Package 'alpine'

April 12, 2022

Title alpine

Version 1.20.0

Author Michael Love, Rafael Irizarry

Maintainer Michael Love <michaelisaiahlove@gmail.com>

Description Fragment sequence bias modeling and correction for RNA-seq transcript abundance estimation.

License GPL (>=2)

VignetteBuilder knitr

Depends R (>= 3.3)

Imports Biostrings, IRanges, GenomicRanges, GenomicAlignments, Rsamtools, SummarizedExperiment, GenomicFeatures, speedglm, splines, graph, RBGL, stringr, stats, methods, graphics, GenomeInfoDb, S4Vectors

Suggests knitr, testthat, markdown, alpineData, rtracklayer, ensembldb, BSgenome.Hsapiens.NCBI.GRCh38, RColorBrewer

biocViews Sequencing, RNASeq, AlternativeSplicing, DifferentialSplicing, GeneExpression, Transcription, Coverage, BatchEffect, Normalization, Visualization, QualityControl

RoxygenNote 5.0.1

git_url https://git.bioconductor.org/packages/alpine

git_branch RELEASE_3_14

git_last_commit 9348ef1

git_last_commit_date 2021-10-26

Date/Publication 2022-04-12

R topics documented:

`alpine-package` *alpine: bias corrected transcript abundance estimation*

Description

alpine is a package for estimating and visualizing many forms of sample-specific biases that can arise in RNA-seq, including fragment length distribution, positional bias on the transcript, read start bias (random hexamer priming), and fragment GC content (amplification). It also offers bias-corrected estimates of transcript abundance (FPKM). It is currently designed for un-stranded paired-end RNA-seq data.

Details

See the package vignette for a detailed workflow.

The main functions in this package are:

1. buildFragtypes - build out features for fragment types from exons of a single gene (GRanges)
2. fitBiasModels - fit parameters for one or more bias models over a set of ~100 medium to highly expressed single isoform genes (GRangesList)
3. estimateAbundance - given a set of genome alignments (BAM files) and a set of isoforms of a gene (GRangesList), estimate the transcript abundances for these isoforms (FPKM) for various bias models
4. extractAlpine - given a list of output from `estimateAbundance`, compile an FPKM matrix across transcripts and samples
5. predictCoverage - given the exons of a single gene (GRanges) predict the coverage for a set of samples given fitted bias parameters and compute the observed coverage

Some helper functions for preparing gene objects:

1. splitGenesAcrossChroms - split apart "genes" where isoforms are on different chromosomes

2. splitLongGenes - split apart "genes" which cover a suspiciously large range, e.g. 1 Mb
3. mergeGenes - merge overlapping isoforms into new "genes"

Some other assorted helper functions:

1. normalizeDESeq - an across-sample normalization for FPKM matrices
2. getFragmentWidths - return a vector estimated fragment lengths given a set of exons for a single gene (GRanges) and a BAM file
3. getReadLength - return the read length of the first read across BAM files

The plotting functions are:

1. plotGC - plot the fragment GC bias curves
2. plotFragLen - plot the framgent length distributions
3. plotRelPos - plot the positional bias (5' to 3')
4. plotOrder0, plotOrder1, plotOrder2 - plot the read start bias terms
5. plotGRL - a simple function for visualizing GRangesList objects

Author(s)

Michael Love

References

Love, M.I., Hogenesch, J.B., and Irizarry, R.A., Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation. Nature Biotechnologyh (2016) doi: 10.1038/nbt.3682

`buildFragtypes` *Build fragment types from exons*

Description

This function constructs a DataFrame of fragment features used for bias modeling, with one row for every potential fragment type that could arise from a transcript. The output of this function is used by fitBiasModels, and this function is used inside estimateAbundance in order to model the bias affecting different fragments across isoforms of a gene.

