An Introduction to *TransView*

Julius Müller[‡][*]

October 24, 2023

[‡]Institute of Molecular and Cell Biology
Singapore

Contents

---

[*]ju-mu@alumni.ethz.ch

1 Introduction

In modern biology, high-throughput sequencing has become an indispensable tool to understand transcriptional regulation. Numerous sequencing based methods have been developed allowing for an unbiased, genome wide analysis. Techniques like ChIP-Seq and RNA-Seq are routinely used and integrated to study the transcriptional outcome of transcription factor binding. With the large amount of data generated from such experiments however, the processing tools need to be designed with special focus on memory management and algorithm efficiency.

On the file storage side, the BAM file format [Li *et al.*, 2009] has become the *de facto* standard container for storing sequencing reads after the alignment. It can contain results and parameters of the alignment, allows for random access and furthermore keeps the file size at a minimum using file compression. In order to do genome wide calculations based on the read densities however, the individual chunks of the BAM file need to be decompressed, making repeated random access fairly slow.

The *TransView* package provides a mechanism to pre-fetch all reads from a SAM/BAM file into the memory as a genome wide read density map, making it instantly accessible to the user. The SAM/BAM parser is based on the Samtools C API , and the slicing functions are designed for fast random slicing and memory efficient storage. The package provides superior performance to existing methods and the density generation process is highly configurable based on flags stored in SAM/BAM files. It is compatible to paired end data sets, strand specific protocols, variable read lengths and spliced reads. Plotting facilities and accessors for read metrics derived from parsing are provided.

The *TransView* package is available at bioconductor.org and can be installed with:

```
> if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
> BiocManager::install("TransView")
```

2 Read density map construction and accession

The code to reproduce the examples in the following is partially attached to the package. The ChIP-Seq reads mapped to mm9 and the results from MACS peak calling [Zhang *et al.*, 2008] are attached. The amount of peaks of the two ChIP-Seq experiments has been reduced to 21, 500Bp long peak regions. A matching GTF file from UCSC is attached as well to demonstrate the annotation facilities. Due to space limitations the RNA-Seq visualisation is based on the bam files from the *pasillaBamSubset* data set and is therefore required to be loaded from bioconductor. A matching dm3 GTF file is included in TransView.

```
> library("TransView")
> library("GenomicRanges")
> library("pasillaBamSubset")
> fn.chipseq.bam <- dir(system.file("extdata", package = "TransView"),
+     full = T, patt = "bam$")
> fn.macs <- dir(system.file("extdata", package = "TransView"),
+     full = T, patt = "xls$")
> fn.dm3.gtf <- dir(system.file("extdata", package = "TransView"),
+     full = T, patt = "gtf.gz$")[1]
> fn.mm9.gtf <- dir(system.file("extdata", package = "TransView"),
+     full = T, patt = "gtf.gz$")[2]
> fn.pas_paired <- untreated1_chr4()
> fn.pas_upaired <- untreated3_chr4()
```

2.1 Creating and accessing the DensityContainer class

To create the read density map, the central function `parseReads` has to be called. The function takes `filename` as an argument and returns a `DensityContainer` object.

```
> dens.ind <- parseReads(fn.chipseq.bam[2], verbose = 0,
+     description = "Induced")
> dens.wt <- parseReads(fn.chipseq.bam[1], verbose = 0,
+     description = "Basal")
```

During parsing, with `verbose` set to 1 the currently processed chromosome will be displayed and a warning will be issued, stating the chromosomes that were found in the SAM/BAM header, but missing in the file body. Accessing the object is fairly simple using the corresponding getter methods. A brief overview will be plotted by calling the show method:

```
> dens.ind

class: DensityContainer
  Experiment: Induced
  Source: /tmp/RtmpxQwtAD/Rinst3b615526683028/TransView/extdata/example_sam.bam
  Spliced: FALSE
  Paired: FALSE
  Filtered: FALSE
  Reads in file: 10395
  Reads used: 10395
  Coverage: 0.0002331195
  Local Coverage: 35.89296
  Max Score: 170
  Local Max Score: 170
```

3

```
  Low Quality / Unmapped: 0
  Strands: both
  Memory usage [MB]: 0.3
 Available Slots:
data_pointer histogram size env ex_name origin spliced paired filtered readthrough_pairs str
compression chromosomes filtered_reads pos neg lcoverage lmaxScore fmapmass lsize
nreads gsize gcoverage maxScore lowqual paired_reads proper_pairs collapsed
Chromosomes: chr1|chr2|chr3|chr6|chr8|chr9|chr10|chr11|chr14|chr16|chr17|chr18
```

The meaning of the available slots which can all be directly accessed with the corresponding method, are sub divided into 3 sections:

1. Basic information

   ex_name A user provided string to define a name of this data set

   origin File name of the source file

   spliced Should the class be treated like an RNA-Seq experiment for e.g. plotTV?

   paired Did the source file contain reads with proper pairs?

   readthrough_pairs Determines if the pairs were used from start to end or if individually.

   filtered Is there a range filter in place? If yes, slicing should be only conducted using the same filter

   strands Which strands were parsed. Can be "+", "-" or "both"

   size The current memory occupied by the whole object including the read density maps.

