How to use breakpointR

David Porubsky*

*david.porubsky@gmail.com

June 3, 2021

Contents

1 Introduction

BreakpointR is a novel algorithm designed to accurately tracks template strand changes in Strand-seq data using a bi-directional read-based binning. Read-based binning strategy scales each bin size dynamically to accommodate defined number of reads what accounts for mappability bias in sparsely covered single-cell Strand-seq data. In such dynamically scaled bins, read directionality is tracked in order to search for points where template-strand-state changes. BreakpointR takes as an input reads aligned to the reference genome and stored in a single BAM file per single cell. BreakpointR outputs locations where directionality of sequenced template strands changes. Template strands changes are defined by changes in proportion of reads mapped to positive ('Crick') and negative ('Watson') strand of the reference genome. In a diploid organism such as human we distinguish three possible scenarios of template strand inheritence. If both parental homologues were inherited as Watson template (we assign a WW state), if only Crick templates were inherited (we assign CC state), or one Watson and one Crick template was inherited by each parent (we assign WC state).

2 Quickstart

The main function of this package is called `breakpointr()` and performs all the necessary steps to get from aligned reads in BAMs to predicted breakpoints (changes) in strand directionality. For an unexperienced user we advise to run *breakpointR* with default parameters and later based on the obtained results start to tweak certain parameters. For more detailed guidance on how to tweak certain parameters see section 3.

```
library(breakpointR)
## Run breakpointR with a default parameters
breakpointr(inputfolder='folder-with-BAMs', outputfolder='output-folder')
```

Although in most cases the one of the above commands will produce reasonably good results, it might be worthwhile to adjust the default parameters in order to improve performance and the quality of the results. You can get a description of all available parameters by typing

```
?breakpointr
```

After the function has finished, you will find the folder **output-directory** containing all produced files and plots. This folder contains the following **files** and **folders**:

- *breakpointR.config*: This file contains all parameters that are necessary to reproduce your analysis. You can specify this file as shown below in order to run another analysis with the same parameter settings.

```
breakpointr(..., configfile='breakpointR.config')
```

- **breakpoints** UCSC browser formatted bedgraphs compile all breakpoints across all single-cell libraries. This folder also contains list of all localized breakpoints in all single-cell libraries. Lastly, locations of breakpoint hotspots are reported here if

```
callHotSpots=TRUE
```

- **browserfiles** UCSC browser formatted files with exported reads, deltaWs and breakpoints for every single-cell library.

- **data** Contains RData files that store complete results of breakpointR analysis for each single-cell library.
- **plots**: Genome-wide plots for selected chromosomes, genome-wide heatmap of strand states as well as chromosome specific read distribution together with localized breakpoints. All aforementioned plots are created by default.

2.1 Running breakpointR

The function `breakpointr()` takes as an input BAM files stored in the inputfolder and produces an outputfolder with results, plots and browserfiles. The following code is an example of how to run *breakpointR* on single-end reads with a 'windowsize' defined by size of 1Mb (see subsection 3.3). Results will be stored in **outputfolder/data** as RData objects. Such data can be later loaded for further processing and customized plotting.

```
library(breakpointR)

## Get some example files
datafolder <- system.file("extdata", "example_bams", package="breakpointRdata")
outputfolder <- tempdir()
## Run breakpointR
breakpointr(inputfolder = datafolder, outputfolder = outputfolder,
            chromosomes = 'chr22', pairedEndReads = FALSE,
            reuse.existing.files = FALSE, windowsize = 1000000,
            binMethod = 'size', pair2frgm = FALSE, min.mapq = 10,
            filtAlt = TRUE)
```

3 Recommended settings

3.1 Reading BAM files

Currently *breakpointR* can take as an input only aligned reads stored in BAM files. All BAM files are expected to be present in a folder specified as `breakpointr(..., inputfolder)`. We advise to remove reads with low mapping quality and reads with alternative alignments. Duplicated reads are removed by default.

```
breakpointr(..., min.mapq = 10, filtAlt = TRUE)
```

3.2 Removing certain regions

breakpointR allows a user to exclude certain genomic regions from the analysis. This comes handy when one wants to remove reads that falls into low complexity regions such as segmental duplications or centromeres. Such low complexity regions might cause false positive breakpoints due to the spurious mappings of short reads. To mask certain genomic regions user has to define path to a bed formatted text file as `breakpointr(..., maskRegions)`. All reads falling into these regions will be discarded prior to breakpoint detection. User defined regions to mask can be downloaded from the UCSC Table Browser.