Usage

```
buildFragtypes(exons, genome, readlength, minsize, maxsize, gc = TRUE,
  gc.str = TRUE, vlmm = TRUE)
```

Arguments

exons	a GRanges object with the exons for a single transcript
genome	a BSgenome object
readlength	the length of the reads. This doesn't necessarily have to be exact (+/- 1 bp is acceptable)
minsize	the minimum fragment length to model. The interval between minsize and maxsize should contain the at least the central 95 percent of the fragment length distribution across samples
maxsize	the maximum fragment length to model
gc	logical, whether to calculate the fragment GC content
gc.str	logical, whether to look for presence of stretches of very high GC within fragments
vlmm	logical, whether to calculate the Cufflinks Variable Length Markov Model (VLMM) for read start bias

Value

a DataFrame with bias features (columns) for all potential fragments (rows)

Examples

```
library(GenomicRanges)
library(BSgenome.Hsapiens.NCBI.GRCh38)
data(preprocessedData)
readlength <- 100
minsize <- 125 # see vignette how to choose
maxsize <- 175 # see vignette how to choose
fragtypes <- buildFragtypes(ebt.fit[["ENST00000624447"]],
                            Hsapiens, readlength,
                            minsize, maxsize)
```

---

estimateAbundance *Estimate bias-corrected transcript abundances (FPKM)*

Description

This function takes the fitted bias parameters from fitBiasModels and uses this information to derive bias corrected estimates of transcript abundance for a gene (with one or more isoforms) across multiple samples.

Usage

```
estimateAbundance(transcripts, bam.files, fitpar, genome, model.names,
  subset = TRUE, niter = 100, lib.sizes = NULL, optim = FALSE,
  custom.features = NULL)
```

Arguments

`transcripts`	a GRangesList of the exons for multiple isoforms of a gene. For a single-isoform gene, just wrap the exons in `GRangesList()`
`bam.files`	a named vector pointing to the indexed BAM files
`fitpar`	the output of fitBiasModels
`genome`	a BSGenome object
`model.names`	a character vector of the bias models to use. These should have already been specified when calling fitBiasModels. Four exceptions are models that use none, one or both of the offsets, and these are called with: `"null"`, `"fraglen"`, `"vlmm"`, or `"fraglen.vlmm"`.
`subset`	logical, whether to downsample the non-observed fragments. Default is TRUE
`niter`	the number of EM iterations. Default is 100.
`lib.sizes`	a named vector of library sizes to use in calculating the FPKM. If NULL (the default) a value of 1e6 is used for all samples.
`optim`	logical, whether to use numerical optimization instead of the EM. Default is FALSE.
`custom.features`	an optional function to add custom features to the fragment types DataFrame. This function takes in a DataFrame returned by buildFragtypes and returns a DataFrame with additional columns added. Default is NULL, adding no custom features.

Value

a list of lists. For each sample, a list with elements: theta, lambda and count.

- **theta** gives the FPKM estimates for the isoforms in `transcripts`
- **lambda** gives the average bias term for the isoforms
- **count** gives the number of fragments which are compatible with any of the isoforms in `transcripts`

References

The model describing how bias estimates are used to estimate bias-corrected abundances is described in the Supplemental Note of the following publication:

Love, M.I., Hogenesch, J.B., and Irizarry, R.A., Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation. Nature Biotechnologyh (2016) doi: 10.1038/nbt.3682

The likelihood formulation and EM algorithm for finding the maximum likelihood estimate for abundances follows this publication:

Salzman, J., Jiang, H., and Wong, W.H., Statistical Modeling of RNA-Seq Data. Statistical Science (2011) doi: 10.1214/10-STS343

Examples

```
# see vignette for a more realistic example

# these next lines just write out a BAM file from R
# typically you would already have a BAM file
library(alpineData)
library(GenomicAlignments)
library(rtracklayer)
gap <- ERR188088()
dir <- system.file(package="alpineData", "extdata")
bam.file <- c("ERR188088" = file.path(dir,"ERR188088.bam"))
export(gap, con=bam.file)

data(preprocessedData)
library(GenomicRanges)
library(BSgenome.Hsapiens.NCBI.GRCh38)

model.names <- c("fraglen","GC")

txs <- txdf.theta$tx_id[txdf.theta$gene_id == "ENSG00000198918"]

res <- estimateAbundance(transcripts=ebt.theta[txs],
                         bam.files=bam.file,
                         fitpar=fitpar.small,
                         genome=Hsapiens,
                         model.names=model.names)
```

`extractAlpine` *Extract results from estimateAbundance run across genes*

Description

This function extracts estimates for a given model from a list over many genes, returning a matrix with dimensions: number of transcript x number of samples. Here, the count of compatible fragments aligning to the genes is used to estimate the FPKM, dividing out the previously used estimate `lib.sizes`.

