Package 'diffHic'

April 23, 2016

Version 1.2.2

Date 2016/01/15

Title Differential analyis of Hi-C data

Author Aaron Lun <alun@wehi.edu.au>

Maintainer Aaron Lun <alun@wehi.edu.au>

Depends R (>= 3.2.0), GenomicRanges

Imports Rsamtools, Biostrings, BSgenome, rhdf5, edgeR, limma, csaw, locfit, methods, IRanges, S4Vectors, GenomeInfoDb, BiocGenerics

Suggests BSgenome.Ecoli.NCBI.20080805

biocViews MultipleComparison, Preprocessing, Sequencing, Coverage, Alignment, Normalization, Clustering, HiC

Description Detects differential interactions across biological conditions in a Hi-C experiment. Methods are provided for read alignment and data pre-processing into interaction counts. Statistical analysis is based on edgeR and supports normalization and filtering. Several visualization options are also available.

License GPL-3

NeedsCompilation yes

R topics documented:

boxPairs 2
clusterPairs 4
compartmentalize 6
connectCounts 8
consolidatePairs 10
correctedContact 12
cutGenome 15
diffHicUsersGuide 16
DIList-class 17
DIList-wrappers 20

DNaseHiC . . . 21
domainDirections . . . 23
enrichedPairs . . . 25
Filtering diagonals . . . 27
Filtering methods . . . 29
filterPeaks . . . 31
getArea . . . 32
getDistance . . . 34
getPairData . . . 35
loadData . . . 36
marginCounts . . . 37
mergePairs . . . 39
neighborCounts . . . 40
normalizeCNV . . . 42
pairParam . . . 44
plotDI . . . 46
plotPlaid . . . 48
preparePairs . . . 50
prunePairs . . . 53
savePairs . . . 55
squareCounts . . . 57
totalCounts . . . 59

Index **61**

`boxPairs` *Put bin pairs into boxes*

Description

Match smaller bin pairs to the larger bin pairs in which they are nested.

Usage

```
boxPairs(..., reference, minbox=FALSE)
```

Arguments

`...`	one or more named `DIList` objects produced by `squareCounts`, with smaller bin sizes than `reference`
`reference`	an integer scalar specifying the reference bin size
`minbox`	a logical scalar indicating whether coordinates for the minimum bounding box should be returned

Details

Consider the bin size specified in `reference`. Pairs of these bins are referred to here as the parent bin pairs, and are described in the output `pairs` and `region`. The function accepts a number of `DIList` objects of bin pair data in the ellipsis, referred to here as input bin pairs. The aim is to identify the parent bin pair in which each input bin pair is nested.

All input `DIList` objects in the ellipsis must be constructed carefully. In particular, the value of `width` in `squareCounts` must be such that `reference` is an exact multiple of each `width`. This is necessary to ensure complete nesting. Otherwise, the behavior of the function will not be clearly defined.

In the output, one vector will be present in `indices` for each input `DIList` in the ellipsis. In each vector, each entry represents an index for a single input bin pair in the corresponding `DIList`. This index points to the entries in `anchors` and `targets` that specify the coordinates of the parent bin pair. Thus, bin pairs with the same index are nested in the same parent.

Some users may wish to identify bin pairs in one `DIList` that are nested within bin pairs in another `DIList`. This can be done by supplying both `DIList` objects in the ellipsis, and leaving `reference` unspecified. The value of `reference` will be automatically selected as the largest width of the supplied `DIList` objects. Nesting can be identified by `matching` the output indices for the smaller bin pairs to those of the larger bin pairs.

If `minbox=TRUE`, the coordinates in `anchors` and `targets` represent the minimum bounding box for all nested bin pairs in each parent. This may be more precise if nesting only occurs in a portion of the interaction space of the parent bin pair.

Value

A list object containing:

`indices` a named list of integer vectors for every `DIList` in the ellipsis, see Details

`anchors,targets` `GRanges` objects specifying the coordinates of the parent bin pair or, if `minbox=TRUE`, the minimum bounding box

Author(s)

Aaron Lun

See Also

`squareCounts`

Examples

```
# Setting up the objects.
a <- 10
b <- 20
cuts <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))
param <- pairParam(cuts)

all.combos <- combn(length(cuts), 2) # Bin size of 1.
```

```
y <- DIList(matrix(0, ncol(all.combos), 1), anchors=all.combos[2,],
    targets=all.combos[1,], regions=cuts, exptData=List(param=param, width=1))

a5 <- a/5
b5 <- b/5
all.combos2 <- combn(length(cuts)/5, 2) # Bin size of 5.
y2 <- DIList(matrix(0, ncol(all.combos2), 1), anchors=all.combos2[2,],
    targets=all.combos2[1,], exptData=List(param=param, width=5),
regions=GRanges(rep(c("chrA", "chrB"), c(a5, b5)),
IRanges(c((1:a5-1)*5+1, (1:b5-1)*5+1), c(1:a5*5, 1:b5*5))))

# Clustering.
boxPairs(reference=5, larger=y2, smaller=y)
boxPairs(reference=10, larger=y2, smaller=y)
boxPairs(reference=10, larger=y2, smaller=y, minbox=TRUE)
boxPairs(larger=y2, smaller=y)
```

`clusterPairs` *Cluster bin pairs*

Description

Aggregate bin pairs into local clusters for summarization.

Usage

```
clusterPairs(..., tol, upper=1e6)
```

Arguments

`...`	one or more `DIList` objects, optionally named
`tol`	a numeric scalar specifying the maximum distance between bin pairs
`upper`	a numeric scalar specifying the maximum size of each cluster

Details

Clustering is performed by putting a interaction in a cluster if the smallest Chebyshev distance to any interaction already inside the cluster is less than `tol`. This is a cross between single-linkage approaches and density-based methods, especially after filtering removes low-density regions. In this manner, adjacent events in the interaction space can be clustered together. Interactions that are assigned with the same cluster ID belong to the same cluster.

The input `data` objects can be taken from the output of `squareCounts` or `connectCounts`. For the former, inputs can consist of interactions with multiple bin sizes. It would be prudent to filter the former based on the average abundances, to reduce the density of bin pairs in the interaction space. Otherwise, clusters may be too large to be easily interpreted.

Alternatively, to avoid excessively large clusters, this function can also split each cluster into roughly-equally sized subclusters. The maximum value of any dimension of the subclusters is

approxiamtely equal to upper. This aims to improve the spatial interpretability of the clustering result.

There is no guarantee that each cluster forms a regular shape in the interaction space. Instead, a minimum bounding box is reported containing all bin pairs in each cluster. The coordinates of the box for each cluster is stored in each row of the output anchors and targets. The cluster ID in each indices vector represents the row index for these coordinates.