2. Reads before filtering

   nreads Total number of reads in the file, regardless of mapping state

   gcoverage Total gcoverage computed by total map mass/(chromosome end - chromosome start). Chromosome length derived from the SAM/BAM header

   maxScore Maximum read pileup found in file

   lowqual Amount of reads that did not pass the quality score set by min_quality or were not mapped

   paired_reads Amount of reads having multiple segments in sequencing

   proper_pairs Amount of pairs with each segment properly aligned according to the aligner

   collapsed If `maxDups` of `parseReads` is in place, the reads at the same position and strand exceeding this value will be counted here.

3. Reads after filtering

   compression Size of a gap triggering an index event

   chromosomes Character string with the chromosomes with reads used for map construction

   filtered_reads Amount of reads after filtering

   pos Reads used from the forward strand

   neg Reads used from the reverse strand

   lcoverage Local coverage which is computed by filtered map mass/covered region

   lmaxScore Maximum score of the density maps

   fmapmass Map mass after quality filtering

All these slots can also be accessed by the function `tvStats` which returns a list with these values and the corresponding names of the slots.

```
> tvs <- tvStats(dens.ind)
> tvs$nreads

[1] 10395
```

Furthermore a histogram of mean read densities of windows across the read densities is computed during parsing. The window size can be changed by the `hwindow` argument to `parseReads` and the histogram can be retrieved by calling the method `histogram`

```
> dens.ind.hist <- histogram(dens.ind)
> dens.wt.hist <- histogram(dens.wt)


> barplot(rbind(dens.ind.hist[1:50] + 1, dens.wt.hist[1:50] +
+     1), xlab = "Read count", ylab = "Positions",
+     col = c("blue", "red"), beside = T, legend.text = c("Induced",
+         "Basal"), log = "y", args.legend = list(x = "topright"))
```
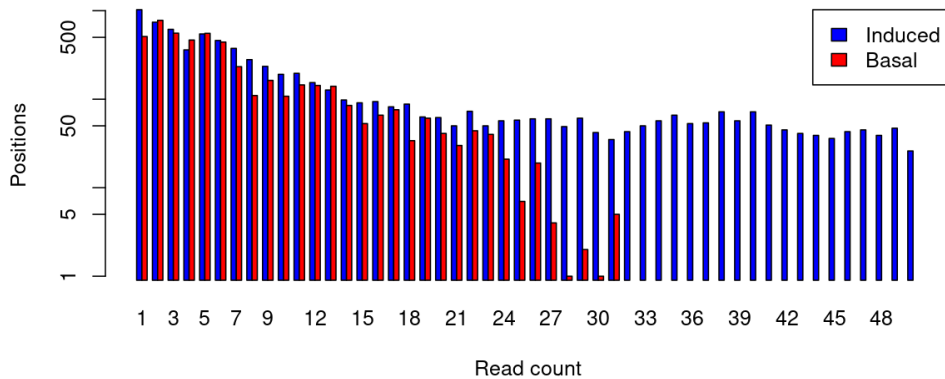

Figure 1: Bar plot showing the read density distribution in the data set

2.2 Parsing details and filtering

`parseReads` determines the read length including insertions, matches and any soft clipping as specified in the CIGAR string for each read. The parsing process can be customized by several options as stated in the help file.

All read density maps are stored in an indexed vector per chromosome. During parsing the overlapping reads will be stored uncompressed to keep accession speed at a maximum. If continuous score maps are interrupted by gaps which are defined by `compression`, the gaps will be skipped and a linear index will be set. The higher this value will be, the lower the amount of indexing and therefore the faster the accession speed. A very low value can lead to a very big index and therefore slower accession speeds. Memory usage on the other side can profit significantly and is therefore recommended to leave this value at the default of 1. The benefit however highly depends on the coverage of the data set.

As a further effort to keep the memory food print for genome wide maps at a minimum the read pile ups are stored as 16bit integers which means that they can have a maximum value of 65535. If this value will be exceeded, the pile up will be capped to this value. A warning will be issued. To avoid the capping one possibility is to set quality thresholds such as `max_dups` for the maximum amount of reads at the exact same position and re-run the analysis.