Usage

```
extractAlpine(res, model, lib.sizes = 1e+06, divide.out = TRUE,
  transcripts = NULL)
```

Arguments

`res`	a list where each element is the output of estimateAbundance
`model`	the name of a model, corresponds to names of `models` used in fitBiasModels
`lib.sizes`	the vector of library sizes passed to estimateAbundance. not needed if `divide.out=FALSE`

divide.out	logical, whether to divide out the initial estimate of library size and to instead use the count of compatible fragments for genes calculated by estimateAbundance. Default is TRUE
transcripts	an optional GRangesList of the exons for each transcript. If this is provided, the output will be a SummarizedExperiment. The transcripts do not need to be provided in the correct order, extractAlpine will find the correct transcript by the names in res and put them in the correct order.

Value

a matrix of FPKM values across transcripts and samples, or a SummarizedExperiment if transcripts is provided

Examples

```
data(preprocessedData)
extractAlpine(res, "GC")
```

---

fitBiasModels *Fit bias models over single-isoform genes*

Description

This function estimates parameters for one or more bias models for a single sample over a set of single-isoform genes. ~100 medium to highly expressed genes should be sufficient to estimate the parameters robustly.

Usage

```
fitBiasModels(genes, bam.file, fragtypes, genome, models, readlength, minsize,
  maxsize, speedglm = TRUE, gc.knots = seq(from = 0.4, to = 0.6, length =
  3), gc.bk = c(0, 1), relpos.knots = seq(from = 0.25, to = 0.75, length =
  3), relpos.bk = c(0, 1))
```

Arguments

genes	a GRangesList with the exons of different single-isoform genes
bam.file	a character string pointing to an indexed BAM file
fragtypes	the output of buildFragtypes. must contain the potential fragment types for the genes named in genes
genome	a BSgenome object
models	a list of lists: the outer list describes multiple models. each element of the inner list has two elements: formula and offset. formula should be a character strings of an R formula describing the bias models, e.g. "count ~ ns(gc) + gene". The end of the string is required to be "+ gene". offset should be a

	character vector listing possible bias offsets to be used ("fraglen" or "vlmm"). Either offset or formula can be NULL for a model. See vignette for recommendations and details.
readlength	the read length
minsize	the minimum fragment length to model
maxsize	the maximum fragment length to model
speedglm	logical, whether to use speedglm to estimate the coefficients. Default is TRUE.
gc.knots	knots for the GC splines
gc.bk	boundary knots for the GC splines
relpos.knots	knots for the relative position splines
relpos.bk	boundary knots for the relative position splines

Value

a list with elements: coefs, summary, models, model.params, and optional offets: fraglen.density, vlmm.fivep, and vlmm.threep.

- **coefs** gives the estimated coefficients for the different models that specified formula.
- **summary** gives the tables with coefficients, standard errors and p-values,
- **models** stores the incoming models list,
- **model.params** stores parameters for the models, such as knot locations
- **fraglen.density** is a estimated density object for the fragment length distribution,
- **vlmm.fivep** and **vlmm.threep** store the observed and expected tabulations for the different orders of the VLMM for read start bias.