Value

A list containing indices, a named list of integer vectors where each vector contains a cluster ID for each interaction in the corresponding input DIList object; anchors, a GRanges object containing the dimensions of the bounding box of each cluster on the anchor chromosome; and targets, a GRanges object containing the corresponding dimensions on the target chromosome;

Author(s)

Aaron Lun

See Also

squareCounts

Examples

```
# Setting up the object.
a <- 10
b <- 20
regions <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))

set.seed(3423)
all.anchors <- sample(length(regions), 50, replace=TRUE)
all.targets <- as.integer(runif(50, 1, all.anchors+1))
y <- DIList(matrix(0, 50, 1), anchors=all.anchors, targets=all.targets,
    regions=regions, exptData=List(width=1))

# Clustering; note, small tolerances are used in this toy example.
clusterPairs(y, tol=1)
clusterPairs(y, tol=3)
clusterPairs(y, tol=5)
clusterPairs(y, tol=5, upper=5)

# Multiple bin sizes allowed.
a2 <- a/2
b2 <- b/2
regions2 <- GRanges(rep(c("chrA", "chrB"), c(a2, b2)),
    IRanges(c(1:a2*2, 1:b2*2), c(1:a2*2, 1:b2*2)))
all.anchors2 <- sample(length(regions2), 10, replace=TRUE)
all.targets2 <- as.integer(runif(10, 1, all.anchors2+1))
y2 <- DIList(matrix(0, 10, 1), anchors=all.anchors2, targets=all.targets2,
    regions=regions2, exptData=List(width=2))
```

```
clusterPairs(y, y2, tol=1)
clusterPairs(y, y2, tol=3)
clusterPairs(y, y2, tol=5)
clusterPairs(y, tol=5, upper=5)
```

`compartmentalize` *Identify genomic compartments*

Description

Use contact matrices to identify self-interacting genomic compartments

Usage

```
compartmentalize(data, centers=2, dist.correct=TRUE,
    cov.correct=TRUE, robust.cov=5, ...)
```

Arguments

`data`	a `DIList` object containing bin pair data, like that produced by `squareCounts`
`centers`	an integer scalar, specifying the number of clusters to form in `kmeans`
`dist.correct`	a logical scalar, indicating whether abundances should be corrected for distance biases
`cov.correct`	a logical scalar, indicating whether abundances should be corrected for coverage biases
`robust.cov`	a numeric scalar, specifying the multiple of MADs beyond which coverage outliers are removed
`...`	other arguments to pass to `kmeans`

Details

This function uses the interaction space to partition each linear chromosome into compartments. Bins in the same compartment interact more frequently with each other compared to bins in different compartments. This forms a checkerboard-like pattern in the interaction space that can be used to define the genomic intervals in each compartment. Typically, one compartment is gene-rich and is defined as "open", while the other is gene-poor and defined as "closed".

Compartment identification is done by setting up a contact matrix where each row/rolumn represents a bin and each matrix entry contains the frequency of contacts between bins. Bins (i.e., rows) with similar interaction profiles (i.e., entries across columns) are clustered together with the k-means method. Those with the same ID in the output `compartment` vector are in the same compartment. Note that clustering is done separately for each chromosome, so bins with the same ID across different chromosomes cannot be interpreted as being in the same compartment.

If `dist.correct=TRUE`, frequencies are normalized to mitigate the effect of distance and to improve the visibility of long-range interactions. This is done by computing the residuals of the distance-dependent trend - see `filterTrended` for more details. If `cov.correct=TRUE`, frequencies are

also normalized to eliminate coverage biases betwen bins. This is done by computing the average coverage of each row/column, and dividing each matrix entry by the square root averages of the relevant row and column.

Extremely low-coverage regions such as telomeres and centromeres can confound k-means clustering. To protect against this, all bins with (distance-corrected) coverages that are more than `robust.cov` MADs away from the median coverage of each chromosome are identified and removed. These bins will be marked with `NA` in the returned `compartment` for that chromosome. To turn off robustification, set `robust.cov` to `NA`.

By default, `centers` is set to 2 to model the open and closed compartments. While a larger value can be used to obtain more clusters, care is required as the interpretation of the resulting compartments becomes more difficult. If desired, users can also apply their own clustering methods on the `matrix` returned in the output.

Value

A named list of lists is returned where each internal list corresponds to a chromosome in `data` and contains `compartment`, an integer vector of compartment IDs for all bins in that chromosome; and `matrix`, a numeric matrix of (normalized) contact frequencies for the intra-chromosomal space. Entries in `compartment` and rows/columns in `matrix` are named according to the matching index of `regions(data)`.

Author(s)

Aaron Lun

References

Lieberman-Aiden E et al. (2009). Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science* 326, 289-293.

Lajoie BR, Dekker J, Kaplan N (2014). The hitchhiker's guide to Hi-C analysis: practical guidelines. *Methods* 72, 65-75.

See Also

`squareCounts`, `filterTrended`, `kmeans`

Examples

```
# Dummying up some data.
set.seed(3426)
npts <- 100
npairs <- 5000
nlibs <- 4
anchors <- sample(npts, npairs, replace=TRUE)
targets <- sample(npts, npairs, replace=TRUE)
data <- DIList(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs),
  totals=runif(nlibs, 1e6, 2e6), anchors=pmax(anchors, targets), targets=pmin(anchors, targets),
  regions=GRanges(c(rep("chrA", 80), rep("chrB", 20)), IRanges(c(1:80, 1:20), c(1:80, 1:20))),
   exptData=List(width=1))
```

```
# Running compartmentalization.
out <- compartmentalize(data)
head(out$chrA$compartment)
dim(out$chrA$matrix)
head(out$chrB$compartment)
dim(out$chrB$matrix)

test <- compartmentalize(data, cov.correct=FALSE)
test <- compartmentalize(data, dist.correct=FALSE)
test <- compartmentalize(data, robust.cov=NA)
```

`connectCounts` *Count connecting read pairs*

Description

Count the number of read pairs connecting pairs of user-specified regions

Usage

```
connectCounts(files, param, regions, filter=1L, type="any", second.regions=NULL)
```

Arguments

`files`	a character vector containing the paths to the count file for each library
`param`	a `pairParam` object containing read extraction parameters
`regions`	a `GRanges` object specifying the regions between which read pairs should be counted
`filter`	an integer scalar specifying the minimum count for each interaction
`type`	a character string specifying how restriction fragments should be assigned to regions
`second.regions`	a `GRanges` object containing the second regions of interest, or an integer scalar specifying the bin size

Details

Interactions of interest are defined as those formed by pairs of elements in `regions`. The number of read pairs connecting each pair of elements can then be counted in each library. This can be useful for quantifying/summarizing interactions between genomic features, e.g., promoters or gene bodies.