3.3 Binning strategy

breakpointR uses read based binning strategy and offers two approaches to set the bin size: (1) user defined number of reads in each bin or (2) number of reads in every bin is selected based on desired genomic size of each bin.

```
library(breakpointR)
## Binning strategy based on desired bin length
breakpointr(inputfolder='folder-with-BAM', outputfolder='output-folder',
            windowsize=1e6, binMethod='size')
## Binning strategy based user-defined number of reads in each bin
breakpointr(inputfolder='folder-with-BAM', outputfolder='output-folder',
            windowsize=100, binMethod='reads')
```

The sensitivity and specificity of breakpoint detection depend on user defined bin size. We recommend to select rather large bin size (>=1Mb) in order to reliably detect low frequency sister chromatid exchange (SCE) events. In order to detect smaller events like inversions smaller bin size is recommended. Keep in mind that such settings also leads to a higher level of false positive breakpoints. In this case one might need to tweak other breakpoint detection parameters (see subsection 3.4).

3.4 Breakpoint peak detection

Breakpoint detection is based on finding significant peaks in deltaW values. Level of peak significance is measured in the number of standard deviations (SD) from the set threshold (z-score) `breakpointr(..., peakTh)`. By default the threshold is set to the 1/3 of the highest detlaW value. For the data with noisy and uneven coverage we recommend to set higher threshold, for example 1/2 of the highest deltaW value. In addition, we also recommend to tweak 'trim' option `breakpointr(..., trim)` in order to set a fraction of extreme deltaW values to be excluded from SD calculation.

```
## Example deltaW values
exampleFolder <- system.file("extdata", "example_results",
                             package="breakpointRdata")
exampleFile <- list.files(exampleFolder, full.names=TRUE)[1]
breakpoint.object <- loadFromFiles(exampleFile)
head(breakpoint.object[[1]]$deltas)
```

```
## GRanges object with 6 ranges and 1 metadata column:
##       seqnames      ranges strand |    deltaW
##          <Rle>   <IRanges>  <Rle> | <numeric>
##   [1]     chr1   7560-7594      - |        17
##   [2]     chr1   8569-8612      + |         0
##   [3]     chr1 15116-15143      - |        57
##   [4]     chr1 17235-17240      - |       130
##   [5]     chr1 19615-19720      - |        41
##   [6]     chr1 19849-19911      - |        38
##   -------
##   seqinfo: 23 sequences from an unspecified genome
```

3.5 Background reads

Background reads are a common feature of Strand-seq libraries. Strand-seq is based on removal of newly synthesized strand during DNA replication, however this process is not perfect. Therefore, we usually expect low abundance reads aligned in a opposite direction even for purely WW or CC chromosomes. An another reason to see such artifacs is imperfect read mapping especially in repetitive and complex genomic regions. To remove reads falling into such regions see subsection 3.2.

3.6 Calling breakpoint hotspots

In order to find locations where breakpoints occur around the same genomic position in multiple Strand-seq libraries there is `hotspotter()`. Function can be invoked by setting corresponding parameter to 'TRUE'. It make sense to set this parameter only if there is available a reasonable number (>=50) of Strand-seq libraries.

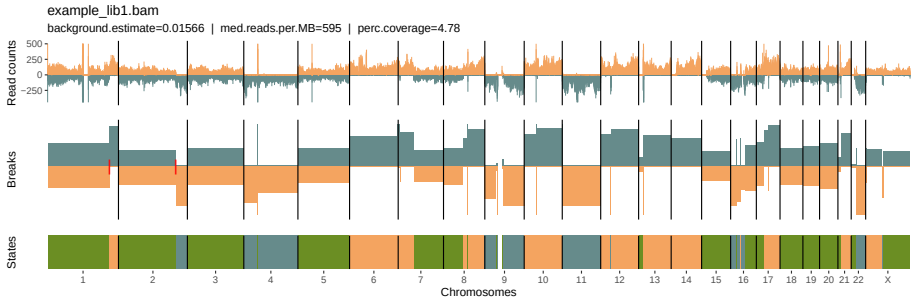
```
## To run breakpoint hotspot analysis using the main breakpointR function
breakpointr(..., callHotSpots=TRUE)
```

```
## To run breakpoint hotspot analysis using exported data
exampleFolder <- system.file("extdata", "example_results",
                             package="breakpointRdata")
exampleFiles <- list.files(exampleFolder, full.names=TRUE)
breakpoint.objects <- loadFromFiles(exampleFiles)
## Extract breakpoint coordinates
breaks <- lapply(breakpoint.objects, '[[', 'breaks')
## Get hotspot coordinates
hotspots <- hotspotter(breaks, bw=1e6)
```

