
> plotPosBias(exprs(rpb3_loess_ratio)[,1], featureData(rpb3_loess)$seq, ylim=c(-0.5,0.5))
```

5 Data analysis

Besides the typical ChIP-chip analysis of the data, like visualization (see *Ringo*) or peak finding, *Starr* provides additional useful functions to analyze ChIP-signals along specific genomic regions. For this purpose, we need a mapping of the probe intensities in our *ExpressionSet* to the genomic positions. To achieve that, we construct a *probeAnno* object (as provided by *Ringo*). The object

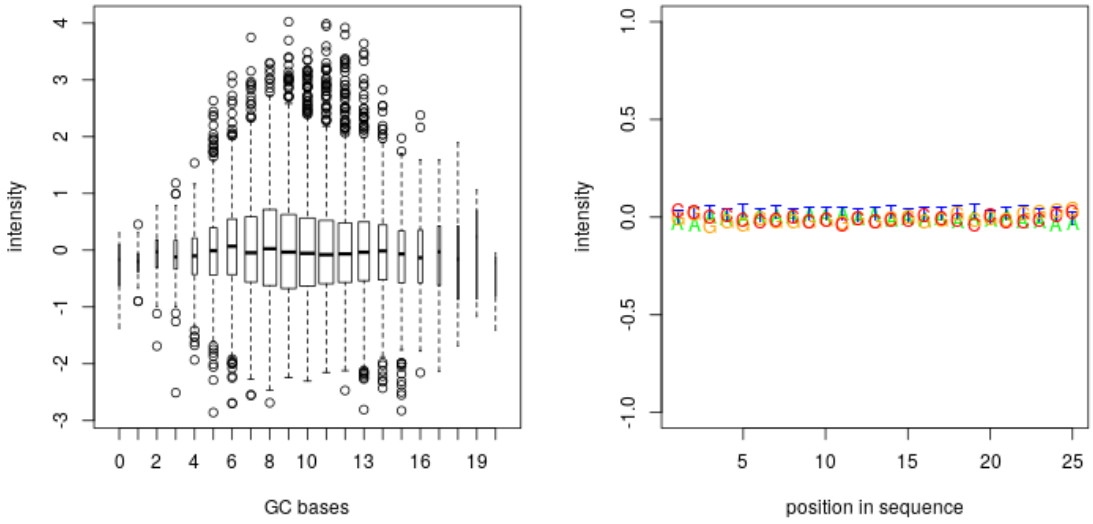

Figure 7: Dependency of probe intensity on sequences (normalized ratio). The systematic bias could be corrected.

consists of four vectors of equal length and ordering for each chromosome. The vectors specify probe start and end, as well as the index of the probe intensity in the *ExpressionSet*. The unique vector encodes how many matches the corresponding probe has on the given array. An entry of '0' indicates that the probe matching at this position has only this one match. See *Ringo* for a detailed description of the *probeAnno* class.
If the array was designed on an outdated assembly of the genome, a re-mapping of reporters to the genome can be necessary. Further on, the unique vector does possibly not identify all probes, that match the genome uniquely at the sepcified position. A re-mapping can be used to identify all uniquely matching reporters.

```
> probeAnnoChr1 <- bpmapToProbeAnno(bpmapChr1)
```

5.1 Remapping probes to a current genome build

Starr provides an easy-to-use method for remapping probe sequences and building new bpmap annotation. It implements the Aho-Corasick [1] string matching algorithm, which is designed for searching a given set of sequences in a text. The genomic sequences must be provided as fasta files. Each file is supposed to contain one chromosome. The sequences to be searched can be passed to the function either as a character vector or as a bpmap list (returned by RmethodreadBpmap).
An example below, shows how to match the sequences of the given bpmap file from above to chromosome 1 of *S. cerevisiae*. Sequences in this bpmap file are taken from both strands in $5' \rightarrow 3'$ direction, that means we have to search the +1 and -1 strand. The sequence of chromosome 1 is stored in a fasta file chrI.fa in the dataPath folder.