For a pair of intervals in `regions`, the interaction count is defined as the number of read pairs with one read in each interval (after rounding each interval to a fragment; see below). To avoid reporting weak interactions, pairs can be filtered to retain only those with a count sum across all libraries above `filter`. In each pair, the anchor interval is defined as that with the higher start position. Note that the end position may not be higher, due to the possibility of nested intervals in `regions`.

The value of type feeds into findOverlaps and controls the manner in which restriction fragments are assigned to each region. By default, a restriction fragment is assigned to one or more regions if said fragment overlaps with any part of those regions. This means that the boundaries of each region are expanded outwards to obtain the effective coordinates in the output region. In contrast, setting type="within" would contract each region inwards.

The modified regions can be extracted from the regions slot in the output DIList object. These will be reordered according to the new start positions. The ordering permutation can be recovered from the original metadata field of the GRanges object. Similarly, the number of restriction fragments assigned to each interval is stored in the nfrags metadata field.

Counting will consider the values of restrict, discard and cap in param. See pairParam for more details.

Value

A DIList is returned, specifying the number of read pairs in each library that are mapped between pairs of regions, or between regions and second.regions.

Matching to a second set of regions

The second.regions argument allows specification of a second set of regions, where interactions are only considered between one entry in regions and one entry in second.regions. This differs from supplying all regions to regions, which would consider all pairwise interactions between regions regardless of whether they belong in the first or second set. If an integer scalar is supplied as second.regions, this value is used as a width to partition the genome into bins. These bins are then used as the set of second regions.

Specification of second.regions is useful for efficiently identifying interactions between two sets of regions. For example, the first set can be set to several "viewpoint" regions of interest. This is similar to the bait region in 4C-seq, or the captured regions in Capture Hi-C. Interactions between these viewpoints and the rest of the genome can then be examined by setting second.regions to some appropriate bin size.

The output DIList will merge all regions and second.regions into a single GRanges object. However, those in the second set can be distinguished with the is.second metadata field. Each original index will also point towards the corresponding entry in the original second.regions when is.second=TRUE. Similarly, if is.second=FALSE, the index will point towards the corresponding entry in the original regions.

Note that this function does *not* guarantee that the second set of regions will be present as the anchor or target regions. Those definitions are dependent on the sorting order of the coordinates for all regions. Users should use the is.second field to identify the region from the second set in each interaction.

Author(s)

Aaron Lun

See Also

squareCounts, findOverlaps, DIList-class

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

# Setting up the parameters
fout <- "output"
invisible(preparePairs(hic.file, param, fout))
regions <- suppressWarnings(c(
    GRanges("chrA", IRanges(c(1, 100, 150), c(20, 140, 160))),
    GRanges("chrB", IRanges(50, 100))))

# Collating to count combinations.
con <- connectCounts(fout, param, regions=regions, filter=1L)
head(counts(con))
con <- connectCounts(fout, param, regions=regions, filter=1L, type="within")
head(counts(con))

# Still works with restriction and other parameters.
con <- connectCounts(fout, param=reform(param, restrict="chrA"),
    regions=regions, filter=1L)
head(counts(con))
con <- connectCounts(fout, param=reform(param, discard=GRanges("chrA", IRanges(1, 50))),
    regions=regions, filter=1L)
head(counts(con))
con <- connectCounts(fout, param=reform(param, cap=1), regions=regions, filter=1L)
head(counts(con))

# Specifying a second region.
regions2 <- suppressWarnings(c(
    GRanges("chrA", IRanges(c(50, 100), c(100, 200))),
    GRanges("chrB", IRanges(1, 50))))

con <- connectCounts(fout, param, regions=regions, filter=1L, second.region=regions2)
head(anchors(con))
head(targets(con))
con <- connectCounts(fout, param, regions=regions, filter=1L, second.region=50)
head(anchors(con))
head(targets(con))
```

`consolidatePairs` *Consolidate results for interactions*

Description

Consolidate differential testing results for interactions from separate analyses.

Usage

```
consolidatePairs(indices, result.list, equiweight=TRUE, combine.args=list())
```

Arguments

indices	a list of index vectors, specifying the cluster ID to which each interaction belongs
result.list	a list of data frames containing the DB test results for each interaction
equiweight	a logical scalar indicating whether equal weighting from each bin size should be enforced
combine.args	a list of parameters to pass to combineTests

Details

Interactions from different analyses can be aggregated together using boxPairs or clusterPairs. For example, test results can be consolidated for bin pairs of differing sizes. This usually produces a indices vector that can be used as an input here. Briefly, each vector in indices should correspond to one analysis, and each entry of that vector should correspond to an analyzed interaction. The vector itself holds cluster IDs, such that interactions within/between analyses with the same ID belong in the same cluster.

For all bin pairs in a cluster, the associated p-values are combined in combineTests using a weighted version of Simes' method. This yields a single combined p-value, representing the evidence against the global null. When equiweight=TRUE, the weight of a p-value of each bin pair is inversely proportional to the number of bin pairs of the same size in that parent bin pair. This ensures that the results are not dominated by numerous smaller bin pairs.

Value

A data frame is returned containing the combined DB results for each cluster.

Author(s)

Aaron Lun

See Also

combineTests, boxPairs, clusterPairs

Examples

```
# Setting up the objects.
a <- 10
b <- 20
cuts <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))
param <- pairParam(cuts)

all.combos <- combn(length(cuts), 2) # Bin size of 1.
y <- DIList(matrix(0, ncol(all.combos), 1), anchors=all.combos[2,],
    targets=all.combos[1,], regions=cuts, exptData=List(param=param, width=1))
```

```
a5 <- a/5
b5 <- b/5
all.combos2 <- combn(length(cuts)/5, 2) # Bin size of 5.
y2 <- DIList(matrix(0, ncol(all.combos2), 1), anchors=all.combos2[2,],
    targets=all.combos2[1,], exptData=List(param=param, width=5),
regions=GRanges(rep(c("chrA", "chrB"), c(a5, b5)),
IRanges(c((1:a5-1)*5+1, (1:b5-1)*5+1), c(1:a5*5, 1:b5*5))))

result1 <- data.frame(logFC=rnorm(nrow(y)), PValue=runif(nrow(y)), logCPM=0)
result2 <- data.frame(logFC=rnorm(nrow(y2)), PValue=runif(nrow(y2)), logCPM=0)

# Consolidating.
boxed <- boxPairs(y, y2)
out <- consolidatePairs(boxed$indices, list(result1, result2))
head(out)
out <- consolidatePairs(boxed$indices, list(result1, result2), equiweight=FALSE)
head(out)