If memory availability is an issue, a range filter can be set with the argument `filter`. As a consequence read map assembly will be restricted to this ranges. This can save a great amount of memory but requires all further slicing operations to be restricted to these ranges only. To demonstrate the filter option, a convenience function called `macs2gr` will be used to load an attached sample MACS run with matching peak regions. In principle however, any data.frame with three columns containing, chromosomes, starts and ends or a corresponding RangedData object would be sufficient.

```
> peaks <- macs2gr(fn.macs, psize = 500)

Version 2.09+ detected
21 peaks matching

> dens.ind.filt <- parseReads(fn.chipseq.bam[2], verbose = 0,
+     description = "ChIP", set_filter = peaks[1, ])
> us <- slice1(dens.ind, chrom = as.character(seqnames(peaks[1])),
+     start = start(peaks[1]), end = end(peaks[1]))
> rs <- slice1(dens.ind.filt, chrom = as.character(seqnames(peaks[1])),
+     start = start(peaks[1]), end = end(peaks[1]))
> all(us == rs)

[1] TRUE

> size(dens.ind.filt)

[1] 0.2666473

> size(dens.ind)

[1] 0.2950287
```

Do note that range filtering is applied **after** the density map generation, and therefore any overlapping reads are included in the region. In the example above the memory advantage is small. With real data however, it can make a difference as large as one gigabyte for a mammalian genome. For efficient repeated queries `sliceN` returns a list of numeric vectors with the peak names as names.

```
> slices.ind <- sliceN(dens.ind, ranges = peaks)
> slices.wt <- sliceN(dens.wt, ranges = peaks)
```

All `slice` methods also provide an option to handle background reads. These can be either subtracted after correction for total reads in the data set or a fold change can be calculated and is returned on $log2$ scale.

```
> slices.nobckgd <- sliceN(dens.ind, control = dens.wt,
+     ranges = peaks, treads_norm = F)
> plot(slices.ind[[1]], ylab = "Total reads")
> lines(slices.wt[[1]], type = "p", col = 4)
> lines(slices.nobckgd[[1]], type = "p", col = 2)
> legend(400, 150, c("Induced", "Basal", "Induced corrected"),
+     col = c(1, 4, 2), pch = "o", bty = "n")
> slices.nobckgd.fc <- sliceN(dens.ind, ranges = peaks,
+     control = dens.wt, input_method = "/", treads_norm = F)
> summary(slices.nobckgd.fc[[1]])
```

Total read correction was turned off in this example, since the example regions were selected for demonstration purposes and do not reflect the total amount of reads.

Data from spliced data sets such as RNA-Seq data can be parsed analogous to ungapped reads.

```
> dens.pas_upaired <- parseReads(fn.pas_upaired, spliced = T,
+     verbose = 0, description = "Unpaired")
> dens.pas_paired <- parseReads(fn.pas_paired, spliced = T,
+     verbose = 0, description = "Paired")
```

The argument `spliced` has no influence on the resulting density map and will only be important for plotting and annotation with `plotTV` and all CIGAR operations are used by default. One exception are paired end reads however, if parsed with the `read_through` argument set the CIGAR string will be ignored and densities will be calculated from the beginning of the left to the end of the right read.

2.3 Read map accession

Density maps generated by `parseReads` can only be accessed with one of the included slicing methods such as `slice1` or `slice1T`. The latter reconstructs the original transcript structure typically of a RNA-Seq data set. The transcript strucure information will be taken from a data.frame or *GRanges* object witch one row per exon. A data frame should have four columns with chromosomes, starts, ends and strands as well as a fifth column with the transcript_id of the associated transcript. Alternatively a GTF file can be provided and converted to a GenomicRanges object using the function `gtf2gr`.

```
> gtf.mm9 <- gtf2gr(fn.mm9.gtf)

172 rows matching

> gtf.dm3 <- gtf2gr(fn.dm3.gtf)

2862 rows matching

> head(gtf.mm9)

GRanges object with 6 ranges and 3 metadata columns:
            seqnames              ranges strand | transcript_id
               <Rle>           <IRanges>  <Rle> |   <character>
  NM_008057.1    chr1 59538991-59543799      + |     NM_008057
  NM_144528.9   chr10 79221260-79221981      - |     NM_144528
  NM_144528.8   chr10 79223536-79223651      - |     NM_144528
  NM_144528.7   chr10 79223734-79223827      - |     NM_144528
  NM_144528.6   chr10 79224175-79224244      - |     NM_144528
  NM_144528.5   chr10 79224310-79224378      - |     NM_144528
              exon_id     gene_id
            <numeric> <character>
  NM_008057.1       1   NM_008057
  NM_144528.9       9   NM_144528
  NM_144528.8       8   NM_144528
  NM_144528.7       7   NM_144528
  NM_144528.6       6   NM_144528
  NM_144528.5       5   NM_144528
  -------
  seqinfo: 12 sequences from an unspecified genome; no seqlengths
```