```
> newbpmap <- remap(bpmapChr1, path=dataPath, reverse_complementary=TRUE, return_bpmap=TRUE)

Number of nodes: 365098
Searching: chrI
89.5 % of the probes could be mapped uniquely.
```

In this case, 89.5 % of the probe sequences could be mapped to a unique position on chromosome 1. The method returns a list in the output format of the *affxparser* function *readBpmap*.

array	time	#sequences	genome size (bp)
S. cerevisiae Tiling 1.0R	34 s	2 697 594	12 495 682
Drosophila Tiling 2.0R	1 min 16s	2 907 359	122 653 977
Human Promoter 1.0R	14 min 22 s	4 315 643	$3.3 * 10^9$

Table 1: Time for remapping of reporter sequences from Affymetrix tiling arrays to a current genome build. Results were calculated on an Intel Core Duo E8600 3.33 GHz machine.

```
> str(newbpmap)

List of 1
 $ chrI:List of 8
  ..$ seqInfo :List of 7
  .. ..$ name        : chr "chrI"
  .. ..$ groupname   : chr ""
  .. ..$ fullname    : chr "chrI"
  .. ..$ version     : chr ""
  .. ..$ mapping     : chr "onlypm"
  .. ..$ number      : int 1
  .. ..$ numberOfHits: int 17900
  ..$ pmx     : int [1:17900] 129 129 27 27 170 170 131 131 130 130 ...
  ..$ pmy     : int [1:17900] 0 100 86 186 42 142 66 166 75 175 ...
  ..$ mmx     : NULL
  ..$ mmy     : NULL
  ..$ probeseq: chr [1:17900] "GTGTGGGTGTGTGGGTGTGGTGTGG" "ACCACACCCACACACCCACACACCA" "GGTGTGGTGT
  ..$ strand  : int [1:17900] 0 1 0 1 0 1 0 1 0 1 ...
  ..$ startpos: int [1:17900] 1 5 9 13 17 21 25 29 33 37 ...
```

One can use this list either to write a new binary bpmap file, or to create a new *probeAnno* object. Note, that this bpmap file differs from the original file. Consequently, one has to read in the data using this file, otherwise the *probeAnno* object will not be compatible with the *ExpressionSet*.

```
> writeTpmap("newbpmap.tpmap", newbpmap)
> tpmap2bpmap("newbpmap.tpmap", "newbpmap.bpmap")
> pA <- bpmapToProbeAnno(newbpmap)
```

The function works efficinetly for all sizes of genomes. Table 1 shows a comparison of computation time for different Affymetrix tiling arrays. If the memory on your machine is not sufficient for the amount of sequences that should be mapped, the parameter nseq can be set to search the sequences in more than one iteration.

5.2 Analyse the correlation of ChIP signals to other data

In the following section we want to demonstrate, how the binding profiles of the protein of interest can be analyzed over annotated genomic features. First we read in a gff file, which contains annotations for transcription start (TSS) and termination sites (TTS) of some genes on chromosome 1 [3]. The *filterGenes()* function filters the annotated features with respect to length, overlaps or distance to other features. In this case the genes are supposed to have a minimal length of 1000 base pairs.

```
> transcriptAnno <- read.gffAnno(file.path(dataPath, "transcriptAnno.gff"), feature="transcript")
> filteredIDs <- filterGenes(transcriptAnno, distance_us = 0, distance_ds = 0, minLength = 1000)
```

The *correlationPlot()* can then be used to visualize e.g. the correlation between the mean binding intensity of specific regions around these transcripts and gene expression. First we need

to define the regions around the annotated features, that we want to analyze. This is realised with a data frame.

```
> pos <- c("start", "start", "start", "region", "region","region","region", "stop","stop","stop")
> upstream <- c(500, 0, 250, 0, 0, 500, 500, 500, 0, 250)
> downstream <- c(0, 500, 250, 0, 500, 0, 500, 0, 500, 250)
> info <- data.frame(pos=pos, upstream=upstream, downstream=downstream, stringsAsFactors=F)
```

Every row of this data frame represents one region, flanking the annotated features. The first row e.g. indicates, that we want to calculate the mean ChIP signal 500 bp upstream and 0 bp downstream around the start of the feature. The term "region" means in this context the area between start and stop of the feature. Once we have defined these regions, we use the *getMeans()* function to calculate the mean intensity over these regions for each transcript in our gff annotation. This function returns a list. Each entry of the list represents one of the regions defined above and contains a vector with all mean signals of the annotated featues.