# Repeating with three sizes.
a10 <- a/10
b10 <- b/10
all.combos3 <- combn(length(cuts)/10, 2) # Bin size of 10.
y3 <- DIList(matrix(0, ncol(all.combos3), 1), anchors=all.combos3[2,],
targets=all.combos3[1,], exptData=List(param=param, width=10),
regions=GRanges(rep(c("chrA", "chrB"), c(a10, b10)),
IRanges(c((1:a10-1)*10+1, (1:b10-1)*10+1), c(1:a10*10, 1:b10*10))))
result3 <- data.frame(logFC=rnorm(nrow(y3)), PValue=runif(nrow(y3)), logCPM=0)

boxed <- boxPairs(y, y2, y3)
out <- consolidatePairs(boxed$indices, list(result1, result2, result3))
head(out)
```

`correctedContact` *Iterative correction of Hi-C counts*

Description

Perform iterative correction on counts for Hi-C interactions to correct for biases between fragments.

Usage

```
correctedContact(data, iterations=50, exclude.local=1, ignore.low=0.02,
    winsor.high=0.02, average=TRUE, dist.correct=FALSE)
```

Arguments

`data`	a `DIList` object produced by `squareCounts`
`iterations`	an integer scalar specifying the number of correction iterations

exclude.local	an integer scalar, indicating the distance off the diagonal under which bin pairs are excluded
ignore.low	a numeric scalar, indicating the proportion of low-abundance bins to ignore
winsor.high	a numeric scalar indicating the proportion of high-abundance bin pairs to winsorize
average	a logical scalar specifying whether counts should be averaged across libraries
dist.correct	a logical scalar indicating whether to correct for distance effects

Details

This function implements the iterative correction procedure described by Imakaev *et al.* in their 2012 paper. Briefly, this aims to factorize the count for each bin pair into the bias for the anchor bin, the bias for the target bin and the true interaction probability. The bias represents the ease of sequencing/mapping/other for that genomic region.

The data argument should be generated by taking the output of squareCounts after setting filter=1. Filtering should be avoided as counts in low-abundance bin pairs may be informative upon summation for each bin. For example, a large count sum for a bin may be formed from many bin pairs with low counts. Removal of those bin pairs would result in loss of information.

For average=TRUE, if multiple libraries are used to generate data, an average count will be computed for each bin pairs across all libraries using mglmOneGroup. The average count will then be used for correction. Otherwise, correction will be performed on the counts for each library separately.

The maximum step size in the output can be used as a measure of convergence. Ideally, the step size should approach 1 as iterations pass. This indicates that the correction procedure is converging to a single solution, as the maximum change to the computed biases is decreasing.

Value

A list with several components.

truth: a numeric vector containing the true interaction probabilities for each bin pair

bias: a numeric vector of biases for all bins

max: a numeric vector containing the maximum fold-change change in biases at each iteration

trend: a numeric vector specifying the fitted value for the distance-dependent trend, if dist.correct=TRUE

If average=FALSE, each component is a numeric matrix instead. Each column of the matrix contains the specified information for each library in data.

Additional parameter settings

Some robustness is provided by winsorizing out strong interactions with winsor.high to ensure that they do not overly influence the computed biases. This is useful for removing spikes around repeat regions or due to PCR duplication. Low-abundance bins can also be removed with ignore.low to avoid instability during correction, though this will result in NA values in the output.

Local bin pairs can be excluded as these are typically irrelevant to long-range interactions. They are also typically very high-abundance and may have excessive weight during correction, if not

removed. This can be done by removing all bin pairs where the difference between the anchor and target indices is less than `exclude.local`. Setting `exclude.local=NA` will only use inter-chromosomal bin pairs for correction.

If `dist.correct=TRUE`, abundances will be adjusted for distance-dependent effects. This is done by computing residuals from the fitted distance-abundance trend, using the `filterTrended` function. These residuals are then used for iterative correction, such that local interactions will not always have higher contact probabilities.

Ideally, the probability sums to unity across all bin pairs for a given bin (ignoring `NA` entries). This is complicated by winsorizing of high-abundance interactions and removal of local interactions. These interactions are not involved in correction, but are still reported in the output `truth`. As a result, the sum may not equal unity, i.e., values are not strictly interpretable as probabilities.

Author(s)

Aaron Lun

References

Imakaev M et al. (2012). Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nat. Methods* 9, 999-1003.

See Also

`squareCounts`, `mglmOneGroup`

Examples

```
# Dummying up some data.
set.seed(3423746)
npts <- 100
npairs <- 5000
nlibs <- 4
anchors <- sample(npts, npairs, replace=TRUE)
targets <- sample(npts, npairs, replace=TRUE)
data <- DIList(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs),
totals=runif(nlibs, 1e6, 2e6), anchors=pmax(anchors, targets), targets=pmin(anchors, targets),
regions=GRanges("chrA", IRanges(1:npts, 1:npts)))

# Correcting.
stuff <- correctedContact(data)
head(stuff$truth)
head(stuff$bias)
plot(stuff$max)

# Different behavior with average=FALSE.
stuff <- correctedContact(data, average=FALSE)
head(stuff$truth)
head(stuff$bias)
head(stuff$max)

# Creating an offset matrix, for use in glmFit.
```

```
anchor.bias <- stuff$bias[anchors(data, id=TRUE),]
target.bias <- stuff$bias[targets(data, id=TRUE),]
offsets <- log(anchor.bias * target.bias)
difference <- log(stuff$truth) - (log(counts(data)) - offsets) # effective function of offset in GLMs.
stopifnot(all(is.na(difference) | difference < 1e-8))

# Adjusting for distance, and computing offsets with trend correction.
stuff <- correctedContact(data, average=FALSE, dist.correct=TRUE)
head(stuff$truth)
head(stuff$trend)
offsets <- log(stuff$bias[anchors(data, id=TRUE),]) +
    log(stuff$bias[targets(data, id=TRUE),]) +
    stuff$trend/log2(exp(1))
difference <- log(stuff$truth) - (log(counts(data)) - offsets)
stopifnot(all(is.na(difference) | difference < 1e-8))
```

---

cutGenome *Cut up the genome*

Description

Perform an in silico restriction digest of a target genome.

Usage

