The wavClusteR package

(version 2.0)

Federico Comoglio and Cem Sievers
Department of Biosystems Science and Engineering
ETH Zürich, Basel, Switzerland
federico.comoglio@bsse.ethz.ch
cem.sievers@bsse.ethz.ch

April 16, 2015

Abstract

Different recently developed next-generation sequencing based methods (e.g. PAR-CLIP or Bisulphite sequencing) specifically induce nucleotide substitutions within the short reads with respect to the reference genome. This package provides functions for the analysis of the data obtained by such methods - with a major focus on PAR-CLIP - and exploits the experimentally induced substitutions in order to identify high confidence signals, such as RNA-binding sites, in the data. The workflow consists of two steps; (i) the estimation of a non-parametri two-component mixture model, identifying substitution frequencies most affected by the experimental procedure; (ii) a binding sites (clusters) identification algorithm which resolves clusters at high resolution. Key functions support multicore computing, if available. For a detailed description of the method see [1, 2].

Contents

1 Preparing the input

Starting with a fastq file, a commonly used short read format, the short reads should be aligned to the reference genome using a short read aligner, e.g. Bowtie [3]. The output file (e.g. in SAM format) should then be converted to BAM format (e.g. using `samtools view`, `http://samtools.sourceforge.net/samtools.shtml`) and sorted (e.g. using `samtools sort`). Since *wavClusteR* requires an indexed BAM file containing the short read alignments, an index file (`.bai`) should be generated from the sorted BAM file (see, e.g. `samtools index`).
The following code provides an example of the steps described above using the samtools toolkit. The first line is pseudo code. Please replace it with the aligner specific syntax.

```
ALIGN: sample.fastq -> sample.sam

CONVERT: samtools view -b -S sample.sam  -o sample.bam

SORT: samtools sort sample.bam sample_sorted

INDEXING: samtools index sample_sorted.bam
```

1.1 Example dataset

In this vignette, we consider as an example a chunk of a published Argonaute 2 (AGO2) PAR-CLIP data set obtained from human HEK293 cells [4]. This chunk contains reads mapping to chromosome X in the interval: 23996166 - 24023263. This data set is provided in *wavClusteR* 2.0.

1.2 Importing short reads into the R session

An indexed BAM file can be loaded into the R session using the `readSortedBam` function. This calls `scanBam` from *Rsamtools* [5] and extracts the mismatch MD field and the read sequence from the BAM file, returning a `GRanges` object.

```
> library(wavClusteR)
> filename <- system.file( "extdata", "example.bam", package = "wavClusteR" )
> Bam <- readSortedBam(filename = filename)
> Bam

GRanges object with 5358 ranges and 2 metadata columns:
         seqnames               ranges strand   |
            <Rle>            <IRanges>  <Rle>   |
     [1]     chrX [24001819, 24001844]      -   |
     [2]     chrX [24001819, 24001843]      -   |
     [3]     chrX [24001834, 24001863]      -   |
     [4]     chrX [24001836, 24001865]      -   |
     [5]     chrX [24001841, 24001876]      -   |
     ...      ...                 ...    ... ...
  [5354]     chrX [24023018, 24023051]      -   |
  [5355]     chrX [24023018, 24023051]      -   |
```

```
  [5356]     chrX [24023019, 24023051]      -   |
  [5357]     chrX [24023019, 24023051]      -   |
  [5358]     chrX [24023067, 24023090]      -   |
                                       qseq          MD
                                <DNAStringSet> <character>
     [1]           CAGAGATAAAGAAGTATATTTTAAAG          26
     [2]            CAGAGATAAAGAAGTATATTTTAAG         24A0
     [3]       ATATTTTAGAGATTAAAAATATTTTATTTA         8A21
     [4]       TTTTTAAAGATTAAGAATATTTTATTTAAA     0A13A15
     [5] AAAGATTAAAAATATTTTATTTAAGCTTTTCTTCAT        24A11
     ...                                 ...          ...
  [5354]   GTTTCACAGCGTTTTGGAGGAAAAAAAAATATGT        10A23
  [5355]   GTTTCACAGCGTTTTGGAGGAAAAAAAAATATGT        10A23
  [5356]    TTTCACAGCGTTTTGGAGGAAAAAAAAATATGT         9A23
  [5357]    TTTCACAGCGTTTTGGAGGAAAAAAAAATATGT         9A23
  [5358]             CAAAGGCGCGAATGGGTTTATTTT         9A14
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

1.3 Extracting the informative positions used for model parameter estimation

To estimate the mixture model both mixing coefficients and density functions (components) have to be estimated from the data. To this purpose, genome-wide substitutions are first identified and filtered according to a minimum coverage value at substitutions. The minimum coverage, which should be chosen to account for variables such as sequencing depth, provides a way to select the positions used for parameter estimation. Hence, it can be used to tune the stringency of the analysis. There is no obvious theoretical justification to find the optimal minimum coverage. However since relative substitution frequencies have to be computed for parameter estimation, the minimum coverage will influence the variance of the estimate. The lowest minimum coverage used for our analysis was 10.

The `getAllSub` function identifies the genomic positions that show at least one substitution and satisfy the minimum coverage requirement. It returns a `GRanges` object specifying the genomic position, the strand, the observed substitution (e.g. "TC" implies a T in the reference genome and a C in the read), the strand-specific coverage and the number of observed substitutions at the specific position.

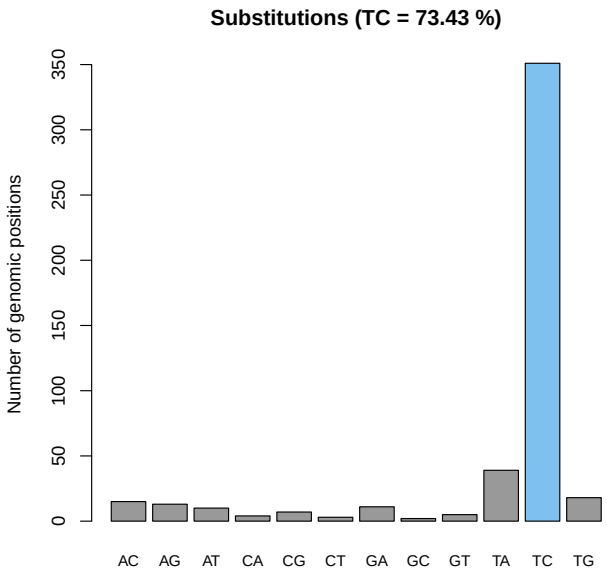
```
> countTable <- getAllSub( Bam, minCov = 10 )
> head( countTable )
```