```
> means_rpb3 <- getMeans(rpb3_loess_ratio, probeAnnoChr1, transcriptAnno[which(transcriptAnno$nam
```

Now, that we have the mean ChIP signals, we could define another vector, which contains e.g. gene expression values and visualize the correlation of the specific regions to the expression value. For this purpose the function *correlate()* from this package can be used to easily calculate the correlation between the different areas and the corresponding expression value. In this example, we just plot the mean signal over the different areas. The last thing we need to define for the visualization is the order of the boxes in the lower panel of the plot (see figure 8). In this lower panel, the different regions along the transcript, defined in the data frame `info` are shown. The numbering of the levels starts at the bottom with level 1. Now we add this information to the data frame and call *correlationPlot()*.

```
> info$cor <- sapply(means_rpb3, mean, na.rm=T)
> level <- c(1, 1, 2, 3, 4, 5, 6, 1, 1, 2)
> info$level <- level
> correlationPlot(info, labels=c("TSS", "TTS"))
```

5.3 Visualization of a set of "profiles"

Starr provides functions for the visualization of a set of "profiles" (e.g. time series, signal levels along genomic positions). Suppose that we are interested in the ChIP profile of a protein along the transcription start site. One way of looking over the inensity profiles is to take the mean intensity at each available position along this region. This illustration gives a first view of the main tendency in the profiles. But some profiles with e.g. extremely high values can easily lead to a distorted mean profile. To get a more detailed view on a group of profiles and their divergence, we developed the *profileplot*.
In this function, the profiles are given as the rows of a samples times $\times$ positions matrix that contains the respective signal of a sample at given position. Instead of plotting a line for each profile (e.g. column of the row), the q-quantiles for each position (e.g. column of the matrix) are calculated, where q runs through a set of representative quantiles. Then for each q, the profile line of the q-quantiles is plotted. Color coding of the quantile profiles aids the interpretation of the plot: There is a color gradient from the median profile to the 0 (=min) resp. 1 (=max) quantile. The following example shows how this function is used. First we construct an example data matrix.

```
> sampls = 100
> probes = 63
```

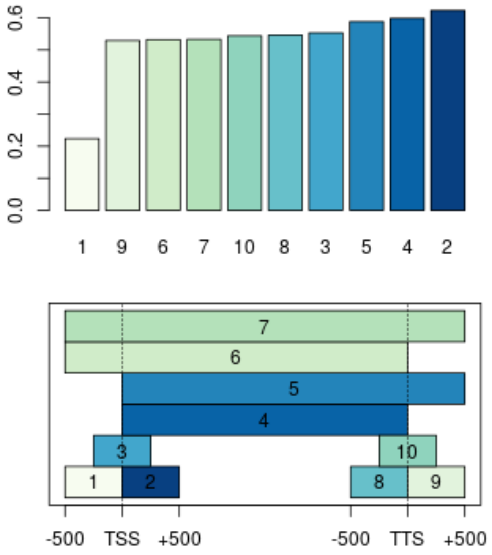

Figure 8: *correlationPlot* of the mean intensities over areas around transcription start site (TSS) and the transcription termination site (TTS) of annotated transcripts from chromosome 1 [3]. The lower panel shows the the analyzed regions. The upper panel shows the mean intensity over the individual regions.

```
> at = (-31:31)*14
> clus = matrix(rnorm(probes*samples,sd=1),ncol=probes)
> clus= rbind( t(t(clus)+sin(1:probes/10))+1:nrow(clus)/samples , t(t(clus)+sin(pi/2+1:probes/10))
```

Next, we apply kmeans clustering to identify two different clusters and construct a "character" vector, that indicates to which cluster an individual profile belongs.

```
> labs = paste("cluster",kmeans(clus,2)$cluster)
```

Then we apply the *profileplot* function. In this case, the quantiles from the 5%- to the 95%-quantile are shown with the color gradient (see figure 9). The median is shown as black line. The 25%- and the 75%-quantile are shown as grey lines. The grey lines in the background show the original profiles of the different clusters.