```
cutGenome(bs, pattern, overhang=4L)
```

Arguments

bs	a BSgenome object or a character string pointing to a FASTA file
pattern	character string describing the recognition site
overhang	integer scalar specifying the length of the 5' overhang

Details

This function simulates a restriction digestion of a specified genome, given the recognition site and 5' overhang of the cutter The total sequence spanned by each fragment is recorded, including the two sticky ends. No support is currently provided for searching the reverse strand, so the recognition site should be an inverse palindrome.

The genome should be specified as a BSgenome object. However, a character string can also be provided, specifying a FASTA file containing all the reference sequences in a genome. The latter may be necessary to synchronise the fragments with the genome used for alignment.

Note that some of the reported fragments may be physically impossible to form, e.g., for overlapping sites or consecutive sites when overhang==nchar(pattern). Nonetheless, they are still reported to maintain the correspondence between fragments and cut sites. Cleavage sites on the forward strand can be obtained as the start locations of all fragments (excepting the first fragment on each chromosome).

Value

A GRanges object containing the boundaries of each restriction fragment in the genome.

Warning

If bs is a FASTQ file, the chromosome names in the FASTQ headers will be loaded faithfully by cutGenome. However, many mapping pipelines will drop the rest of the name past the first whitespace when constructing the alignment index. To be safe, users should ensure that the chromosome names in the FASTQ headers consist of one word. Otherwise, there will be a discrepancy between the chromosome names in the output GRanges, and those in the BAM files after alignment.

Author(s)

Aaron Lun

See Also

matchPattern

Examples

```
require(BSgenome.Ecoli.NCBI.20080805)

cutGenome(Ecoli, "AAGCTT", overhang=4L) # HindIII
cutGenome(Ecoli, "CCGCGG", overhang=2L) # SacII
cutGenome(Ecoli, "AGCT", overhang=0L) # AluI

# Trying with FastA files.
x <- system.file("extdata", "fastaEx.fa", package="Biostrings")
cutGenome(x, "AGCT", overhang=2)
cutGenome(x, "AGCT", overhang=4)
```

---

diffHicUsersGuide *View diffHic user's guide*

Description

Finds the location of the user's guide and opens it for viewing.

Usage

```
diffHicUsersGuide(view=TRUE)
```

Arguments

view	logical scalar specifying whether the document should be opened

Details

The diffHic package is designed for the detection of differential interactions from Hi-C data. It provides methods for read pair counting, normalization, filtering and statistical analysis via edgeR. As the name suggests, the diffHic user's guide for can be obtained by running this function.

For non-Windows operating systems, the PDF viewer is taken from `Sys.getenv("R_PDFVIEWER")`. This can be changed to `x` by using `Sys.putenv(R_PDFVIEWER=x)`. For Windows, the default viewer will be selected to open the file.

Note that this guide is not a true vignette as it is not generated using `Sweave` when the package is built. This is due to the time-consuming nature of the code when run on realistic case studies.

Value

A character string giving the file location. If `view=TRUE`, the system's default PDF document reader is started and the user's guide is opened.

Author(s)

Aaron Lun

See Also

`system`

Examples

```
# To get the location:
diffHicUsersGuide(view=FALSE)
# To open in pdf viewer:
## Not run: diffHicUsersGuide()
```

DIList-class *DIList class and methods*

Description

Using the DIList class to store pairwise genomic interactions.

Details

Each DIList stores counts for pairwise genomic interactions. Slots are defined as:

`counts`: an integer matrix containing the number of read pairs for an interaction in each library

`colData`: a `DataFrame` object containing library-specific information in each row, e.g., the total number of read pairs

`exptData`: a `List` object containing data or parameters pertinent to counting

`anchors`: an integer vector specifying the index of the first interacting region

`targets`: an integer vector specifying the index of the second interacting region

`regions`: a `GRanges` object containing the coordinates of all interacting regions

Each row of `counts` corresponds to an interaction, while each column corresponds to a library. Each interaction is parameterized by an `anchors` and `targets` index, pointing to the anchor and target intervals in `regions`, respectively. The anchor interval is defined as that with the higher genomic start coordinate, compared to that of the target interval. This means that `anchors` is never less than `targets` to avoid redundant permutations.

Constructor

`DIList(counts, totals=colSums(counts), anchors, targets, regions, exptData=List(), ...)`: Returns a DIList object where each entry is used to fill the corresponding slots. Type coercion is performed as required. Arguments in `...` are used as columns in `colData`.

Accessors

In the code snippets below, `x` is a DIList object.

`anchors(x, id=FALSE)`: Get the `GRanges` corresponding to the anchor intervals for each interaction. If `id=TRUE`, indices to the corresponding intervals in `regions` are returned instead.

`targets(x, id=FALSE)`: Get the `GRanges` corresponding to the target intervals for each interaction. If `id=TRUE`, indices to the corresponding intervals in `regions` are returned instead.

`regions(x)`: Get the `GRanges` for all interacting regions.

`counts(x)`: Get the matrix of counts for all interactions in all libraries.

`colData(x)`: Get the `DataFrame` of library-specific information.

`exptData(x)`: Get the `List` of experiment-specific information.

`x$name`: Get the value of `colData(x)$name`.

`dim(x)`: Get the dimensions, i.e., number of interactions (rows) against number of libraries (columns).

`dimnames(x)`: Get the dimension names. This returns a list of length 2, where each element is `NULL` or a character vector.

Modifiers

In the code snippets below, `x` is a DIList object.

`x$name <- value`: Assign `value` to the `name` field of the `colData` in `x`. Primarily intended for modification of the library sizes in `totals`, but can also be used to store additional library-specific information.

`exptData(x) <- value`: Assign a `SimpleList` object named `value` to the `exptData` slot of `x`. Note that this also works with list accessors, e.g., by assigning to `exptData(x)$name`.

Subsetting and combining

In the code snippets below, x is a DIList object.

x[i, j]: Get count and coordinate data for all interactions i in libraries j. Either i or j can be missing, in which case all interactions or libraries are returned, respectively.

c(x, ..., add.totals=TRUE): Merge x with other DIList objects in ..., by concatenating the count matrices along with the anchor and target indices. By default, the totals will be added across all objects to be merged. This can be turned off by setting add.totals=FALSE, whereby only the totals of x are used (a warning will be generated if totals are not identical between objects). In all cases, objects to be merged should have the same value in the regions slot.

Other methods

In the code snippets below, x is a DIList object.

show(x): Shows a summary of the information in x. Specifically, a string is printed that reports the number of libraries, number of interactions and total number of regions in x.

as.matrix(x, first=NULL, second=first, fill=NULL, ...): Converts x into a contact matrix where rows and columns represent regions on the first and second chromosomes, respectively. Entries of the matrix correspond to an interaction of x. Each entry is extracted from fill, where fill is a numeric vector with one value per interaction in x. By default, the average abundance of each interaction in x is used as the entry. For intra-chromosomal spaces, both sides of the diagonal are filled. Multiple chromosomes can also be specified in first or second (all chromosomes are used by default). Each row and column is named according to the matching index of regions(x).

Author(s)

Aaron Lun

See Also

squareCounts

Examples