References

The complete bias model including fragment sequence bias is described in detail in the Supplemental Note of the following publication:

Love, M.I., Hogenesch, J.B., and Irizarry, R.A., Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation. Nature Biotechnologyh (2016) doi: 10.1038/nbt.3682

The read start variable length Markov model (VLMM) for addressing bias introduced by random hexamer priming was introduced in the following publication (the sequence bias model used in Cufflinks):

Roberts, A., Trapnell, C., Donaghey, J., Rinn, J.L., and Pachter, L., Improving RNA-Seq expression estimates by correcting for fragment bias. Genome Biology (2011) doi: 10.1186/gb-2011-12-3-r22

Examples

```
# see vignette for a more realistic example

# these next lines just write out a BAM file from R
# typically you would already have a BAM file
library(alpineData)
```

```
library(GenomicAlignments)
library(rtracklayer)
gap <- ERR188088()
dir <- system.file(package="alpineData", "extdata")
bam.file <- c("ERR188088" = file.path(dir,"ERR188088.bam"))
export(gap, con=bam.file)

library(GenomicRanges)
library(BSgenome.Hsapiens.NCBI.GRCh38)
data(preprocessedData)

readlength <- 75
minsize <- 125 # see vignette how to choose
maxsize <- 175 # see vignette how to choose

# here a very small subset, should be ~100 genes
gene.names <- names(ebt.fit)[6:8]
names(gene.names) <- gene.names
fragtypes <- lapply(gene.names, function(gene.name) {
                     buildFragtypes(ebt.fit[[gene.name]],
                                    Hsapiens, readlength,
                                    minsize, maxsize)
})
models <- list(
  "GC" = list(formula = "count ~ ns(gc,knots=gc.knots, Boundary.knots=gc.bk) + gene",
              offset=c("fraglen","vlmm"))
)

fitpar <- fitBiasModels(genes=ebt.fit[gene.names],
                        bam.file=bam.file,
                        fragtypes=fragtypes,
                        genome=Hsapiens,
                        models=models,
                        readlength=readlength,
                        minsize=minsize,
                        maxsize=maxsize)
```

`getFragmentWidths` *Get fragment widths*

Description

From a BAM file and a particular transcript (recommened to be the single isoform of a gene), this function returns estimates of the fragment widths, by mapping the fragment alignments to the transcript coordinates.

Usage

```
getFragmentWidths(bam.file, tx)
```

Arguments

bam.file	a character string pointing to a BAM file
tx	a GRanges object of the exons of a single isoform gene

Value

a numeric vector of estimated fragment widths

Examples

```
# these next lines just write out a BAM file from R
# typically you would already have a BAM file
library(alpineData)
library(GenomicAlignments)
library(rtracklayer)
gap <- ERR188088()
dir <- system.file(package="alpineData", "extdata")
bam.file <- c("ERR188088" = file.path(dir,"ERR188088.bam"))
export(gap, con=bam.file)

data(preprocessedData)

w <- getFragmentWidths(bam.file, ebt.fit[[2]])
quantile(w, c(.025, .975))
```

---

getReadLength *Get read length*

Description

Gets the length of the first read in a BAM file

Usage

```
getReadLength(bam.files)
```

Arguments

bam.files	a character vector pointing to BAM files

Value

a numeric vector, one number per BAM file, the length of the first read in the file

Examples

```
# these next lines just write out a BAM file from R
# typically you would already have a BAM file
library(alpineData)
library(GenomicAlignments)
library(rtracklayer)
gap <- ERR188088()
dir <- system.file(package="alpineData", "extdata")
bam.file <- c("ERR188088" = file.path(dir,"ERR188088.bam"))
export(gap, con=bam.file)

getReadLength(bam.file)
```

`mergeGenes` *Merge overlapping "genes" into gene clusters*

Description

This function looks for overlapping exons in ebg. The overlapping "genes" are used to form a graph. Any connected components in the graph (sets of "genes" which can be reached from each other through overlap relations) are connected into a new gene cluster, which is given the suffix "_mrg" and using one of the original gene names.

Usage

```
mergeGenes(ebg, txdf, ignore.strand = TRUE)
```

Arguments

ebg	an exons-by-genes GRangesList, created with `exonsBy`
txdf	a data.frame created by running `select` on a TxDb object. Must have a column GENEID.
ignore.strand	Default is TRUE.