```
> par(mfrow=c(1,2))
> profileplot(clus,label=labs,main="Clustered data",colpal=c("heat","blue"),add.quartiles=T,fromt
```

5.4 Visualize profiles of ChIP signals along genomic features

With the just described methods, one can easily visualize ChIP profiles over annotated features of groups of genes (e.g. different groups of genes identified by a clustering method). To exemplify the usage of this visualization method, we build a gff annotation for the transcription start sites from our transcript annotation. For that purpose, we use the transcript annotation and set the end position of each transcript to its start site.

```
> tssAnno <- transcriptAnno
> watson <- which(tssAnno$strand == 1)
```

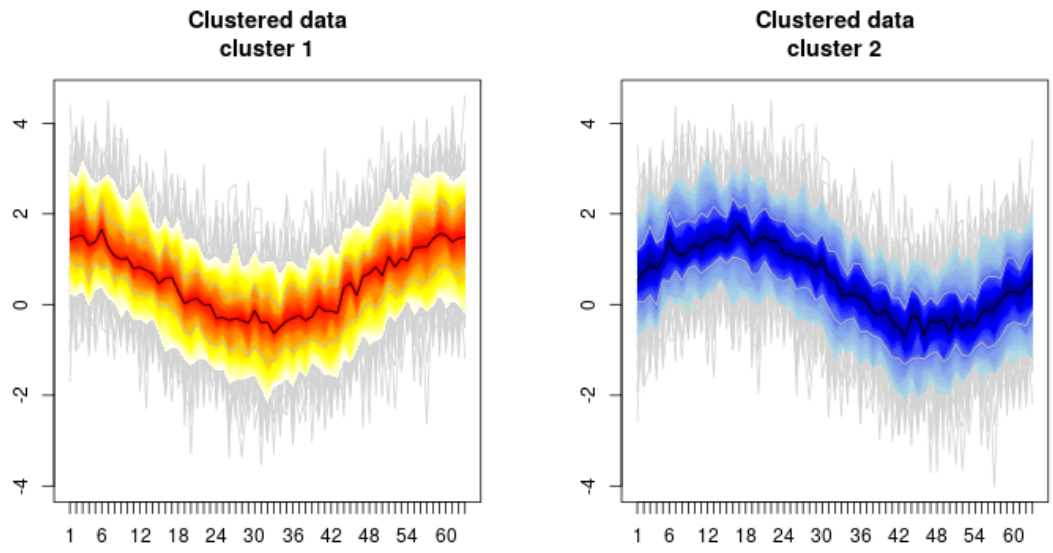

Figure 9: *profileplot* of two clusters identified by kmeans clustering. The quantiles from the 5%- to the 95%-quantile are shown with the color gradient. The median is shown as black line. The 25%- and the 75%-quantile are shown as grey lines. The grey lines in the background show the original profiles of the different clusters.

```
> tssAnno[watson,]$end <- tssAnno[watson,]$start
> crick <- which(tssAnno$strand == -1)
> tssAnno[crick,]$start <- tssAnno[crick,]$end
```

Then we use the *getProfiles()* function to obtain the profiles over 500 bp upstream and downstream around the transcription start site. The function constructs a list with the profiles and stores information about the border (like TSS, TTS, start, stop codon, etc.), as well as the length of the flanking upstream and downstream areas (500 bp here).

```
> profile <- getProfiles(rpb3_loess_ratio, probeAnnoChr1, tssAnno, 500, 500, feature="TSS", borde
```

Further on, we use the *plotProfiles()* function generate a plot of the mean ChIP profiles and a *profileplot* (as described in the previous section) of the annotated features (see figure 10).