```
blah <- DIList(counts=matrix(c(1,1,2,2,3,3,4,4,5,5,6,6), ncol=2),
    totals=c(10L, 10L), anchors=c(1,2,3,4,5,6), targets=c(1,1,2,2,3,3),
regions=GRanges("chrA", IRanges(1:6*10, 1:6*10+9)))
nrow(blah)
ncol(blah)

blah
blah[1,]
blah[,2]
blah[1:2,2]

anchors(blah)
anchors(blah, id=TRUE)
targets(blah)
```

```
targets(blah, id=TRUE)
counts(blah)
regions(blah)

blah$totals
colData(blah)
exptData(blah)

blah$totals <- c(10L, 20L)
exptData(blah)$width <- 5
exptData(blah)

c(blah, blah)
c(blah[1:2,], blah[3:4,], add.totals=FALSE)

as.matrix(blah, anchor="chrA")
as.matrix(blah, fill=runif(nrow(blah)))
```

DIList-wrappers *Statistical wrappers for DIList objects*

Description

Convenience wrappers for statistical routines operating on DIList objects.

Usage

```
## S4 method for signature 'DIList'
normOffsets(object, lib.sizes, ...)
## S4 method for signature 'DIList'
normalize(object, lib.sizes, ...) # deprecated, use normOffsets
## S4 method for signature 'DIList'
asDGEList(object, lib.sizes, ...)
```

Arguments

object	a DIList object, like that produced by squareCounts
lib.sizes	an (optional) integer vector of library sizes
...	other arguments to be passed to the function being wrapped

Details

Counts are extracted from the matrix in the DIList object. If not specified in lib.sizes, library sizes are taken from the totals field in the column data of object. Warnings will be generated if this field is not present.

In the normOffsets method, the extracted counts and library sizes are supplied to normOffsets,matrix-method. Similarly, the asDGEList method wraps the DGEList constructor. In both cases, any arguments in ... are also passed to the wrapped functions.

Value

For normOffsets, either a numeric matrix or vector is returned; see normOffsets,matrix-method.

For asDGEList, a DGEList object is returned.

Author(s)

Aaron Lun

See Also

DGEList, normOffsets, squareCounts

Examples

```
blah <- DIList(counts=matrix(c(1,1,2,2,3,3,4,4,5,5,6,6), ncol=2),
    totals=c(10L, 10L), anchors=c(1,2,3,4,5,6), targets=c(1,1,2,2,3,3),
regions=GRanges("chrA", IRanges(10+1:20, 2+21:40)))

asDGEList(blah)
asDGEList(blah, lib.sizes=c(1,2)) # Manually set to some other value.
asDGEList(blah, group=c("a", "b"))

normOffsets(blah)
normOffsets(blah, lib.sizes=c(1,2)) # Manually set to some other value.
normOffsets(blah, logratioTrim=0)
normOffsets(blah, type="loess")

c(blah, blah)
c(blah[1:2,], blah[3:4,], add.totals=FALSE)
```

DNaseHiC *Methods for processing DNase Hi-C data*

Description

Processing of BAM files for DNase Hi-C into index files

Usage

```
segmentGenome(bs, size=500)
prepPseudoPairs(bam, param, file, dedup=TRUE, yield=1e7,
    ichim=TRUE, chim.span=1000, minq=NA)
```

Arguments

bs	a BSgenome object, or a character string pointing to a FASTA file, or a named integer vector of chromosome lengths
size	an integer scalar indicating the size of the pseudo-fragments
bam	a character string containing the path to a name-sorted BAM file
param	a pairParam object containing read extraction parameters
file	a character string specifying the path to an output index file
dedup	a logical scalar indicating whether marked duplicate reads should be removed
yield	a numeric scalar specifying the number of reads to extract at every iteration
ichim	a logical scalar indicating whether invalid chimeras should be counted
chim.span	an integer scalar specifying the maximum span between a chimeric 3' end and a mate read
minq	an integer scalar specifying the minimum mapping quality for each read

Details

DNase Hi-C involves random fragmentation with DNase instead of restriction enzymes. This is accommodated in diffHic by partitioning the genome into small pseudo-fragments, using segmentGenome. Reads are then assigned into these pseudo-fragments using prepPseudoPairs. The rest of the analysis pipeline can then be used in the same manner as that for standard Hi-C.

The behavior of prepPseudoPairs is almost identical to that for preparePairs, if the latter were asked to assign reads into pseudo-fragments. However, for prepPseudoPairs, no reporting or removal of self-circles or dangling ends is performed, as these have no meaning for artificial fragments. Also, invalidity of chimeras is determined by checking whether the 3' end is more than chim.span away from the mate read, rather than checking for localization in different fragments.

The size of the pseudo-fragments is determined by, well, size in segmentGenome. Smaller sizes provide better resolution but increase computational work. Needless to say, the param$fragments field should contain the output from segmentGenome, rather than from cutGenome. Also see cutGenome documentation for a warning about the chromosome names.

Some loss of spatial resolution is inevitable when reads are summarized into pseudo-fragments. This is largely irrelevant, though, as counting across the interaction space will ultimately use much larger bins (usually at least 2 kbp).

Value

For segmentGenome, a GRanges object is produced containing the coordinates of the pseudo-fragments in the specified genome.

For prepPseudoPairs, a HDF5-formatted index file is produced at the specified location. A list of diagnostic vectors are also returned in the same format as that from preparePairs, without the same.id entry.

Author(s)

Aaron Lun

See Also

preparePairs, cutGenome

Examples

```
require(BSgenome.Ecoli.NCBI.20080805)
segmentGenome(BSgenome.Ecoli.NCBI.20080805)
segmentGenome(BSgenome.Ecoli.NCBI.20080805, size=1000)

# Pretend that this example is DNase Hi-C.
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
pseudo <- segmentGenome(seqlengths(cuts), size=50)
param <- pairParam(pseudo)

tmpf <- "gunk.h5"
prepPseudoPairs(hic.file, param, tmpf)
prepPseudoPairs(hic.file, param, tmpf, dedup=FALSE)
prepPseudoPairs(hic.file, param, tmpf, minq=50)
prepPseudoPairs(hic.file, param, tmpf, chim.span=20)
```

---

domainDirections *Calculate domain directionality*

Description

Collect directionality statistics for domain identification with genomic bins.

Usage