```
GRanges object with 6 ranges and 3 metadata columns:
      seqnames              ranges strand | substitutions  coverage     count
         <Rle>           <IRanges>  <Rle> |   <character> <numeric> <integer>
  [1]     chrX [24001959, 24001959]      - |            TC        17         2
  [2]     chrX [24001973, 24001973]      - |            TC        17        12
  [3]     chrX [24001977, 24001977]      - |            TC        13         1
  [4]     chrX [24002046, 24002046]      - |            TC        10         1
  [5]     chrX [24002057, 24002057]      - |            TC        10         6
  [6]     chrX [24002147, 24002147]      - |            TC        22         3
```

```
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Once all substitutions are computed, the corresponding substitution profile can be plotted with `plotSubstitutions`. This function returns a barplot showing the total number of genomic positions that exhibit a given type of substitution and highlights the substitution type that is expected to be generated by the experimental procedure. In addition, the percentage of substitution of this type with respect to all identified substitutions is indicated. This plot conveys information about the quality of the data and can be used to compare different data sets generated by the same experimental conditions.

```
> plotSubstitutions( countTable, highlight = "TC" )
```



2 Estimating the non-parametric mixture model

The genomic positions identified above (see `getAllSub`) are used to estimate the mixture model densities and mixing coefficients. The estimation is performed by the function `fitMixtureModel` for the substitution of interest. The function returns a list containing:

- the two mixing coefficients (`l1` and `l2`)
- the two individual components (`p1` and `p2`).
- the full density (`p`)

Given an observed relative substitution frequency, the model is used to compute the posterior probability that it was obtained by experimental induction.
Given the `GRanges` object `countTable` the model can be estimated as follows:

```
> model <- fitMixtureModel(countTable, substitution = "TC")        #not run
```

The small size of the example dataset would not allow a reliable model estimation. Therefore, the mixture model for the entire AGO2 dataset has been precomputed and is provided in *wavClusteR* for convenience. The model can be loaded as

```
> data(model)
> str(model)

List of 5
 $ l1: Named num 0.181
  ..- attr(*, "names")= chr "TC"
 $ l2: Named num 0.819
  ..- attr(*, "names")= chr "TC"
 $ p : num [1:999] 7.52 9.44 10.05 10.38 10.48 ...
 $ p1: num [1:999] 89.6 64.4 50.4 41.5 35.3 ...
 $ p2: num [1:999] 0 0 1.14 3.51 5 ...
```

Once the mixture model is estimated, the model fit can be inspected and the RSF interval most likely to be affected by the experimental procedure can be identified using the `getExpInterval` function. Besides then high-confidence RSF interval, two plots are returned. The first one illustrates the estimated densities p, p_1 and p_2 and ensuing log odds o

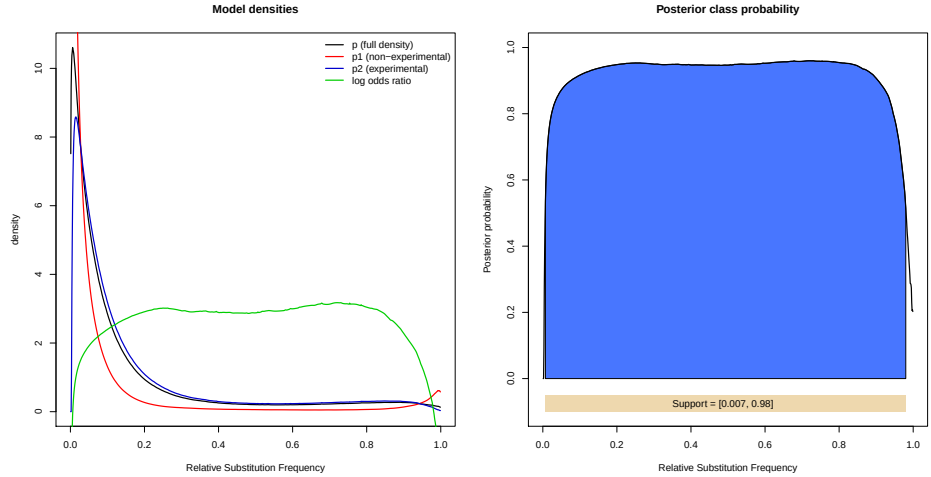
$$o = \log \frac{\mathrm{p(k = 2|x)}}{\mathrm{p(k = 1|x)}}$$