```
> clust <- rep(1, dim(tssAnno)[1])
> names(clust) <- tssAnno$name
> plotProfiles(profile, cluster=clust)
```

5.5 Peak-finding with CMARRT

Starr implements the CMARRT [5] agorithm for identification of bound regions. CMARRT extends the standard moving average approach commonly used in the analysis of ChIP-chip data by incorporating the correlation structure in identifying bound regions for data from tiling arrays. Probes are declared as bound using adjusted p-values for multiple comparisons under the Gaussian approximation. The main function is *cmarrt.ma* which computes the p-values for each probe by taking into account the correlation structure. CMARRT is developed using the Gaussian approximation approach and thus it is important to check if this assumption is violated.

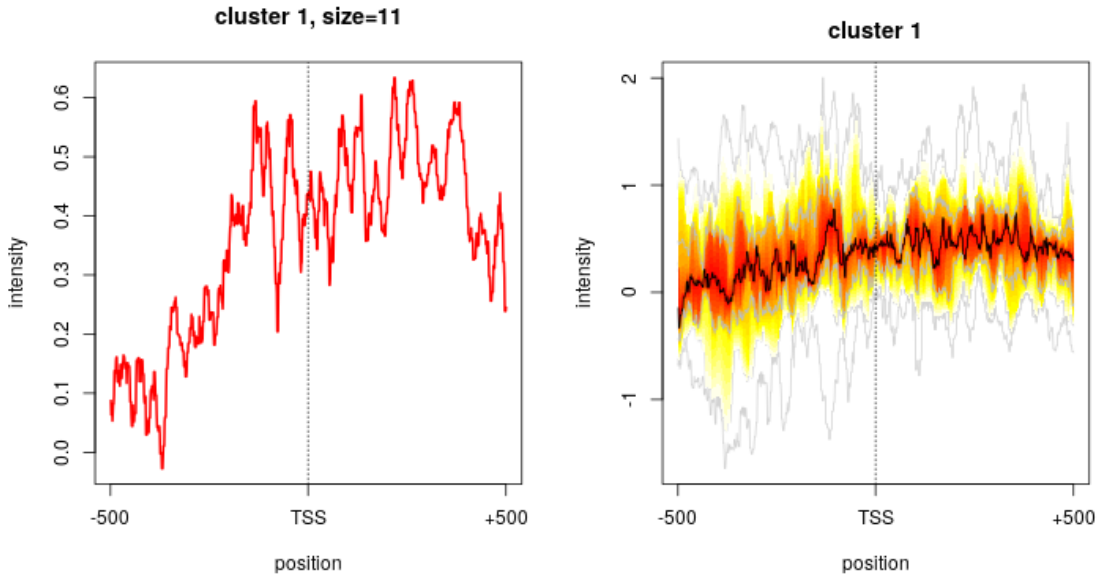

Figure 10: Visualization of the ChIP profiles along the transcription start site (TSS). On left left side the mean profile of the ChIP signals are shown. I.e., the mean signal at each available position is plotted. The *profileplot* on the rigtht side gives a more detailed view of the intensity profiles.

```
> peaks <- cmarrt.ma(rpb3_loess_ratio, probeAnnoChr1, chr=NULL, M=NULL, frag.length=300)
```

The function *plotcmarrt* produces the diagnostic plots (histogram of p-values and normal QQ plots) for comparing the distribution of standardized MA statistics under correlation and independence. If the distribution of the standardized moving average statistics S_i^* is correctly specified, the quantiles of S_i^* for unbound probes fall along a 45 degree reference line against the quantiles from the standard Gaussian distribution. In addition, the p-values obtained should be a mixture of uniform distribution between 0 and 1 and a non-uniform distribution concentrated near 0. Figure shows the summary statistics.

```
> plotcmarrt(peaks)
```

The list of bound regions is obtained using the function cmarrt.peak for a given error rate control which adjusts the p-values for multiple comparisons.

```
> peaklist <- cmarrt.peak(peaks, alpha = 0.05, method = "BH", minrun = 4)

> str(peaklist)