```
domainDirections(files, param, width=50000, span=10)
```

Arguments

files	a character vector containing paths to the index files generated from each Hi-C library
param	a pairParam object containing read extraction parameters
width	an integer scalar specifying the width of each bin in base pairs
span	an integer scalar specifying the distance to consider for up/downstream interactions

Details

The genome is partitioned into bins of size `width`. For each bin, this function computes the total count for all bin pairs involving that bin and the `span` upstream bins. This is repeated for the `span` downstream bins. If multiple `files` are specified, the total of the averages across all libraries for each bin pair is computed instead.

The total up- and downstream counts are returned in the `Up` and `Down` fields of the output `GRanges` object. This can be used to compute a directionality statistic, e.g., as defined by Dixon et al, or by computing the log-fold change between fields. The latter is easier to compute and model with a Gaussian distribution. In any case, the directionalities can be used in a HMM to identify domains - see the user's guide for more details.

Value

A `GRanges` object with one entry for each bin in the genome. The `Up` and `Down` fields in the metadata contain the total average count for the up-/downstream interactions involving each bin.

Author(s)

Aaron Lun

References

Dixon JR et al. (2012). Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature* 485:376-380.

See Also

`squareCounts`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(fragments=cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, file=fout))

# Not really that informative; see user's guide.
out <- domainDirections(fout, param, width=10)
out
domainDirections(fout, param, width=10, span=1)

# Calculating directionality log-FC with a large prior.
dir.logFC <- log2((out$Up+10)/(out$Down+10))
dir.logFC

# Calculating directionality index with Dixon's method.
dixon.stat <- sign(out$Up-out$Down)*2*((out$Up-out$Down)/(out$Up+out$Down))^2
```

dixon.stat

---

enrichedPairs *Compute local enrichment for bin pairs*

Description

Calculate the log-fold increase in abundance for each bin pair against its local neighborhood.

Usage

```
enrichedPairs(data, flank=5, exclude=0, prior.count=2, abundances=NULL)
```

Arguments

data	a DIList object containing bin pair counts, generated by squareCounts
flank	an integer scalar, specifying the number of bins to consider as the local neighborhood
exclude	an integer scalar, specifying the number of bins to exclude from the neighborhood
prior.count	a numeric scalar indicating the prior count to use in computing the log-fold increase
abundances	a numeric vector of abundances for each bin pair

Value

A numeric vector containing the log-fold increase (i.e., enrichment value) for each bin pair in data.

Definition of the neighborhoods

Consider the coordinates of the interaction space in terms of bins, and focus on any particular bin pair (named here as the target bin pair). This target bin pair is characterized by four neighborhood regions, from A to D. Region A is a square with side lengths equal to flank*2+1, where the target bin pair is positioned in the center. Region B is a square with side lengths equal to flank, positioned such that the target bin pair lies at the corner furthest from the diagonal (only used for intra-chromosomal targets). Region C is a horizontal rectangle with dimensions (1, flank*2+1), containing the target bin pair at the center. Region D is the vertical counterpart to C.

Obviously, the target bin pair itself is excluded in the definition of each neighborhood. If exclude is positive, additional bin pairs closest to the target will also be excluded. For example, region A* is constructed with exclude instead of flank, and the resulting area is excluded from region A (and so on for all other regions). This avoids problems where diffuse interactions are imperfectly captured by the target bin pair, such that genuine interactions spill over into the neighborhood. Spill-over is undesirable as it will inflate the neighborhood abundance for genuine interactions. Setting a larger exclude ensures that this does not occur.

The size of flank requires consideration, as it defines the size of each neighborhood region. If the value is too large, other peaks may be included in the background such that the neighborhood abundance is inflated. On the other hand, if flank is too small, there will not be enough neighborhood bin pairs to dilute the increase in abundance from spill-over. Both scenarios result in a decrease in enrichment values and loss of power to detect punctate events. The default value of 5 seems to work well, though users may wish to test several values for themselves.

Computing the enrichment values

For a target bin pair in data, the enrichedPairs function computes the mean abundance for each of its surrounding neighborhoods. This is defined as the mean of the counts for all constituent bin pairs in that neighborhood (average counts are used for multiple libraries). The local background for the target bin pair is defined as the maximum of the mean abundances for all neighborhoods. The enrichment value is then defined as the the difference between the target bin pair's abundance and its local background. The idea is that bin pairs with high enrichments are likely to represent punctate interactions between clearly defined loci. Selecting for high enrichments can then select for these peak-like features in the interaction space.

The maximizing strategy is designed to mitigate the effects of structural features. Region B will capture the high interaction intensity within genomic domains like TADs, while the C and D will capture any bands in the interaction space. The abundance will be high for any neighborhood that captures a high-intensity feature, as the average counts will be large for all bin pairs within the features. This will then be chosen as the maximum during calculation of enrichment values. Otherwise, if only region A were used, the background abundance would be decreased by low-intensity bin pairs outside of the features. This results in spuriously high enrichment values for target bin pairs on the feature boundaries.

By default, nothing is done to adjust for the effect of distance on abundance for intra-chromosomal bin pairs. This is because the counts are generally too low to routinely fit a reliable trend. That said, users can still supply distance-adjusted abundances as abundances. Such values can be defined as the residuals of the fit from filterTrended. Obviously, no such work is required for inter-chromosomal bin pairs.

Author(s)

Aaron Lun

References

Rao S et al. (2014). A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*. 159, 1665-1690.

See Also

squareCounts, filterPeaks

Examples

```
# Setting up the object.
a <- 10
b <- 20
```

```
regions <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))

set.seed(23943)
all.anchors <- sample(length(regions), 50, replace=TRUE)
all.targets <- as.integer(runif(50, 1, all.anchors+1))
data <- DIList(matrix(rnbinom(200, mu=10, size=10), 50, 4), anchors=all.anchors,
    targets=all.targets, regions=regions, exptData=List(width=1))

# Getting peaks.
head(enrichedPairs(data))
head(enrichedPairs(data, flank=3))
head(enrichedPairs(data, flank=1))
head(enrichedPairs(data, exclude=1))

# Accounting for distance.
filtered <- filterTrended(data, prior.count=0)
adj.ab <- filtered$abundances - filtered$threshold
head(enrichedPairs(data, abundances=adj.ab))
```

`Filtering diagonals` *Filtering of diagonal bin pairs*

Description

Filtering to remove bin pairs on or near the diagonal of the interaction space.

Usage

```
filterDiag(data, by.dist=0, by.diag=0L, dist, ...)
```

Arguments

`data`	a `DIList` object produced by `squareCounts`
`by.dist`	a numeric scalar indicating the base-pair distance threshold below which bins are considered local
`by.diag`	an integer scalar indicating the bin distance threshold below which bins are considered local
`dist`	a optional numeric vector containing pre-computed distances
`...`	other arguments to pass to `getDistance`, if `dist` is not specified

Details

Pairs of the same bin will lie on the diagonal of the interaction space. Counts for these pairs can be affected by local artifacts (e.g., self-circles, dangling ends) that may not have been completely removed during earlier quality control steps. These pairs are also less interesting, as they capture highly local structure that may be the result of non-specific compaction. In many cases, these bin pairs are either removed or, at least, normalized separately within the analysis.