whereas the second plot shows the resulting posterior class probability, i.e. the probability that a given relative substitution frequency (RSF, horizontal axis) has been experimentally induced. The area under the curve corresponding to the returned RSF interval is colored, and the RSF interval indicated. By default, `getExpInterval` returns the RSF interval according to the Bayes classifier, i.e. RSF values having probability larger than or equal to 0.5.

```
> (support <- getExpInterval( model, bayes = TRUE ) )

$supportStart
[1] 0.007

$supportEnd
[1] 0.98
```



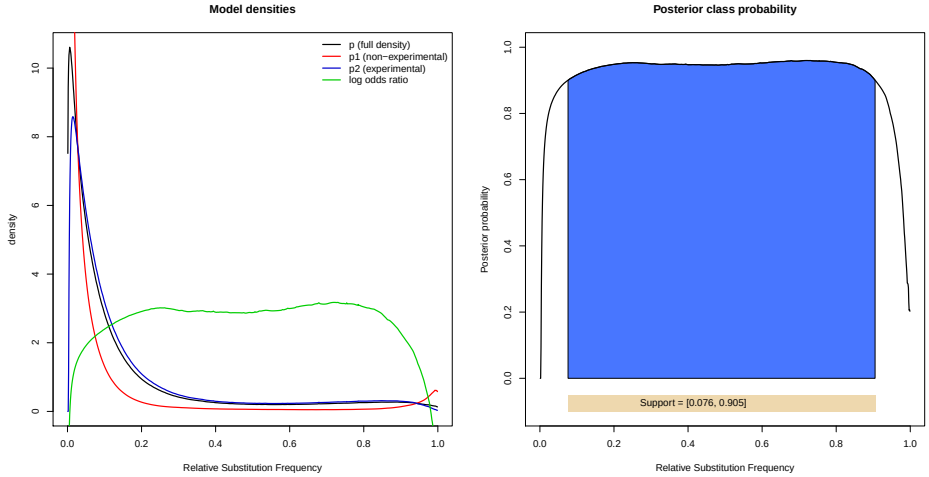
However, the user can modify the stringency of the analysis and determine a custom RSF interval in two ways:

1. By setting the `rightProb` and `leftProb` parameters to a desired posterior probability cutoff, e.g.

```
> (support <- getExpInterval( model, bayes = FALSE, leftProb = 0.9, rightProb = 0.9 ) )

$supportStart
[1] 0.076

$supportEnd
[1] 0.905
```