List of 2
 $ cmarrt.bound:List of 6
  ..$ Chr    : chr [1:45] "chr1" "chr1" "chr1" "chr1" ...
  ..$ Start  : num [1:45] 30261 32441 34077 34169 34729 ...
  ..$ Stop   : num [1:45] 30613 32785 34141 34209 35101 ...
  ..$ n.probe: int [1:45] 83 81 11 5 88 4 58 43 11 118 ...
  ..$ min.pv : num [1:45] 0.002175 0.002064 0.008177 0.010143 0.000301 ...
  ..$ ave.pv : num [1:45] 0.00731 0.0038 0.00935 0.01061 0.00313 ...
 $ indep.bound :List of 6
  ..$ Chr    : chr [1:39] "chr1" "chr1" "chr1" "chr1" ...
  ..$ Start  : num [1:39] 11201 11297 14489 14545 14597 ...
```

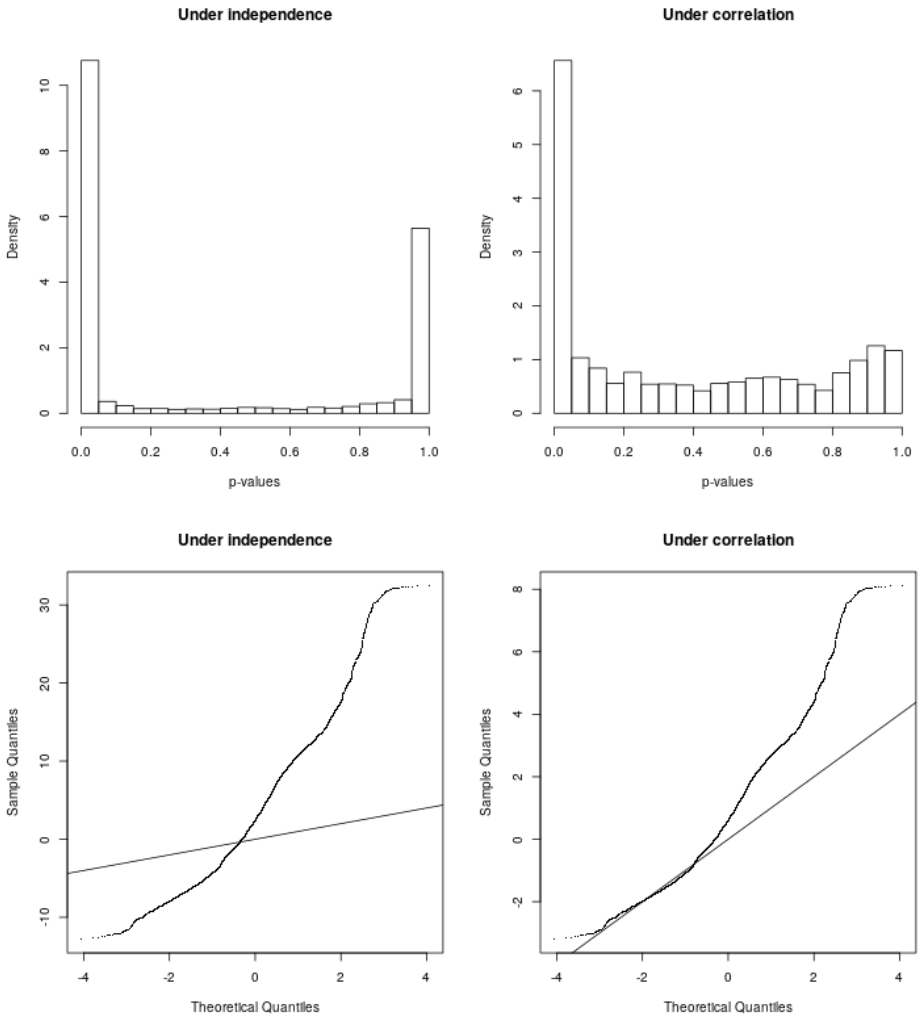

Figure 11: Normal quantile-quantile plots (qqplot) and histograms of p-values. The right panels show the qqplot of S_i and distribution of p-values under correlation structure. The bottom left panel shows that if the correlation structure is ignored, the distribution of S_i^* for unbound probes deviates from the standard Gaussian distribution. The top left panel shows that if the correlation structure is ignored, the distribution of p-values for unbound probes deviates from the uniform distribution for larger p-values.

```
..$ Stop   : num [1:39] 11244 11577 14545 14601 14661 ...
..$ n.probe: int [1:39] 6 65 9 9 11 15 7 8 8 4 ...
..$ min.pv : num [1:39] 1.58e-02 4.32e-08 1.37e-02 1.36e-02 1.01e-02 ...
..$ ave.pv : num [1:39] 0.02099 0.00187 0.01975 0.01741 0.01397 ...
```

The list of bound regions obtained under independence (ignoring the correlation structure) is for comparison. It is not recommended to use this list for downstream analysis.

5.6 Peak-finding and visualization using *Ringo*

In this section we shortly present how functions from the package *Ringo* can be used for peak finding and visualization. For a detailed description of the following work-flow see the *Ringo* vignette. Like it is recommended in the *Ringo* vignette, we first need to smooth the ChIP-chip intensities and define a threshold `y0` for enriched regions.