`gtf2gr` efficiently parses the GTF formats used by ENSEMBL (ensembl.org/info/data/ftp) and UCSC (e.g. the UCSC table browser) and can be used as a whole genome transcript library. This information can also be provided to `parseReads` as a filter if memory space is limited and subsequent slice operations are limited to these transcripts only.

```
> dens.pas_paired.filt <- parseReads(fn.pas_paired,
+     spliced = T, verbose = 0, description = "RNA-Seq",
+     set_filter = gtf.dm3)
> size(dens.pas_paired)/size(dens.pas_paired.filt)
```

```
[1] 1.719504
```

Despite of the limited amount of reads in the demo data set, the size advantage of the filtered parsing is here already about factor 2. Slice operations can now be performed using slice1T for single queries or sliceNT for multiple.

```
> slices.exprs.pangolin <- slice1T(dens.pas_paired.filt,
+     "NM_001014685", gtf.dm3, stranded = T, concatenate = F)
> pangolin.exon.12 <- slices.exprs.pangolin[["NM_001014685.12"]]
> pangolin.exon.12
```

```
 [1] 30 30 30 30 31 32 32 33 33 33 34 36 35 35 35 34 34 35 32 32
[21] 32 32 32 32 33 33 33 33 32 32 32 31 31 32 26 27 25 25 25 25
[41] 25 24 23 23 24 23 23 23 23 23 23 24 25 26 26 26 27 25 25 26
[61] 27 28 28 21 21 21 21 21 21 21
```

By default, an integer vector with the read densities will be returned. If the precise transcript structure is required, concatenate can be switched off and with stranded set to FALSE all read maps on the reverse strand will be reversed so that all returned read maps are store from 5 prime to 3 prime. Also here a control experiment can be provided for background handling. Analogous to sliceN, there is also an optimized version for multiple queries called sliceNT that takes a character vector of identifiers and returns a list with all requested transcript structures.

```
> #Just for demonstration all refseq ids are taken, not recommended for a full sized GTF!
> all_ids<-unique(mcols(gtf.dm3)$transcript_id)
> slices.exprs<-sliceNT(dens.pas_paired,all_ids,gtf.dm3,stranded=T,concatenate=F)
> pangolin.all.exon.12<-slices.exprs[["NM_001014685.12"]]
> all(pangolin.exon.12,pangolin.all.exon.12)
```

```
[1] TRUE
```

In order to associate ChIP-Seq peaks to the next transcriptional start side (TSS), an annotation step is required. This can be done with several available bioconductor packages. However also *TransView* contains such a convenience function called annotatePeaks.

```
> peaks.anno <- annotatePeaks(peaks = peaks, gtf = gtf.mm9,
+     limit = 2000)
```

annotatePeaks will use the digested gtf file to find the TSS of the next transcript to the peak center. The resulting *GRanges* object will contain an updated or added meta data column with the associated transcript identifier. If multiple transcripts are found using the same TSS, the first transcript will be used arbitrarily. Optionally a reference RNA-Seq data set can be provided to resolve the ambiguous TSS associations based on a function. By default the transcript will be selected having the largest amount of reads over the length of the transcript.

Additionally it might be of interest to slice or visualize the promoter region of the transcript associated to the peak rather than the peak region itself. This can be achieved by the function peak2tss which takes the digested gtf as well as a character vector of matching IDs as an argument.

```
> peaks.tss <- peak2tss(peaks.anno, gtf.mm9, peak_len = 1000)
```

3 Plotting read density maps

To get a visual impression about the outcome of ChIP-Seq experiments or the correlation to expression data, it is often desirable to visualize the whole data set. This can be done by using false color plots centering on genomic features such as enhancers or the TSS or on the peak itself. *TransView* contains the flexible `plotTV` function that can perform plotting of peak profile plots optionally along with results from RNA-Seq, micro arrays or any other matching experiment. All plots will be generated using the `image` function, however scaling and clustering can be highly customized. As the attached demo data is very small to keep the overall package size low, only a limited functionality can be shown here. The package however is designed to efficiently display hundreds to thousands of peaks along with expression data from the corresponding genes.