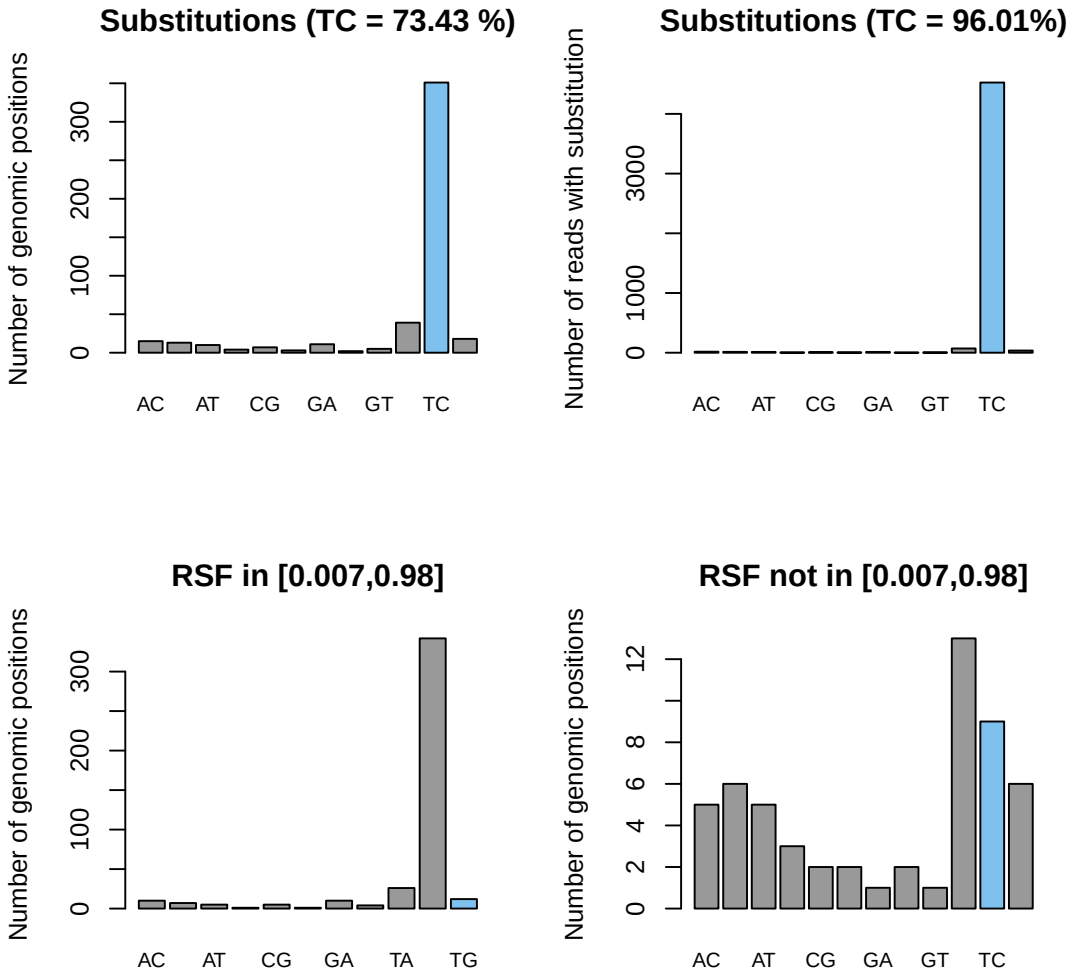


2. By inspecting the posterior class probability density and directly enter the RSF interval boundaries when calling high-confidence substitutions (see call to `getHighConfSub` function in the next section)

Finally, the model can be used to produce further diagnostic plots. Particularly, besides the barplot returned by `plotSubstitutions` (see previous section), the total number of reads carrying a given substitution and an RSF-based partitioning of genomic positions with substitutions is returned by

```
> plotSubstitutions( countTable, highlight = "TC", model )
```

```
within
FALSE  TRUE
   55   423
```



3 Identifying protein binding sites (clusters)

3.1 Filtering high confidence signal sites

High-confidence transitions are identified by the `getHighConfSub` function. The RSF interval returned by `getExpInterval` (see previous section) can either directly enter a `getHighConfSub` function call as

```
> highConfSub <- getHighConfSub( countTable,
+                                support = support,
+                                substitution = "TC" )
```

or, alternatively, the interval can be specified by the user as

```
> highConfSub <- getHighConfSub( countTable,
+                                supportStart = 0.2,
+                                supportEnd = 0.7,
+                                substitution = "TC" )
```

The function returns a `GRanges` object with genomic position, strand, strand-specific coverage (`coverage`), occurence (`count`), and relative substitution frequency (`rsf`) for each identified high-confidence substitution.

```
> head( highConfSub )
```

```
GRanges object with 6 ranges and 3 metadata columns:
      seqnames               ranges strand |  coverage     count
         <Rle>            <IRanges>  <Rle> | <numeric> <integer>
  [1]     chrX [24002057, 24002057]      - |        10         6
  [2]     chrX [24002331, 24002331]      - |        10         5
  [3]     chrX [24002335, 24002335]      - |        10         3
  [4]     chrX [24002348, 24002348]      - |        10         4
  [5]     chrX [24002677, 24002677]      - |        15         4
  [6]     chrX [24002680, 24002680]      - |        15         4
                    rsf
              <numeric>
  [1]               0.6
  [2]               0.5
  [3]               0.3
  [4]               0.4
  [5] 0.266666666666667
  [6] 0.266666666666667
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

3.2 Identifying cluster boundaries and computing cluster statistics

Binding sites (referred to as clusters) can be identified by the function `getClusters`. This function takes as input high-confidence substitution sites and the overall coverage across the genome, which can be computed using *GenomicRanges* as

```
> coverage <- coverage( Bam )
> coverage$chrX
```

```
integer-Rle of length 24023090 with 914 runs
  Lengths: 24001818       15        2        5 ...        1       15       24
  Values :        0        2        3        4 ...       21        0        1
```

Cluster boundaries in *wavClusteR* 2.0 are resolved by default using the Mini-Rank Norm (MRN) algorithm. Briefly, this algorithm finds an optimal cluster boundary for each high-confidence substitution by solving an optimization problem that integrates prior knowledge on the geometry of PAR-CLIP clusters. The algorithm first considers differences in the coverage function. Then, it removes background fluctuations via learning of a local background threshold or hard thresholding (default). This choice is controlled by the `threshold` parameter. The MRN algorithm proceeds by evaluating all ensuing possible cluster boundaries and computes ranking of boundary signals and cluster widths, which are finally used to find the optimum cluster boundary for each high-confidence substitution. Please see [2] for further details. Please notice that the MRN algorithm is strongly recommended as computationally faster (up to 10x) and more sensitive than the previously adopted cluster identification algorithm based on continuous wavelet transform (CWT) of the coverage function (see [1]). The latter computes the continuous wavelet transform (CWT) of the coverage function on a 1 kb window centered at a high-confidence substitution site. The minimum required signal-to-noise ratio can be specified with the parameter `snr` (default `snr`=3). Since multiple high-confidence substitution sites can localize in close proximity, the step size (controlled by the`step` parameter) can be set to values larger than 1 (default), such that the CWT is computed only if the subsequent high-confidence substitution is located further than the specified value from the previously considered position. Starting from the peak positions the cluster boundaries are then expanded. This algorithm is still maintained in *wavClusteR* 2.0 and can be called by setting `method = "cwt"` in the `getClusters` function.
Clusters can be computed by default as