Value

a manipulated `txdf`.

Examples

```
library(GenomicRanges)
txdf <- data.frame(GENEID=c("101","102","103","104"))
ebg <- GRangesList(GRanges("1",IRanges(c(100,200),width=50)),
                   GRanges("1",IRanges(c(200,300),width=50)),
                   GRanges("1",IRanges(c(300,400),width=50)),
                   GRanges("1",IRanges(c(500,600),width=50)))
names(ebg) <- c("101","102","103","104")
mergeGenes(ebg, txdf)
```

normalizeDESeq *DESeq median ratio normalization for matrix*

Description

Simple implementation of DESeq median ratio normalization

Usage

```
normalizeDESeq(mat, cutoff)
```

Arguments

mat	a matrix of numeric values
cutoff	a numeric value to be used as the cutoff for the row means of mat. Only rows with row mean larger than cutoff are used for calculating the size factors

Value

a matrix with the median ratio size factors divided out

References

Anders, S. and Huber, W., Differential expression analysis for sequence count data. Genome Biology (2010) doi: 10.1186/gb-2010-11-10-r106

Examples

```
x <- runif(50,1,100)
mat <- cbind(x, 2*x, 3*x)
norm.mat <- normalizeDESeq(mat, 5)
```

plotFragLen *Plot fragment length distribution over samples*

Description

Plots the fragment length distribution.

Usage

```
plotFragLen(fitpar, col, lty)
```

Arguments

fitpar	a list of the output of fitBiasModels over samples
col	a vector of colors
lty	a vector of line types

Value

plot

Examples

```
# fitpar was fit using identical code
# as found in the vignette, except with
# 25 genes, and with fragment size in 80-350 bp
data(preprocessedData)
perf <- rep(1:2, each=2)
plotFragLen(fitpar, col=perf)
```

---

plotGC *Plot the fragment GC bias over samples*

Description

Plots smooth curves of the log fragment rate over fragment GC content.

Usage

```
plotGC(fitpar, model, col, lty, ylim, gc.range = NULL, return.type = 0)
```

Arguments

fitpar	a list of the output of fitBiasModels over samples
model	the name of one of the models
col	a vector of colors
lty	a vector of line types
ylim	the y limits for the plot
gc.range	a numeric of length two, the range of the fragment GC content. By default, [.2,.8] for plotting and [0,1] for returning a matrix
return.type	a numeric, either 0: make a plot, 1: skip the plot and return a matrix of log fragment rate, 2: skip the plot and return a matrix of probabilities

Value

Either plot, or if return.type is 1 or 2, a matrix

Examples

```
# fitpar was fit using identical code
# as found in the vignette, except with
# 25 genes, and with fragment size in 80-350 bp
data(preprocessedData)
perf <- rep(1:2, each=2)
plotGC(fitpar, "all", col=perf)
```

`plotGRL` *Simple segments plot for GRangesList*

Description

Simple segments plot for GRangesList

Usage

```
plotGRL(grl, ...)
```

Arguments

`grl`	GRangesList object
`...`	passed to plot

Value

plot

Examples

```
library(GenomicRanges)
grl <- GRangesList(GRanges("1",IRanges(c(100,200,300),width=50)),
                   GRanges("1",IRanges(c(100,300),width=c(75,50))),
                   GRanges("1",IRanges(c(100,200,400),width=c(75,50,50))),
                   GRanges("1",IRanges(c(200,300,400),width=50)))
plotGRL(grl)
```

`plotOrder0` *Plot parameters of the variable length Markov model (VLMM) for read starts*

Description

This function plots portions of the Cufflinks VLMM for read start bias. The natural log of observed over expected is shown, such that 0 indicates no contribution of a position to the read start bias. As the variable lenght Markov model has different dependencies for different positions (see Roberts et al, 2011), it is difficult to show all the 744 parameters simultaneously. Instead this function offers to show the 0-order terms for all positions, or the 1st and 2nd order terms for selected positions within the read start sequence. For the 1- and 2-order terms, the log bias is shown for each nucleotide (A,C,T,G) given the previous nucleotide (1-order) or di-nucleotide (2-order).