```
> rpb3_ratio_smooth <- computeRunningMedians(rpb3_loess_ratio, probeAnno=probeAnnoChr1, allChr =
> sampleNames(rpb3_ratio_smooth) <- paste(sampleNames(rpb3_loess_ratio),"smoothed")
> y0 <- apply(exprs(rpb3_ratio_smooth), 2, upperBoundNull)
```

The cutoff for maximum amount of base pairs at which enriched probes are condensed into one ChIP enriched region is taken as the maximal transcript length. We use the *Ringo* function *findChersOnSmoothed()* to identify ChIP enriched regions.

```
> distCutOff <- max(transcriptAnno$end - transcriptAnno$start)
> chers <- findChersOnSmoothed(rpb3_ratio_smooth, probeAnno=probeAnnoChr1, thresholds=y0, allChr=
```

Then regions with a maximal distance of 500 bp upstream to a transcript are related to the corresponding annotated features in the `transcriptAnno`. Below, five ChIP enriched regions that could be associated to an annotated feature are shown. They are sorted by the highest smoothed probe level in the enriched region.

```
> chers <- relateChers(chers, transcriptAnno, upstream=500)
```

```
> chersD <- as.data.frame.cherList(chers)
> chersD <- chersD[which(chersD$feature != ""),]
> chersD[order(chersD$maxLevel, decreasing=TRUE)[1:5],]
```