3.1 Peak profile plots

In order to plot a simple overview of the read density distribution in a heat map like manner Fig2a, only two input objects are needed: an object of class *DensityContainer* which is generated by `parseReads` and a *GRanges* with the regions of interest from e.g. `macs2gr`. Instead of two plots with experiment and control, also the background subtracted density map can be plotted alone (Fig2b).
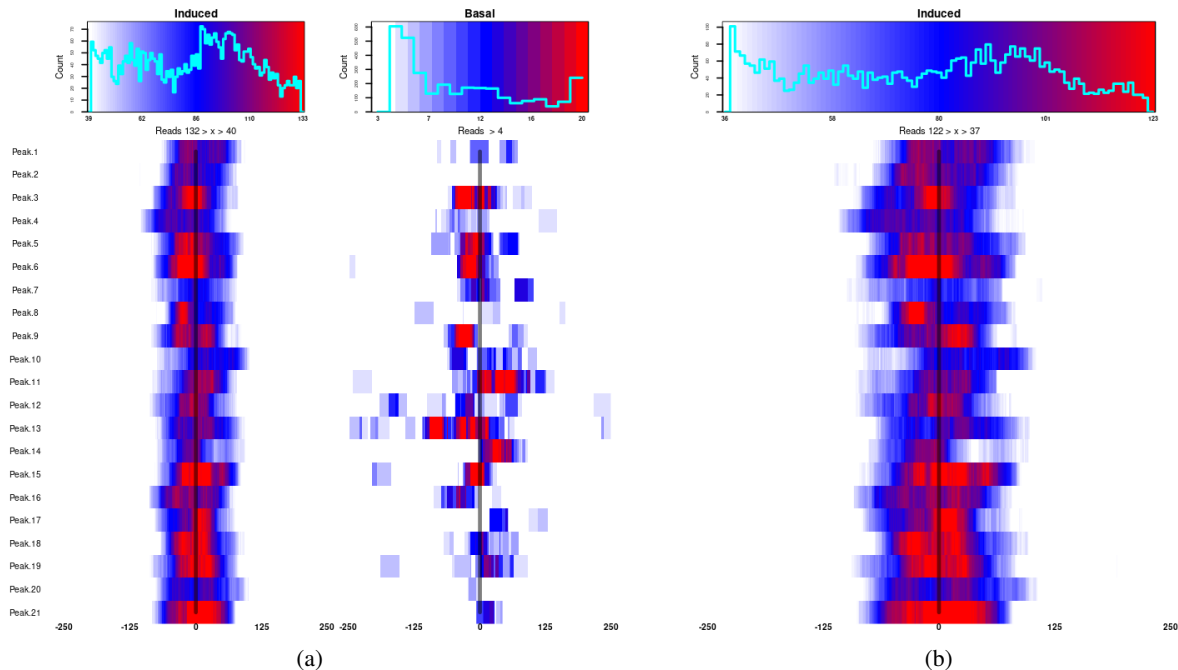

Figure 2: Example of peak profile plots. All plots were generated without total read correction due to the reduced amount of reads in the sample data sets. Figure 2a shows the induced sample along with the basal control. The black line indicates the mid point of the region, which is the MACS peak summit in this case. Note the different scales because `scaling` was set to "individual". Figure 2b shows the same plot with the basal control reads subtracted from the induced sample.

Fig2a and Fig2b have been generated with the following commands:

```
> cluster_results <- plotTV(dens.ind, dens.wt, regions = peaks.anno,
+     show_names = T, norm_readc = F, scale = "individual",
+     verbose = 0)
```

9

```
> cluster_results <- plotTV(dens.ind, regions = peaks.anno,
+       control = c(dens.wt), show_names = F, norm_readc = F,
+       verbose = 0)
```

There are several options to customize the visual appearance, which are explained in the corresponding help file. One of the most important is `color_spread`. This argument regulates the saturation levels on both ends. E.g. if set to `c(0.1, 0.05`, 10 percent of the highest and 5 of the lowest read density levels will be displayed as saturated colors, which are red and white by default.

3.2 Transcript profile plots

In addition to peak profiles designed to visualize ChIP-Seq experiments, *TransView*can also visualize expression based sequencing experiments such as from RNA-Seq. With appropriate scaling and clustering these plots can give a insight into the kind of transcription taking place over the whole gene body. Genes of individual experiments might cluster differentially according to the transcriptional status over the gene body. Although the interpretaion of such clusters might be complex, individual clusters can reveal a visual indication of correlated binding and transcription events such as Polymerase II activity as well as mRNA stability.