3.7 Loading results and plotting single cells

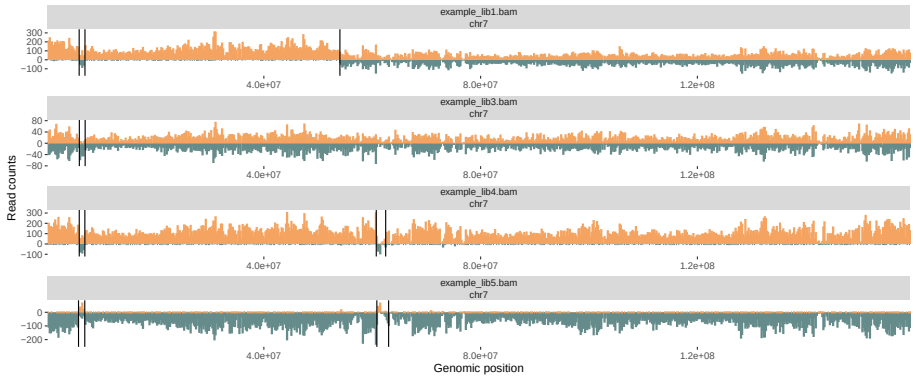
```
## Plotting a single library
exampleFolder <- system.file("extdata", "example_results",
                             package="breakpointRdata")
exampleFile <- list.files(exampleFolder, full.names=TRUE)[1]
plotBreakpoints(exampleFile)
```

```
## [[1]]
```




```
## Plotting a single library
exampleFolder <- system.file("extdata", "example_results",
                             package="breakpointRdata")
exampleFiles <- list.files(exampleFolder, full.names=TRUE)[1:4]
plotBreakpointsPerChr(exampleFiles, chromosomes = 'chr7')
```

```
## $chr7
```



4 Session Info

```
toLatex(sessionInfo())
```

- R version 4.1.0 (2021-05-18), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server x64 (build 17763)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.39.0, GenomeInfoDb 1.29.0, GenomicRanges 1.45.0, IRanges 2.27.0, S4Vectors 0.31.0, breakpointR 1.11.0, breakpointRdata 1.11.0, cowplot 1.1.1, knitr 1.33
- Loaded via a namespace (and not attached): Biobase 2.53.0, BiocManager 1.30.15, BiocParallel 1.27.0, BiocStyle 2.21.1, Biostrings 2.61.0, DBI 1.1.1, DelayedArray 0.19.0, GenomeInfoDbData 1.2.6, GenomicAlignments 1.29.0, Matrix 1.3-4, MatrixGenerics 1.5.0, R6 2.5.0, RCurl 1.98-1.3, Rsamtools 2.9.0, SummarizedExperiment 1.23.0, XVector 0.33.0, assertthat 0.2.1, bitops 1.0-7, codetools 0.2-18, colorspace 2.0-1, compiler 4.1.0, crayon 1.4.1, digest 0.6.27, doParallel 1.0.16, dplyr 1.0.6, ellipsis 0.3.2, evaluate 0.14, fansi 0.5.0, farver 2.1.0, foreach 1.5.1, generics 0.1.0, ggplot2 3.3.3, glue 1.4.2, grid 4.1.0, gtable 0.3.0, gtools 3.8.2, highr 0.9, htmltools 0.5.1.1, iterators 1.0.13, labeling 0.4.2, lattice 0.20-44, lifecycle 1.0.0, magrittr 2.0.1, matrixStats 0.59.0, munsell 0.5.0, pillar 1.6.1, pkgconfig 2.0.3, purrr 0.3.4, rlang 0.4.11, rmarkdown 2.8, scales 1.1.1, stringi 1.6.2, stringr 1.4.0, tibble 3.1.2, tidyselect 1.1.1, tools 4.1.0, utf8 1.2.1, vctrs 0.3.8, xfun 0.23, yaml 2.2.1, zlibbioc 1.39.0