```
                                    name  chr start   end cellType
19 yeast.Rpb3vsWT smoothed.chrchr1.cher19 chr1 61991 62759    yeast
35 yeast.Rpb3vsWT smoothed.chrchr1.cher35 chr1 75291 75631    yeast
27 yeast.Rpb3vsWT smoothed.chrchr1.cher27 chr1 66271 66691    yeast
21 yeast.Rpb3vsWT smoothed.chrchr1.cher21 chr1 63235 63551    yeast
29 yeast.Rpb3vsWT smoothed.chrchr1.cher29 chr1 67463 67635    yeast
           antibody features maxLevel    score
19 Rpb3vsWT smoothed  YAL041W 1.443745 78.67887
35 Rpb3vsWT smoothed  YAL036C 1.287651 25.78318
27 Rpb3vsWT smoothed  YAL040C 1.159928 20.01283
21 Rpb3vsWT smoothed  YAL041W 1.131974 17.16309
29 Rpb3vsWT smoothed  YAL040C 1.110737 10.81569
```

Now we can plot the ChIP-enriched region, which was related to the feature with the maximal signal within the enriched area. Figure 12 shows this region.

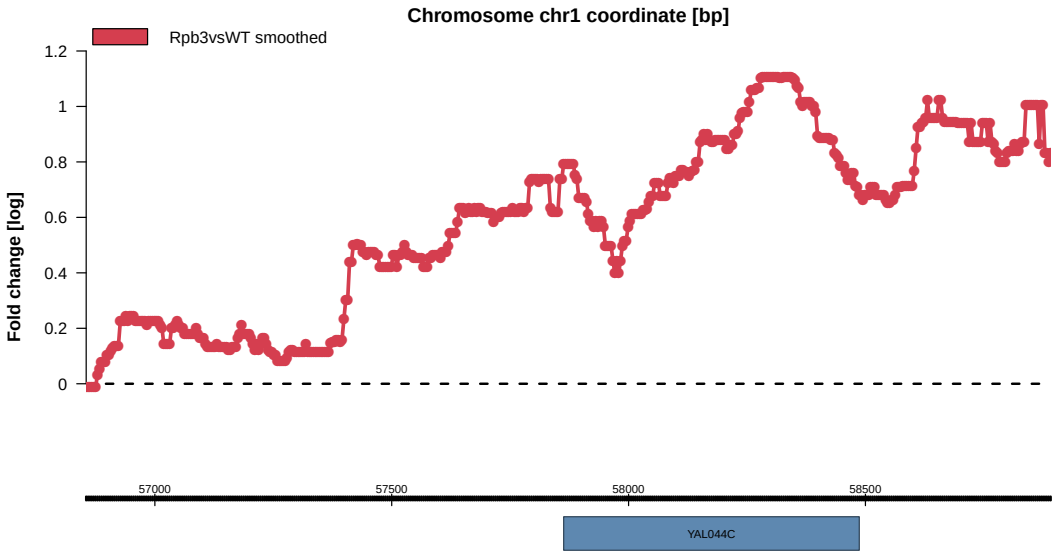

Figure 12: One of the identified Rpb3-antibody enriched regions on chromosome 1

6 Concluding Remarks

The package *Starr* facilitates the analysis of ChIP-chip data, in particular that of Affymetrix. It provides functions for data import, normalization and analysis. Besides that, high-level plots for quality assessment and the analysis of ChIP-profiles and ChIP-signals are available. Functions for smoothing operations, peak-finding, and quality control plots can be applied.

This vignette was generated using the following package versions:

- R version 3.6.1 (2019-07-05), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.10-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.10-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils
- Other packages: Biobase 2.46.0, BiocGenerics 0.32.0, Matrix 1.2-17, RColorBrewer 1.1-2, Ringo 1.50.0, Starr 1.42.0, affxparser 1.58.0, affy 1.64.0, lattice 0.20-38, limma 3.42.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.48.0, BiocManager 1.30.9, DBI 1.0.0, IRanges 2.20.0, MASS 7.3-51.4, R6 2.4.0, RCurl 1.95-4.12, RSQLite 2.1.2, Rcpp 1.0.2, S4Vectors 0.24.0, XML 3.98-1.20, affyio 1.56.0, annotate 1.64.0, assertthat 0.2.1, backports 1.1.5, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.2.0, colorspace 1.4-1, compiler 3.6.1, crayon 1.3.4, digest 0.6.22, dplyr 0.8.3, genefilter 1.68.0, ggplot2 3.2.1, glue 1.3.1, gtable 0.3.0, lazyeval 0.2.2, magrittr 1.5, memoise 1.1.0, munsell 0.5.0, pillar 1.4.2, pkgconfig 2.0.3, preprocessCore 1.48.0, pspline 1.0-18,

purrr 0.3.3, rlang 0.4.1, scales 1.0.0, splines 3.6.1, stats4 3.6.1, survival 2.44-1.1, tibble 2.1.3, tidyselect 0.2.5, tools 3.6.1, vctrs 0.2.0, vsn 3.54.0, xtable 1.8-4, zeallot 0.1.0, zlibbioc 1.32.0

Acknowledgments

I thank Michael Lidschreiber, Andreas Mayer and Kemal Akman for their help. Further on, I want to thank the reviewer for useful comments on the package.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(36):333–340, 1975.

[2] M. J. Buck and J. D. Lieb. Chip-chip: considerations for the design, analysis, and application of genome-wide chromatin immunoprecipitation experiments. *Genomics*, 83(3):349–360, 2004.

[3] L. David, W. Huber, M. Granovskaia, J. Toedling, C. J. Palm, L. Bofkin, T. Jones, R. W. Davis, and L. M. Steinmetz. A high-resolution map of transcription in the yeast genome. *Proc Natl Acad Sci U S A*, 103(14):5320–5325, 2006.

[4] S. Falcon, M. Morgan, and R. Gentleman. *An Introduction to Bioconductor's ExpressionSet Class*. http://wiki.biostat.berkeley.edu/~bullard/courses/Tmexico-08/resources/ExpressionSetIntroduction.pdf, February 2007.

[5] P. F. Kuan, H. Chun, and S. Keles. Cmarrt: a tool for the analysis of chip-chip data from tiling arrays by incorporating the correlation structure. *Pac Symp Biocomput*, pages 515–526, 2008.

[6] J. Toedling, O. Sklyar, T. Krueger, J. J. Fischer, S. Sperling, and W. Huber. Ringo - an R/Bioconductor package for analyzing ChIP-chip readouts. *BMC Bioinformatics*, 8:221, 2007.