As opposed to peaks, transcripts have varying lengths that can not be trivially cut to a uniform size. *TransView*therefore linearly interpolates all transcripts into `ex_windows` amount of points using the method specified by `bin_method`. Furthermore all density profiles on the reverse strand will be reversed, so that all transcripts have their 5 and 3 prime site on the left and the right side of the image respectively Fig3a. In addition to transcript profile plots, ordinary heat maps of a matrix can be plotted as well Fig3b. Since the functionality of heat map plotting in *TransView*is rather limited compared to specialized packages such as *heatmap.2*, plotting heat maps is not intended to be used for stand alone plots but rather in conjunction with peak profile plots.
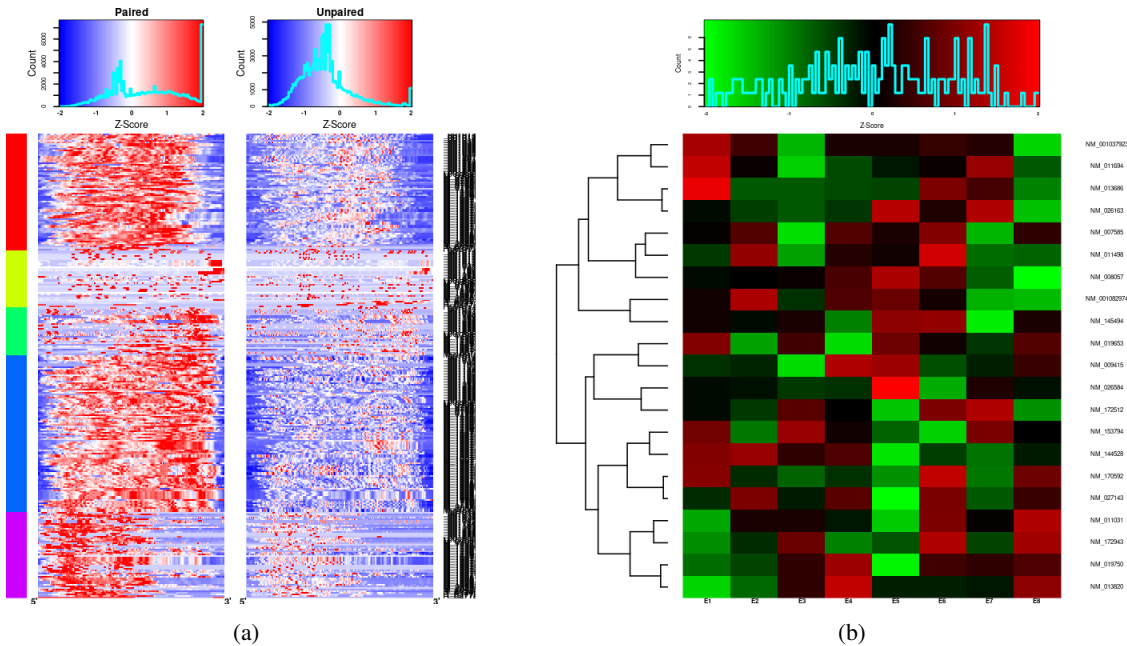

Figure 3: Example of transcript profile plots. In figure 3a, the paired and single end data sets of pasillaBamSubset are plotted. All transcripts found on Chr4 according to the provided GTF were used and k-means clustered according to their z-score. In figure 3b, a randomized sample heat map is shown. The data was passed as a matrix which could contain the results of i.e. micro array experiment.

10

Both expression based plots are converted to z-scores with rows of mean zero and standard deviation of one. If multiple transcript profile plots are supplied, z-scores are computed across all experiments. Fig3a and Fig3b have been generated with the following commands:

```
> genes2plot <- unique(mcols(gtf.dm3)$transcript_id)
> cluster_results <- plotTV(dens.pas_paired, dens.pas_upaired,
+     regions = genes2plot, gtf = gtf.dm3, show_names = T,
+     cluster = 5, verbose = 0, ex_windows = 300)

> ngenes <- length(peaks.anno)
> fake.array <- matrix(rnorm(n = ngenes * 8, mean = 10,
+     sd = 2), nrow = ngenes, ncol = 8, dimnames = list(paste(rep("Gene",
+     ngenes), 1:ngenes), paste("E", 1:8, sep = "")))

> cluster_results2 <- plotTV(fake.array, regions = peaks.anno,
+     show_names = T, gclust = "expression", cluster = "hc_sp",
+     label_size = 0.7, verbose = 0)
```

3.3 Combining peak profile plots and expression

For comprehensive visualization of the transcriptional events including transcription factor binding and change of histone modifications and their impact on the transcriptional response, peak profile plots can be combined together with expression data from e.g. RNA-Seq or micro arrays as shown in Fig4. *TransView* will place the peak profile plots always on the left side and in the order passed to the function. Heat maps of a provided matrix or transcript profile plots are placed on the right side accordingly.

