# Introduction to GeneBreak

Evert van den Broek*
& Stef van Lieshout

July 16, 2015

Department of Pathology
VU University Medical Center
The Netherlands, Amsterdam

## Contents



*email@email.com

# 1 Running GeneBreak

This is a short tutorial on how to use the `GeneBreak` package. It describes an example workflow which uses included copy number aberration (CNA) data obtained by CGHcall analysis of 200 array-CGH (Agilent 180k) samples from advanced colorectal cancers. Let's start by loading the package.

```
> library(GeneBreak)
```

## 1.1 Detect breakpoints from copy-number data

Copy number data can be loaded in two ways. We recommend the usage of bioconductor packages `CGHcall` or `QDNAseq` to process CNA data from array-CGH or sequencing data respectively. The obtained cghCall/QDNAseq object can directly serve as input for the GeneBreak pipeline. Alternatively, a data.frame with exactly these five columns: "Chromosome", "Start", "End" and "FeatureName" (usually probe or bin identifier) followed by columns with sample data can be provided. Note: the first five column names of the data.frame must be similar! In this tutorial we will use a built-in dataset that only contains CNA data from chromosome 20:

### 1.1.1 Loading cghCall object

```
> data( "copynumber.data.chr20" )
```

Inspection of the loaded data shows that we are dealing with an R object of class `cghCall` with 3653 features (array-CGH probes in this case) and 200 samples.

```
> copynumber.data.chr20
```

```
cghCall (storageMode: lockedEnvironment)
assayData: 3653 features, 200 samples
  element names: calls, copynumber, probamp, probgain, probloss, probnorm, segmented
protocolData: none
phenoData
  sampleNames: sample_1 sample_2
    ... sample_200 (200 total)
  varLabels: Cellularity
  varMetadata: labelDescription
featureData
  featureNames: A_16_P03469195
    A_14_P136138 ... A_18_P13856091
    (3653 total)
  fvarLabels: Chromosome Start End
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

To generate an object of class `CopyNumberBreakpoints` with breakpoint locations, run `getBreakpoints()`. This will obtain the required information from the `cghCall` object and determine the breakpoint locations.

```
> breakpoints <- getBreakpoints( data = copynumber.data.chr20 )

Breakpoint detection started...
```

#### 1.1.2 Loading data from a dataframe

There is a possibilty of using a `data.frame()` as input for `GeneBreak`. This allows breakpoint analysis of data from any copy number detection pipeline by importing a text file into `getBreakpoints()`.

Here we show how to use two data.frames() with segment and (optionally) call values as input for `getBreakpoints` instead of a cghCall/QDNAseq object.

```
> library(CGHcall)
> cgh <- copynumber.data.chr20
> segmented <- data.frame( Chromosome=chromosomes(cgh), Start=bpstart(cgh),
+  End=bpend(cgh), FeatureName=featureNames(cgh), segmented(cgh))
> called <- data.frame( Chromosome=chromosomes(cgh), Start=bpstart(cgh),
+  End=bpend(cgh), FeatureName=featureNames(cgh), calls(cgh))
> breakpoints <- getBreakpoints( data = segmented, data2 = called )
```

### 1.2 Breakpoint selection by filtering

Next breakpoints can be filtered by stringent criteria. Different filters can be set (see `?bpFilter` for more details). Default setting is "CNA-ass" which means that breakpoints flanked by copy number neutral segments will be filtered out. Note: you need discrete copy number calls (loss,neutral, gain, etc) for this option.

```
> breakpointsFiltered <- bpFilter( breakpoints, filter = "CNA-ass" )

Applying BP selection...
```

### 1.3 Identification of genes affected by breakpoints

This describes the two steps needed to identify genes affected by breakpoints by the `GeneBreak` package.

#### 1.3.1 Loading gene annotation data

We need to load gene annotations to be able to identify genes affected by breakpoints in the next step. Gene annotation for human reference genome hg18 (and hg19, hg38) are built-in, but also user-defined annotations can be used. The required columns for this data.frame are "Gene", "Chromosome", "Start" and "End".

```
> data( "ens.gene.ann.hg18" )
```

This shows the contents of the hg18 gene annotation dataframe:

```
> head( ens.gene.ann.hg18 )
      Gene           EnsID Chromosome  Start    End   band strand
MIRN1302-2 ENSG00000221311          1  20229  20366 p36.33      1
   FAM138E ENSG00000222027          1  24417  25944 p36.33     -1
   FAM138E ENSG00000222003          1  24417  25944 p36.33     -1
   FAM138A ENSG00000222003          1  24417  25944 p36.33     -1
     OR4F5 ENSG00000177693          1  58954  59871 p36.33      1
    OR4F29 ENSG00000177799          1 357522 358460 p36.33      1
```

### 1.3.2 Detection of gene-associated breakpoints

Here, the loaded gene annotation information will be added to the GeneBreak object and feature-to-gene mapping will be performed.