Usage

```
plotOrder0(order0, ...)

plotOrder1(order1, pos1)

plotOrder2(order2, pos2)
```

Arguments

`order0`	the "order0" element of the list named "vlmm.fivep" or "vlmm.threep" within the list that is the output of fitBiasModels
`...`	parameters passed to `plot`
`order1`	as for "order0" but "order1"
`pos1`	the position of the 1st order VLMM to plot
`order2`	as for "order0" but "order2"
`pos2`	the position of the 2nd order VLMM to plot

Value

plot

Functions

- `plotOrder1`: Plot first order parameters for a position
- `plotOrder2`: Plot second order parameters for a position

References

Roberts et al, "Improving RNA-Seq expression estimates by correcting for fragment bias" Genome Biology (2011) doi:101186/gb-2011-12-3-r22

Examples

```
# fitpar was fit using identical code
# as found in the vignette, except with
# 25 genes, and with fragment size in 80-350 bp
data(preprocessedData)
plotOrder0(fitpar[[1]][["vlmm.fivep"]][["order0"]])
plotOrder1(fitpar[[1]][["vlmm.fivep"]][["order1"]], pos1=5:19)
plotOrder2(fitpar[[1]][["vlmm.fivep"]][["order2"]], pos2=8:17)
```

`plotRelPos` *Plot relative position bias over samples*

Description

Plots the smooth curves of log fragment rate over relative position.

Usage

```
plotRelPos(fitpar, model, col, lty, ylim)
```

Arguments

`fitpar`	a list of the output of fitBiasModels over samples
`model`	the name of one of the models
`col`	a vector of colors
`lty`	a vector of line types
`ylim`	the y limits for the plot

Value

plot

Examples

```
# fitpar was fit using identical code
# as found in the vignette, except with
# 25 genes, and with fragment size in 80-350 bp
data(preprocessedData)
perf <- rep(1:2, each=2)
plotRelPos(fitpar, "all", col=perf)
```

predictCoverage *Predict coverage for a single-isoform gene*

Description

Predict coverage for a single-isoform gene given fitted bias parameters in a set of models, and compare to the observed fragment coverage.

Usage

```
predictCoverage(gene, bam.files, fitpar, genome, model.names)
```

Arguments

gene	a GRangesList with the exons of different genes
bam.files	a character string pointing to indexed BAM files
fitpar	the output of running fitBiasModels
genome	a BSgenome object
model.names	a character vector listing the models, see same argument in estimateAbundance

Details

Note that if the range between minsize and maxsize does not cover most of the fragment length distribution, the predicted coverage will underestimate the observed coverage.

Value

a list with elements frag.cov, the observed fragment coverage from the bam.files and pred.cov, a list with the predicted fragment coverage for each of the models.

Examples

```
# these next lines just write out a BAM file from R
# typically you would already have a BAM file
library(alpineData)
library(GenomicAlignments)
library(rtracklayer)
gap <- ERR188088()
dir <- system.file(package="alpineData", "extdata")
bam.file <- c("ERR188088" = file.path(dir,"ERR188088.bam"))
export(gap, con=bam.file)

data(preprocessedData)
library(BSgenome.Hsapiens.NCBI.GRCh38)

model.names <- c("fraglen","fraglen.vlmm","GC","all")

pred.cov <- predictCoverage(gene=ebt.fit[["ENST00000379660"]],
```

```
                            bam.files=bam.file,
                            fitpar=fitpar.small,
                            genome=Hsapiens,
                            model.names=model.names)

# plot the coverage:
# note that, because [125,175] bp range specified in fitpar.small
# does not cover the fragment width distribution, the predicted curves
# will underestimate the observed. we correct here post-hoc

frag.cov <- pred.cov[["ERR188088"]][["frag.cov"]]
plot(frag.cov, type="l", lwd=3, ylim=c(0,max(frag.cov)*1.5))
for (i in seq_along(model.names)) {
  m <- model.names[i]
  pred <- pred.cov[["ERR188088"]][["pred.cov"]][[m]]
  lines(pred/mean(pred)*mean(frag.cov), col=i+1, lwd=3)
}
legend("topright", legend=c("observed",model.names),
       col=seq_len(length(model.names)+1), lwd=3)
```

`preprocessedData` *Preprocessed data for vignettes and examples*

Description

The following data objects are prepared for use in the alpine vignette and examples pages, as the preparation of these objects requires either long running time or a large amount of disk space.