Row based clustering can be performed in groups. gclust can be set to "expression" or "peaks" to restrict the clustering to transcript profile plots or peak profile plots respectively. The group not being clustered will be reordered based on the respective clustering results. gclust can also be set to "both" to achieve plot wide clustering. The two options available here for the clustering algorithm are hierarchical clustering and k-means clustering. If hierarchical clustering is chosen, a dendrogram will be displayed on the left of the figure. Since hierarchical clustering in *TransView* is performed on the distances within the correlation matrix, a correlation method can be chosen to account for different data distributions. To avoid rows with a standard deviation of zero, the argument remove_lowex can be used to set a threshold to exclude transcripts that are not expressed which is observed frequently in RNA-Seq data sets with low coverage. If k-means clustering is chosen on the other hand, a colored bar will be displayed on the left side of the figure. Alternatively a vector can be provided with ranks for user defined resorting of the rows.
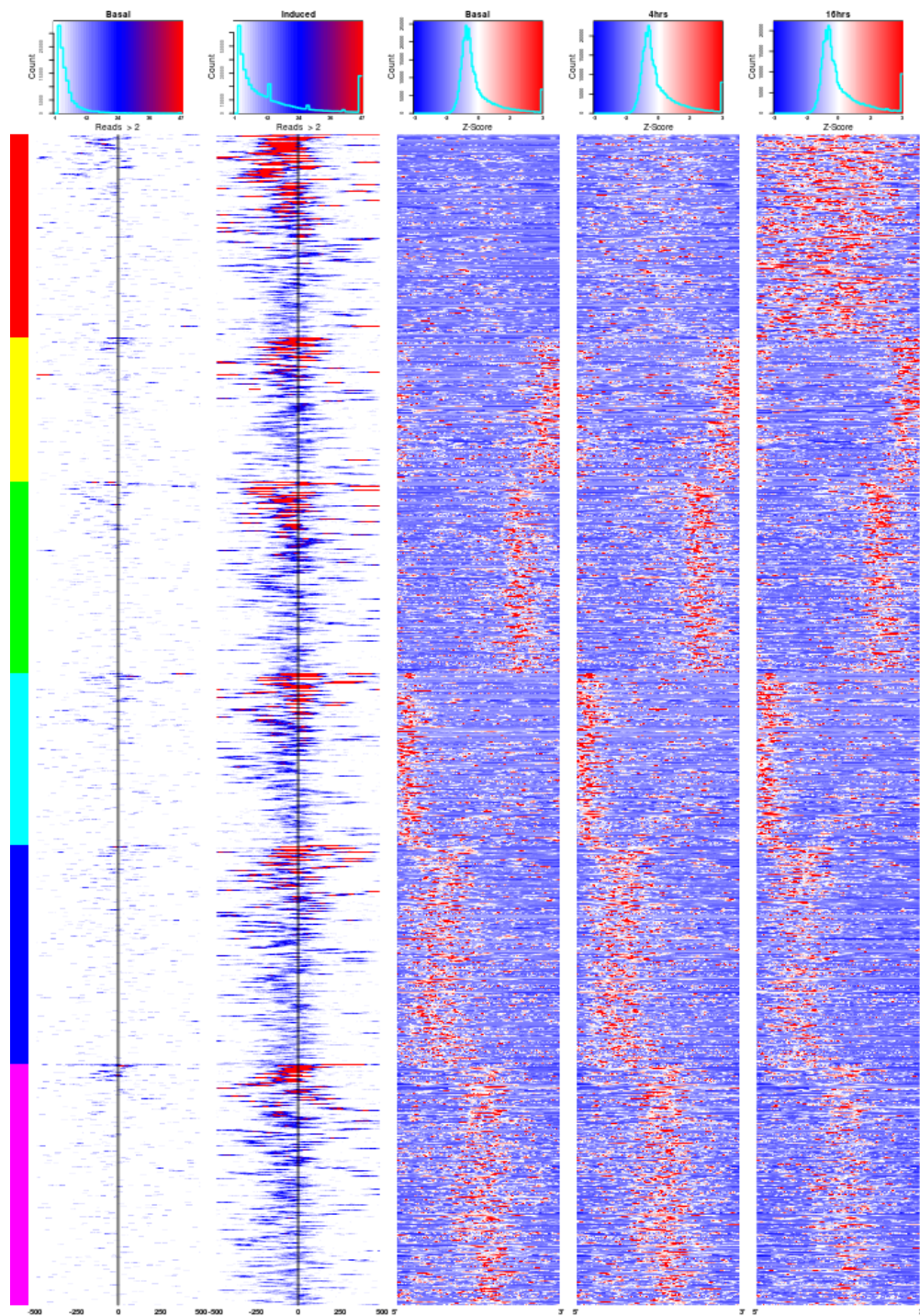
Figure 4: Mixed profile plot with 5 panels. 1000 ChIP-Seq peaks of basal (1) transcription factor levels and induced (2) levels are shown in the first two panels. The corresponding RNA-Seq time course is shown on the right side (3 to 5). k-means clustering was applied to partition the RNA-Seq data into 6 clusters. Cluster 1 (red) clearly contains genes with a strong gain in expression levels over time.

3.4 Return value and clustering

As a return value an object of class *TVResults* will be returned. This object contains the most important parameters and results of a call to plotTV.

```
> cluster_results

class: TVResults
 [1] "          " "     Matrix" "     global" "      hc_sp"
 [5] "     FALSE" "         0" "      TRUE" "       0.7"
 [9] "       0.5" "white, b...." "    redgreen" " 0.05, 0.05"
[13] "      auto" "      auto" "        NA" "       100"
[17] " expression" "      TRUE" "     FALSE" "      TRUE"
[21] "     2, 1" "         0" "         0" "     FALSE"
[25] "         1" "         0" "         0" "      TRUE"
```

To track the clustering performed by plotTV, the order of the peaks and the corresponding clusters of each row can be easily accessed.

```
> cluster_order(cluster_results)[1:10]
> clusters(cluster_results)[1:10]
> summaryTV(cluster_results2)
```