```
> clusters <- getClusters( highConfSub = highConfSub,
+                          coverage = coverage,
+                          sortedBam = Bam,
+                          method = "mrn",
+                          threshold = 1,
+                          cores = 1 )
> clusters

GRanges object with 74 ranges and 0 metadata columns:
       seqnames              ranges strand
          <Rle>           <IRanges>  <Rle>
   [1]     chrX [24002044, 24002057]      -
   [2]     chrX [24002319, 24002339]      -
   [3]     chrX [24002319, 24002339]      -
   [4]     chrX [24002338, 24002348]      -
   [5]     chrX [24002668, 24002683]      -
   ...      ...                 ...    ...
  [70]     chrX [24006170, 24006191]      -
  [71]     chrX [24006533, 24006554]      -
  [72]     chrX [24007061, 24007068]      -
  [73]     chrX [24007061, 24007083]      -
  [74]     chrX [24007061, 24007083]      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

by using the MRN algorithm. Two options are available here:

1. Hard thresholding, based on a globally applied threshold determining the extent of noise in the coverage function. Empirically, 10% of the required `minCov` at high-confidence substitutions worked well in practice on all tested datasets (e.g. a value of 1 in this example where `minCov = 10`. Alternatively, 10% of the mode of the coverage distribution at high-confidence substitutions can represent a valuable choice.

2. Local thresholding, based on a global estimation of background levels via a Gaussian mixture model. Omitting the `threshold` parameter in the call to `getClusters` enables local thresholding, e.g.

```
> clusters <- getClusters( highConfSub = highConfSub,
+                          coverage = coverage,
+                          sortedBam = Bam,
+                          method = "mrn",
+                          cores = 1 )
> clusters
```

Once clusters are identified, the reported genomic regions can be merged in a strand-specific manner and statistics for each resulting cluster, which we call a wavCluster, can be computed using the `filterClusters` function, which takes as input the following elements:

- The identified clusters
- The high-confidence substitution sites
- The genome-wide coverage
- The mixture model
- A *BSgenome* [6] object containing the correct reference sequence
- The reference base expected to be converted by the experimental procedure
- The minimum required width of a wavCluster

The function can be called as follows:

```
> require(BSgenome.Hsapiens.UCSC.hg19)
> wavclusters <- filterClusters( clusters = clusters,
+                    highConfSub = highConfSub,
+                    coverage = coverage,
+                    model = model,
+                    genome = Hsapiens,
+                    refBase = "T",
+                    minWidth = 12)

> wavclusters

GRanges object with 40 ranges and 7 metadata columns:
       seqnames               ranges strand   | Ntransitions    MeanCov
          <Rle>            <IRanges>  <Rle>   |    <integer>  <numeric>
   [1]     chrX [24002044, 24002057]      -   |            1   9.642857
```

```
 [2]     chrX [24002319, 24002348]      -   |            3  10.166667
 [3]     chrX [24002668, 24002683]      -   |            2  14.937500
 [4]     chrX [24002710, 24002726]      -   |            1 154.117647
 [5]     chrX [24002738, 24002761]      -   |            2  29.416667
 ...      ...                   ...    ... ...          ...       ...
[36]     chrX [24005918, 24005941]      -   |            1   14.33333
[37]     chrX [24005949, 24005971]      -   |            2   16.26087
[38]     chrX [24006168, 24006191]      -   |            2   11.12500
[39]     chrX [24006533, 24006554]      -   |            1   31.68182
[40]     chrX [24007059, 24007083]      -   |            3   16.36000
     NbasesInRef CrossLinkEff                       Sequence SumLogOdds
       <integer>    <numeric>                       <factor>  <numeric>
 [1]           4         0.25                  CTAGGATTATTTGA   2.979765
 [2]           9         0.33 GTGTAATATTGAAGTTATACGGTGTACTGA   8.541695
 [3]           6         0.33               CTTTAAATTATGAATT   5.514298
 [4]           7         0.14              GATAGCTTATAAACTGA   2.898055
 [5]          11         0.18       ATATTATAAACTGAAATGTTATGA   5.639963
 ...         ...          ...                             ...       ...
[36]           7         0.14       CAATGTTAGACCAATGGCTTTGAT   3.010309
[37]           3         0.67        CTGGTGAGGTTTTTCTTTATATG   5.607608
[38]           7         0.29       GTGAGGATGGAATCGCTGTAATGA   5.511650
[39]           6         0.17         GGAGGTGGAAGATGAGGTGATT   2.902204
[40]           8         0.38      TGCTGGTGAACATTCTGAAAGTAAT   8.298195
     RelLogOdds
      <numeric>
 [1]  0.7449412
 [2]  0.9490773
 [3]  0.9190497
 [4]  0.4140079
 [5]  0.5127239
 ...        ...
[36]  0.4300442
[37]  1.8692025
[38]  0.7873786
[39]  0.4837006
[40]  1.0372744
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The call returns a `GRanges` object where for each wavCluster:

- the number of high-confidence transitions (`Ntransitions`)
- the the mean coverage (`MeanCov`)
- the number of bases in the reference genome of the same type as the specified `refBase` (`NbasesInRef`)
- the estimated cross-linking efficiency (`CrossLinkEff`), i.e. the ratio between `Ntransitions` and `NbasesInRef`
- the genomic sequence (`Sequence`)

- the sum of the log odds (`SumLogOdds`), contributed by each high-confidence transition within the cluster
- the relative log odds (`RelLogOdds`), i.e. the ratio between `SumLogOdds` and `Ntransitions`

is returned. Notice that the relative log odds can be used to rank clusters according to statistical confidence.

4 Output post-processing

wavClusteR 2.0 contains a variety of functions (summarized in Table 1) that help the user in the post-processing of the identified wavClusters.

Task	**Function**	**Output format**
Export all identified substitutions or high-confidence substitutions	`exportHighConfSub`	BED (for UCSC [7])
Export clusters	`exportClusters`	BED (for UCSC [7])
Export coverage function	`exportCoverage`	BigWig (for UCSC [7])
Visualize the size distribution of wavClusters	`plotSizeDistribution`	histogram
Annotate clusters with respect to genomic features (e.g. CDS, introns, 3'-UTRs, 5'-UTRs) in a strand-specific manner based on the *GenomicFeatures* package	`annotateClusters`	dot chart, vector
Compute metagene profiles of wavClusters, where the density of wavClusters is represented as a function of a reference genomic coordinates	`getMetaGene`	line plot, vector
Compute metaTSS profiles based on all aligned reads in the input BAM file	`getMetaTSS`	line plot, vector
Visualize wavClusteR statistics and meta data to learn pairwise relationships between variables	`plotStatistics`	pairs plot

Table 1: Summary of post-processing functions in *wavClusteR* 2.0.

4.1 Exporting substitutions, wavClusters and coverage function

The identified high-confidence substitutions can be exported as