Format

`ebt.fit` and `ebt.theta` are GRangesList. `fitpar`, `fitpar.small`, `res` are lists created by alpine functions. `genes.theta` is a character vector. `txdf.theta` is a DataFrame.

Details

- **ebt.fit** - the GRangesList prepared in the vignette for fitting the bias models
- **fitpar** - the fitted parameters, similar to those made in the vignette, but using `minsize=80` and `maxsize=350`
- **fitpar.small** - the fitted parameters from the vignette, returned by fitBiasModels
- **res** - the results object from the vignette, returned by estimateAbundance
- **ebt.theta** - the GRangesList prepared in the vignette for running estimateAbundance
- **genes.theta** - the names of genes used in the vignette for running estimateAbundance
- **txdf.theta** - the DataFrame of gene and transcript information used in the vignette for running estimateAbundance

Source

See vignette for details of object construction. The alignments come from alpineData (4 samples from GEUVADIS project), the Ensembl gene annotations come from `Homo_sapiens.GRCh38.84.gtf`, and the genome is `BSgenome.Hsapiens.NCBI.GRCh38`.

`splitGenesAcrossChroms` *Split genes that have isoforms across chromosomes*

Description

This function simply splits apart genes which have isoforms across multiple chromosomes. New "genes" are created with the suffix "_cs" and a number.

Usage

```
splitGenesAcrossChroms(ebg, txdf)
```

Arguments

`ebg`	an exons-by-genes GRangesList, created with `exonsBy`
`txdf`	a data.frame created by running `select` on a TxDb object. Must have columns TXCHROM and GENEID

Value

a list of manipulated `ebg` and `txdf`

Examples

```
library(GenomicRanges)
txdf <- data.frame(TXCHROM=c("1","1","2"),
                   GENEID=c("101","102","102"))
ebg <- GRangesList(GRanges("1",IRanges(c(100,200),width=50)),
                   GRanges(c("1","2"),IRanges(c(400,100),width=50)))
names(ebg) <- c("101","102")
splitGenesAcrossChroms(ebg, txdf)
```

splitLongGenes *Split very long genes*

Description

This function splits genes which have a very long range (e.g. 1 Mb), and new "genes" are formed where each isoform is its own "gene", with the suffix "_ls" and a number. It makes sense to turn each isoform into its own gene only if this function is followed by mergeGenes.

Usage

```
splitLongGenes(ebg, ebt, txdf, long = 1e+06)
```

Arguments

ebg	an exons-by-genes GRangesList, created with exonsBy
ebt	an exons-by-tx GRangesList, created with exonsBy
txdf	a data.frame created by running select on a TxDb object. Must have columns GENEID and TXID, where TXID corresponds to the names of ebt.
long	a numeric value such that ranges longer than this are "long"

Value

a list of manipulated ebg and txdf

Examples

```
library(GenomicRanges)
txdf <- data.frame(GENEID=c("101","101","102"),
                   TXID=c("201","202","203"))
ebt <- GRangesList(GRanges("1",IRanges(c(100,200),width=50)),
                   GRanges("1",IRanges(2e6 + c(100,200),width=50)),
                   GRanges("1",IRanges(3e6 + c(100,200),width=50)))
names(ebt) <- c("201","202","203")
ebg <- GRangesList(reduce(unlist(ebt[1:2])),ebt[[3]])
names(ebg) <- c("101","102")
splitLongGenes(ebg, ebt, txdf)
```

Index