The order of the individual clusters can also be passed to plotTV in order to reproduce a plot clustered with k-means clustering.

```
> plotTV(dens.pas_paired, dens.pas_upaired, regions = genes2plot,
+     gtf = gtf.dm3, show_names = T, rowv = cluster_results,
+     verbose = 0, ex_windows = 300)
```

Apart from the basic order of the peaks and genes after clustering and the corresponding clusters, a data frame with the summarized scores can be extracted.

```
> cluster_df <- plotTVData(cluster_results)
```

3.5 Further visualization

The data frame returned by plotTVData contains the averaged scores of each data set passed to plotTV. The corresponding summaries can be visualized with e.g. *ggplot2*:

```
> ggplot(cluster_df, aes(x = Position, y = Average_scores,
+     color = Sample)) + geom_point() + facet_wrap(Plot ~
+     Cluster, scales = "free", ncol = 5)
```

To visualize individual peaks, *TransView* contains the convenience function meltPeak extracting the normalized read densities from the individual experiments. Furthermore the densities are wrapped into a data.frame that can be directly passed to *ggplot2*:

```
> peak1.df <- meltPeak(dens.ind, dens.wt, region = peaks.tss["Peak.1"],
+     peak_windows = 100, rpm = F)
> ggplot(peak1.df, aes(x = Position, y = Reads, color = Label)) +
+     geom_line(size = 2)
```

Note that rpm is set to FALSE, since the filtered map mass in the toy example contains only a small subset of the original map mass. Normalizing to map mass would result in artificailly upscaled densities in the control experiment:

13

```
> peak1.df <- meltPeak(dens.ind, dens.wt, region = peaks.tss["Peak.1"],
+     peak_windows = 100, rpm = T)
> ggplot(peak1.df, aes(x = Position, y = NormalizedReads,
+     color = Label)) + geom_line(size = 2)
```

4 Memory usage and speed considerations

As the typical mammalian genome comprises of billions of base pairs and each integer in R consumes about 4 bytes, special attention has to be given to memory usage and accession speed. Techniques like Rle can reduce memory consumption of the typical read density map to a minimum. For a large amount of repeated queries however, any form of compression will slow down accession speed. As *TransView* is designed to provide fast, genome wide access to memory maps and large memory configurations are becoming the norm, no further attempt to reduce memory consumption is undertaken apart from 16bit integer storage and gap indexing. With a few adjustments however, *TransView* can be also run on small configurations such as lap tops:

- If the regions to be sliced are already known before parsing, with `set_filter` the memory storage can be reduced to a minimum.

- With the arguments `paired_only`,`min_quality` and `unique_only` the amount of gapped regions can be increased and the amount of memory consumption reduced accordingly if applicable.

- If the genome wide density maps are not needed anymore, the space occupied by the current R session can be reduced by applying the `rmTV` function.

The actual space occupied by the read maps largely depends on the coverage and the distribution of the reads. By far the highest consumption is observed with high coverage ChIP-Seq data sets. During the testing phase of this package, for a human genome data set with $1e8$ reads, 1.5GB was observed. For data sets with more pronounced read pile ups like from RNA-Seq, $2e7$ reads were observed to occupy around 300MB on the other hand. The advantage of this approach is that data accession with e.g. `sliceN` is extremely fast. On the test system (Core i5, 4GB ram), fetching $1e5$, 1kbp large regions was completed in less than a second. BAM file parsing and read map generation speed on the other hand depend almost linearly on I/O speed and the amount of reads provided. On a Linux configuration, an average of $1e6$ reads per second can be expected.

This vignette was built with the following versions of R and

```
> sessionInfo()

R version 4.3.1 (2023-06-16)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_GB              LC_COLLATE=C
 [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

time zone: America/New_York
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets
[7] methods   base

other attached packages:
[1] pasillaBamSubset_0.39.0 TransView_1.46.0
[3] GenomicRanges_1.54.0    GenomeInfoDb_1.38.0
[5] IRanges_2.36.0          S4Vectors_0.40.0
[7] BiocGenerics_0.48.0

loaded via a namespace (and not attached):
 [1] zlibbioc_1.48.0         gtools_3.9.4
 [3] GenomeInfoDbData_1.2.11 KernSmooth_2.23-22
 [5] XVector_0.42.0          RCurl_1.98-1.12
 [7] bitops_1.0-7            caTools_1.18.2
 [9] compiler_4.3.1          tools_4.3.1
[11] gplots_3.1.3
```

References

[Li *et al*., 2009] Li *et al*. (2009) The Sequence alignment/map (SAM) format and SAMtools, *Bioinformatics*, **25**, 2078-9.

[Zhang *et al*., 2008] Zhang *et al*. (2008) Model-based Analysis of ChIP-Seq (MACS), *Genome Biol*, **vol. 9 (9)**, pp. R137.