```
> exportHighConfSub( highConfSub = highConfSub,
+                    filename = "hcTC.bed",
+                    trackname = "hcTC",
+                    description = "hcTC" )
```

where `trackname` and `description` correspond to the very same attributes in the UCSC BED file format specification and define the name of the BED track and its description, respectively. Notice that by replacing `highConfSub` with another set of substitutions (e.g. all identified substitutions of a given type), those can be exported and visualized using the same function call. Similarly, wavClusters can be exported as

```
> exportClusters( clusters = wavclusters,
+                 filename = "wavClusters.bed",
+                 trackname = "wavClusters",
+                 description = "wavClusters" )
```

and the coverage function can be exported as

```
> exportCoverage( coverage = coverage, filename = "coverage.bigWig" )
```

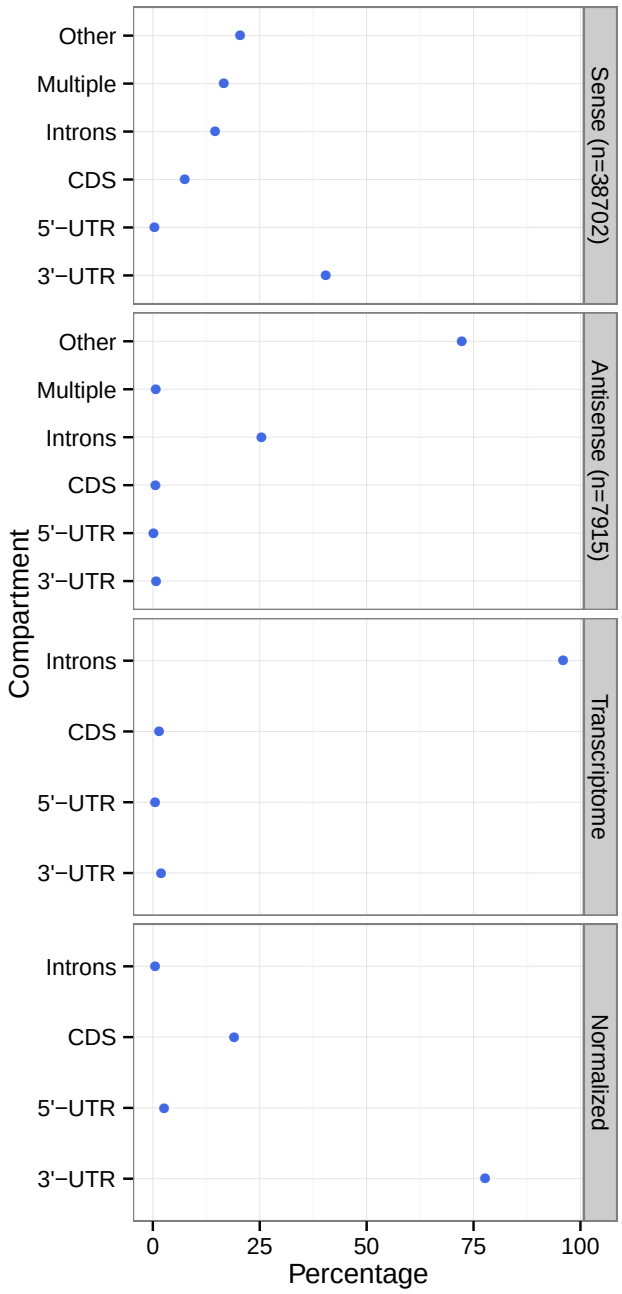
4.2 wavClusters annotation

The identified wavClusters can be annotated with respect to known genomic features using the `annotateClusters` function, which generates a strand-specific dot chart representing wavClusters annotation. The function takes as an input the wavClusters and a `transcriptDB` object containing all transcript annotations. The latter can either be generated a priori using the `makeTranscriptDbFromUCSC` (from the *GenomicFeatures* package) or is automatically fetch and built by `annotateClusters` if not provided. If multiple calls to `annotateClusters` are planned, the recommended solution is to build the object once as