```
> breakpointsAnnotated <- addGeneAnnotation( breakpointsFiltered, ens.gene.ann.hg18 )

Adding of gene annotation started on 659 genes by 200 samples
0% ... 25% ... 50% ... 75% ... Adding gene annotation DONE
```

Added labels describe gene position with respect to feature positions: A: genes located upstream of the first chromosomal feature (no gene-associated features) B: genes located downstream of the last chromosomal feature (no gene-associated features) C: the whole gene is located between two features (in case of array-CGH probes) C: the whole gene is located between start and end of one bin (in case of sequencing data) D: gene represented by one or multiple features E: gene represented by one or multiple features, but the end of the gene is not covered by any feature X: no feature covers the chromosome of the gene

Next, gene-associated breakpoints will be identified.

```
> breakpointGenes <- bpGenes( breakpointsAnnotated )

Running bpGenes: 659 genes and 200 samples
0% ... 25% ... 50% ... 75% ... bpGenes DONE
A total of 1029 gene breaks in 241 genes detected
```

## 1.4 Cohort-based breakpoint statistics

Following identification of (gene) breakpoints per profile, breakpoint events that are significantly recurring will be determined by dedicated statistical analysis. This can be performed at "gene" (breakpoint gene) and/or "feature" (breakpoint location) level. Two different methods of FDR-type correction for multiple testing can be used, the standard Benjamini-Hochberg FDR-type correction ("BH") or dedicated Benjamini-Hochberg FDR-type correction ("Gilbert").

### 1.4.1 Detection of recurrent breakpoint genes

The gene-based statistical analysis includes correction for covariates that may influence the probability to be a breakpoint gene including number of breakpoints in a profile, number of gene-associated features and gene length by gene-associated feature coverage. Multiple testing can be applied by the powerful dedicated Benjamini-Hochberg FDR-type correction ("Gilbert") that accounts for the discreteness of the null-distribution. NOTE: when running `bpStats()` warnings can be generated by a function (glm.fit) of a dependancy package, this does not harm the analysis.

```
> breakpointStatistics <- bpStats( breakpointGenes, level = "gene", method = "Gilbert" )

Applying statistical test over 200 samples for: gene breakpoints: Gilbert test...
```

This will return an object of class `CopyNumberBreakPointGenes`.
By using `recurrentGenes()` we can observe the recurrent affected genes.

```
> head( recurrentGenes( breakpointStatistics ) )

 A total of 19 recurrent breakpoint genes (at FDR < 0.1)
         Gene sampleCount featureTotal
13886   PCMTD2          64           4
13898 C20orf69          33           3
4268     BFSP1           8           5
5473    ABHD12          10           9
4780  C20orf26           7          18
4102    KIF16B           7          19
             pvalue           FDR
13886 1.350385e-103 1.848343e-101
13898  5.522293e-44  3.860197e-42
4268   3.941447e-07  3.148759e-05
5473   5.756361e-05  3.687639e-03
4780   2.748743e-04  1.204846e-02
4102   4.054266e-04  1.322722e-02
```

### 1.4.2 Detection of recurrent breakpoint locations

With this step, statistics at breakpoint location (feature) level will be added to the object of class `CopyNumberBreakPointGenes`. Here, we recommend to use the less computationally intensive standard Benjamini-Hochberg FDR-type correction for multiple testing, because the breakpoint probability is equal across features per profile, which means that all positions correspond to the same null-distribution.

```
> breakpointStatistics <- bpStats(
+   breakpointStatistics, level = "feature", method = "BH" )
```

```
Applying statistical test over 200 samples for feature breakpoints: BH test...
```

```
> breakpointStatistics
```

```
 --- Object Info ---
 This is an object of class "CopyNumberBreakPointGenes"
 3653 features by 200 samples
 A total of 985 breakpoints
 A total of 1029 gene breaks in 241 genes
 A total of 19 recurrent breakpoint genes (FDR < 0.1)
 A total of 29 recurrent breakpoints (FDR < 0.1)
 See accessOptions(object) for how to access data in this object
```

## 1.5 Visualization of breakpoint frequencies

Breakpoint locations and frequencies can be visualized using this function:

```
> bpPlot( breakpointStatistics, fdr.threshold = 0.1 )
```

```
Plotting breakpoint frequencies ...
Plotting Chromosome: 20
```

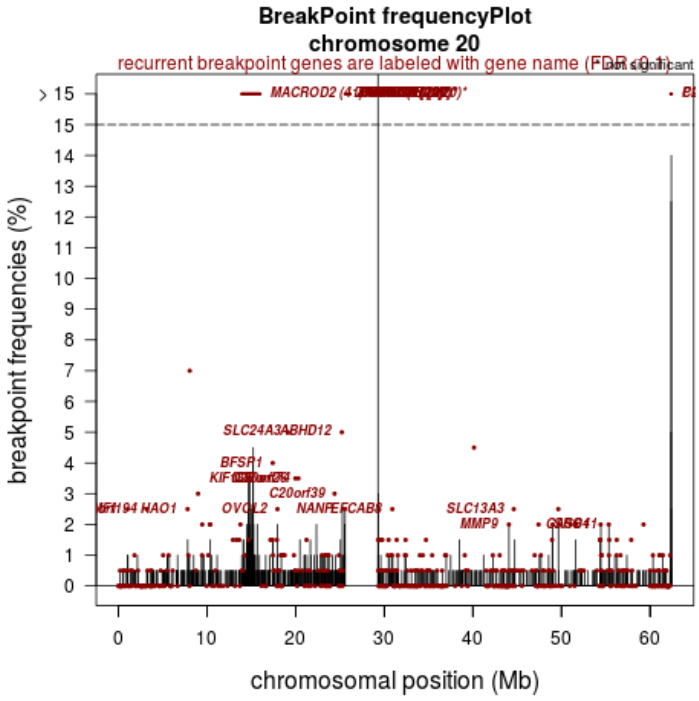


Figure 1: Graphical representation of CNA-associated chromosomal breakpoint frequencies and their distribution over chromosomes 20. The X-axis depicts the genomic position in Mb. The Y-axis depicts the chromosomal breakpoint frequencies across the series of 200 CRC samples. Breakpoint frequencies are indicated on array-CGH probe-level (vertical black bars) and on gene-level (horizontal red bars). Recurrent breakpoint genes (FDR<0.1) are named. When the gene breakpoint frequency exceeded 15% (horizontal dashed line), the breakpoint frequency (%) follows the gene name.

## 2 Storage of R objects

At any time during the analysis, the GeneBreak objects (and any R objects for that matter) can be saved to disk with: `saveRDS`, and in the future be read from the local file with `loadRDS`

## 3 Downloading Gene Annotations

This section describes the steps taken to create the gene annotations used in this package. It may serve as a start for creating your own if required for whatever reason.

```
> # gene annotations obtained via Biomart.
> # HUGO gene names (HGNC symbol), Ensembl_ID and chromosomal location
>
> # Used (and most) recent releases:
> # HG18: release54
> # HG19: release75
> # HG38: release80 (date: 150629)
>
> library(biomaRt)
> ensembl54 = useMart(
+   host = 'may2009.archive.ensembl.org',
+   biomart = 'ENSEMBL_MART_ENSEMBL',
+   dataset = "hsapiens_gene_ensembl"
+ )
> ensembl75 = useMart(
+   host = 'feb2014.archive.ensembl.org',
+   biomart = 'ENSEMBL_MART_ENSEMBL',
+   dataset = "hsapiens_gene_ensembl"
+ )
> ensembl80 = useMart(
+   "ensembl",
+   dataset = "hsapiens_gene_ensembl"
+ )
> createAnnotationFile <- function( biomartVersion ) {
+   biomart_result <- getBM(
+     attributes =  c(
+       "hgnc_symbol", "ensembl_gene_id", "chromosome_name",
+       "start_position", "end_position", "band", "strand"
+     ),
+     mart = biomartVersion
+   )
+
+   biomart_result[ ,3] <- as.vector( biomart_result[ ,3] )
+   idx_x <- biomart_result$chromosome_name == "X"
```

```
+   idx_y <- biomart_result$chromosome_name == "Y"
+   biomart_result$chromosome_name[ idx_x ] <- "23"
+   biomart_result$chromosome_name[ idx_y ] <- "24"
+
+   biomart_genes <- biomart_result[ which(biomart_result[ ,1] != "" &
+     biomart_result[ ,3] %in% c(1:24)) , ]
+   colnames(biomart_genes)[1:5] <- c("Gene","EnsID","Chromosome","Start","End")
+
+   cat(
+     c( "Biomart version:", biomartVersion@host,
+           "including:", dim(biomart_genes)[1], "genes\n"
+     )
+   )
+
+   return( biomart_genes )
+ }
> ens.gene.ann.hg18 <- createAnnotationFile( ensembl54 )
> ens.gene.ann.hg19 <- createAnnotationFile( ensembl75 )
> ens.gene.ann.hg38 <- createAnnotationFile( ensembl80 )
>
```

# 4 Session Information

The version number of R and packages loaded for generating the vignette were:

```
R version 3.2.1 (2015-06-18)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.2 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8
 [2] LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8
 [6] LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8
 [8] LC_NAME=C
 [9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8
[12] LC_IDENTIFICATION=C

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

other attached packages:
[1] CGHbase_1.26.0
[2] marray_1.44.0
[3] limma_3.22.7
[4] Biobase_2.26.0
[5] BiocGenerics_0.12.1
[6] GeneBreak_0.99.0

loaded via a namespace (and not attached):
[1] rversions_1.0.2 tools_3.2.1
[3] curl_0.9.1      Rcpp_0.11.6
[5] memoise_0.2.1   xml2_0.1.1
[7] git2r_0.10.1    digest_0.6.8
[9] devtools_1.8.0
```