```
> txDB <- makeTranscriptDbFromUCSC(genome = "hg19", tablename = "ensGene")
```

Then, the `annotateClusters` can be called as follows

```
> annotateClusters( clusters = wavclusters,
+               txDB = txDB,
+               plot = TRUE,
+               verbose = TRUE)
```



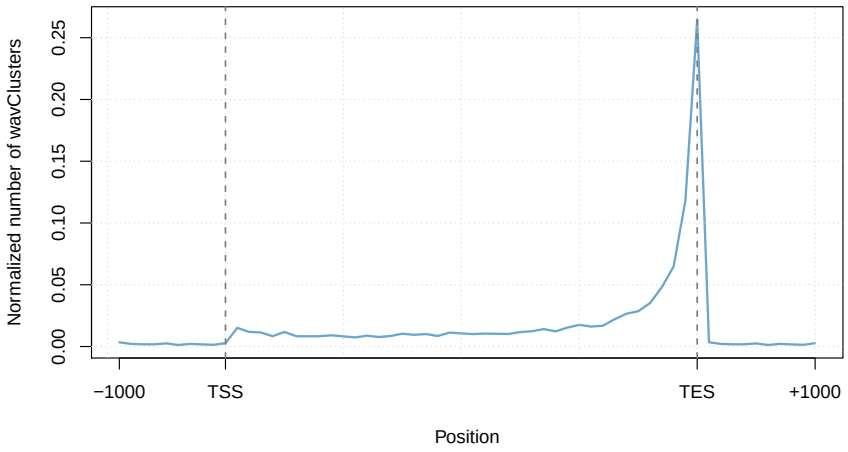
Four dot charts are returned by the function. The first plot (top) represents the percentage of clusters mapping to different transcript features localized on the same strand as the identified clusters. Please note that the dot chart above was produced by providing wavClusters identified on the entire AGO2 dataset. Multiple hits, i.e. wavClusters that overlap with more than one genomic feature, are reported as "multiple", whereas wavClusters that map outside of the considered features are labeled as "other". The latter are then annotated with respect to features on the antisense strand and the results are represented in the second plot. The third plot represents

the relative sequence length of different compartments relative to the total transcriptome length of the organism being considered (clearly, this plot does not depend on the PAR-CLIP data). These ratios are then used to normalize the counts on the sense strand in order to produce the fourth (bottom) plot, which can be used to show enrichments or depletion of clusters in the different functional compartments.

4.3 Computing metagene profiles

A metagene profile, namely a graphical representation of the density of wavClusters as a function of a binning of genomic coordinates across all annotated genes, can be obtained with a call to the `getMetaGene` function as follows:

```
> getMetaGene( clusters = wavclusters,
+              txDB = txDB,
+              upstream = 1e3,
+              downstream = 1e3,
+              nBins = 40,
+              nBinsUD = 10,
+              minLength = 1,
+              plot = TRUE,
+              verbose = TRUE )
```



Please note that the line plot above was produced by providing wavClusters identified on the entire AGO2 dataset. In the function call above, genes were divided in 40 bins (`nBins`) and an upstream/downstream region spanning 1kb was considered (width controlled by `upstream` and `downstream` parameters). This, in turn, was subdivided in 10 bins (`nBinsUD`). No restriction on gene length was applied (`minLength`). The numeric vector of length `nBins` + 2`nBinsUD` with normalized counts is returned by the function and therefore can be used, for example, to compare the distribution of wavClusters across several PAR-CLIP samples.
In addition to metagene profiles, metaTSS profiles based on all aligned reads in the input BAM file can be generated using the `getMetaTSS` function. A default function call is as follows:

```
> getMetaTSS( sortedBam = Bam,
+             txDB = txDB,
```

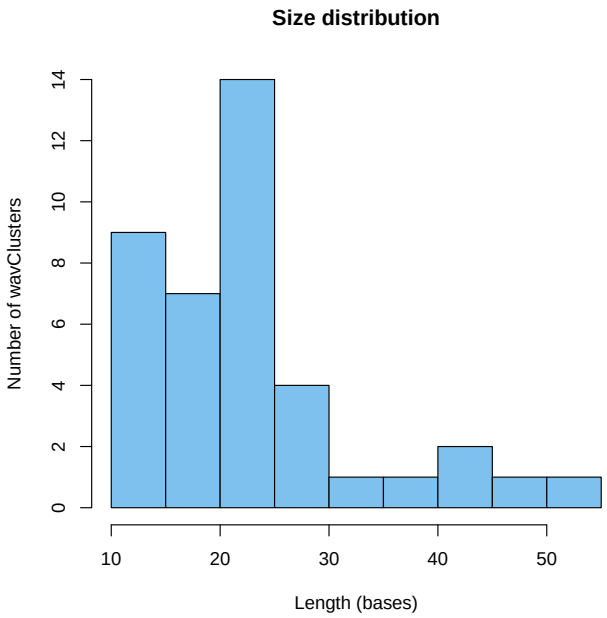
```
+             upstream = 1e3,
+             downstream = 1e3,
+             nBins = 40,
+             unique = FALSE,
+             plot = TRUE,
+             verbose = TRUE )
```

where the `upstream` and `downstream` parameters control the width of the window centered on the transcription start site (TSS) to be considered, `nBins` determines the resolution of the profile. If `unique` is enabled, then overlapping TSSs are discarded. The numeric vector of length `nBins` with normalized read counts is returned by the function and therefore can be used, for example, to compare several PAR-CLIP samples.

4.4 Visualizing the size distribution of wavClusters

The size distribution of wavClusters is visualized as a histogram and returned by the following function call

```
> plotSizeDistribution( clusters = wavclusters, col = "skyblue2" )
```



where additional parameters of the `hist` function can be passed in the function call. Finally, if `showCov=TRUE`, a scatter plot of average cluster coverage vs. cluster length is returned

```
> plotSizeDistribution( clusters = wavclusters, showCov = TRUE, col = "skyblue2" )
```

5 Session Info

```
> sessionInfo()
```

```
R version 3.2.0 (2015-04-16)
Platform: x86_64-unknown-linux-gnu (64-bit)
Running under: Ubuntu 14.04.2 LTS

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

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

other attached packages:
 [1] BSgenome.Hsapiens.UCSC.hg19_1.4.0 BSgenome_1.36.0
 [3] rtracklayer_1.28.0                wavClusteR_2.2.0
 [5] Rsamtools_1.20.0                  Biostrings_2.36.0
 [7] XVector_0.8.0                     GenomicRanges_1.20.0
 [9] GenomeInfoDb_1.4.0                IRanges_2.2.0
[11] S4Vectors_0.6.0                   BiocGenerics_0.14.0

loaded via a namespace (and not attached):
 [1] Rcpp_0.11.5             wmtsa_2.0-0             compiler_3.2.0
 [4] RColorBrewer_1.1-2      futile.logger_1.4       plyr_1.8.1
 [7] GenomicFeatures_1.20.0  bitops_1.0-6            futile.options_1.0.0
[10] iterators_1.0.7         tools_3.2.0             zlibbioc_1.14.0
[13] rpart_4.1-9             biomaRt_2.24.0          digest_0.6.8
[16] mclust_5.0.0            lattice_0.20-31         RSQLite_1.0.0
[19] gtable_0.1.2            foreach_1.4.2           DBI_0.3.1
[22] proto_0.3-10            cluster_2.0.1           stringr_0.6.2
[25] ifultools_2.0-1         ade4_1.7-2              nnet_7.3-9
[28] grid_3.2.0              Biobase_2.28.0          AnnotationDbi_1.30.0
[31] survival_2.38-1         XML_3.98-1.1            BiocParallel_1.2.0
[34] foreign_0.8-63          latticeExtra_0.6-26     Formula_1.2-1
[37] seqinr_3.1-3            ggplot2_1.0.1           reshape2_1.4.1
[40] lambda.r_1.1.7          splus2R_1.2-0           splines_3.2.0
[43] Hmisc_3.15-0            scales_0.2.4            codetools_0.2-11
[46] GenomicAlignments_1.4.0 MASS_7.3-40             colorspace_1.2-6
[49] labeling_0.3            acepack_1.3-3.3         RCurl_1.95-4.5
[52] munsell_0.4.2
```

References

[1] Sievers, C., Schlumpf, T., Sawarkar, R., Comoglio, F. & Paro, R. (2012) Mixture models and wavelet transforms reveal high confidence RNA-protein interaction sites in MOV10 PAR-CLIP

data. *Nucleic Acids Res* **40(2)**: e160

[2] Comoglio, F., Sievers, C. & Paro, R. (2015) Sensitive and highly resolved identification of RNA-protein interaction sites in PAR-CLIP data. BMC Bioinformatics, 16, 32

[3] Langmead,B., Trapnell,C., Pop,M. & Salzberg,S.L. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, **10**, R25

[4] Kishore, S. et al. (2011) A quantitative analysis of CLIP methods for identifying binding sites of RNA-binding proteins. *Nature Methods*, **8(7)**, 559-564

[5] Morgan, M. & Pages, H. Rsamtools: Binary alignment (BAM), variant call (BCF), or tabix file import, http://bioconductor.org/packages/release/bioc/html/Rsamtools. html

[6] Pages, H., BSgenome: Infrastructure for Biostrings-based genome data packages

[7] Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D. (2002) The human genome browser at UCSC. *Genome Res.* **12(6)**, 996-1006.