This function provides a convenience wrapper in order to separate diagonal bin pairs from those in the rest of the interaction space. Users can also consider near-diagonal bin pairs, which are defined as pairs of local bins on the linear genome. Specifically, bins are treated as local if they separated by less than `by.dist` in terms of base pairs, or by less than `by.diag` in terms of bins. These can be separated with the diagonal bin pairs if they are subject to the same issues described above.

Note that if `by.dist` is specified, it should be set to a value greater than 1.5 times the average bin size. Otherwise, the distance between the midpoints of adjacent bins will always be larger than `by.dist`, such that no near-diagonal bin pairs are removed.

Users can expedite processing by supplying a pre-computed vector of distances in `dist`. This vector may already be available if it was generated elsewhere in the pipeline. However, the supplied vector should have the same number of entries as that in `data`.

Value

A logical vector indicating whether each bin pair in `data` is a non-diagonal (or non-near-diagonal) element.

Author(s)

Aaron Lun

See Also

`getDistance`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(fragments=cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, file=fout))

# Collating to count combinations.
y <- squareCounts(fout, param, width=50, filter=1)

summary(filterDiag(y))
summary(filterDiag(y, by.dist=100))
summary(filterDiag(y, by.diag=1))
summary(filterDiag(y, dist=getDistance(y)))
```

`Filtering methods` *Filtering strategies for bin pairs*

Description

Implementations of the direct and trended filtering strategies for bin pair abundances.

Usage

```
filterDirect(data, prior.count=2, reference=NULL)
filterTrended(data, span=0.25, prior.count=2, reference=NULL)
```

Arguments

`data`	a `DIList` object produced by `squareCounts`
`span`	a numeric scalar specifying the bandwidth for loess curve fitting
`prior.count`	a numeric scalar indicating the prior count to use for calculating the average abundance
`reference`	another `DIList` object, usually containing data for larger bin pairs

Details

The `filterDirect` function implements the direct filtering strategy. The rate of non-specific ligation is estimated as the median of average abundances from inter-chromosomal bin pairs. This rate or some multiple thereof can be used as a minimum threshold for filtering, to keep only high-abundance bin pairs. When calculating the median, some finesse is required to consider empty parts of the interaction space, i.e., areas that are not represented by bin pairs.

The `filterTrended` function implements the trended filtering strategy. The rate of non-specific compaction is estimated by fitting a trend to the average abundances against the log-distance for all intra-chromosomal bin pairs. This rate can then be used as a minimum threshold for filtering. For inter-chromosomal bin pairs, the threshold is the same as that from the direct filter.

Curve fitting in `filterTrended` is done using `loessFit` with a bandwidth of `span`. Lower values may need to be used for a more accurate fit when the trend is highly non-linear. The bin size is also added to the distance prior to log-transformation, to avoid problems with undefined values when distances are equal to zero. Empty parts of the interaction space are considered by inferring the abundances and distances of the corresponding bin pairs (though this is skipped if too much of the space is empty).

If `reference` is specified, it will be used to compute filter thresholds instead of `data`. This is intended for large bin pairs that have been loaded with `filter=1`. Larger bins provide larger counts for more precise threshold estimates, while the lack of filtering ensures that estimates are not biased. All threshold estimates are adjusted to account for differences in bin sizes between `reference` and `data`. The final values can be used to directly filter on abundances in `data`; check out the user's guide for more details.

Value

A list is returned containing abundances, a numeric vector with the average abundances of all bin pairs in data. For filterDirect, the list contains a numeric scalar threshold, i.e., the non-specific ligation rate. For filterTrended, the list contains threshold, a numeric vector containing the threshold for each bin pair; and log.distance, a numeric vector with the log-distances for each bin pair.

If reference is specified in either function, an additional list named ref is also returned. This contains the filtering information for the bin pairs in reference, same as that reported above for each bin pair in data.

Author(s)

Aaron Lun

References

Lin, YC et al. (2012) Global changes in the nuclear positioning of genes and intra- and interdomain genomic interactions that orchestrate B cell fate. *Nat. Immunol.* 13. 1196-1204

See Also

squareCounts, scaledAverage

Examples

```
# Setting up the object.
a <- 10
b <- 20
regions <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))

set.seed(138153)
npairs <- 500
all.anchors <- sample(length(regions), npairs, replace=TRUE)
all.targets <- as.integer(runif(npairs, 1, all.anchors+1))
y <- DIList(matrix(rnbinom(npairs*4, mu=10, size=10), npairs, 4), anchors=all.anchors,
    targets=all.targets, regions=regions, exptData=List(width=1))

# Requiring at least 1.5-fold change.
direct <- filterDirect(y)
keep <- direct$abundances > direct$threshold + log2(1.5)
y[keep,]

# Requiring to be above the threshold.
trended <- filterTrended(y)
keep <- trended$abundances > trended$threshold
y[keep,]

# Running reference comparisons, using larger bin pairs.
w <- 5L
a2 <- a/w
b2 <- b/w
```

```
regions2 <- GRanges(rep(c("chrA", "chrB"), c(a2, b2)),
    IRanges(c(1:a2, 1:b2)*w-w+1L, c(1:a2, 1:b2)*w))
npairs2 <- 20
all.anchors2 <- sample(length(regions2), npairs2, replace=TRUE)
all.targets2 <- as.integer(runif(npairs2, 1, all.anchors2+1))
y2 <- DIList(matrix(rnbinom(npairs2*4, mu=10*w^2, size=10), npairs2, 4),
    anchors=all.anchors2, targets=all.targets2, regions=regions2,
    totals=y$totals, exptData=List(width=w))

direct2 <- filterDirect(y, reference=y2)
sum(direct2$abundances > direct2$threshold + log2(1.5))
trended2 <- filterTrended(y, reference=y2)
sum(trended2$abundances > trended2$threshold)
```

`filterPeaks` *Filter bin pairs for likely peaks*

Description

Identify bin pairs that are likely to represent punctate peaks in the interaction space.

Usage

```
filterPeaks(data, enrichment, min.enrich=log2(1.5), min.count=5, min.diag=2L, ...)
```

Arguments

`data`	a `DIList` object produced by `squareCounts` or `neighborCounts`
`enrichment`	a numeric vector of enrichment values, produced by `enrichedPairs` or `neighborCounts`
`min.enrich`	a numeric scalar indicating the minimum enrichment score for a peak
`min.count`	a numeric scalar indicating the minimum average count for a peak
`min.diag`	an integer scalar specifying the minimum diagonal in the interaction space with which to consider a peak
`...`	other arguments to be passed to `aveLogCPM` for the average count filter

Details

Filtering on the local enrichment scores identifies high-intensity islands in the interaction space. However, this alone is not sufficient to identify sensible peaks. Filtering on the absolute average counts prevents the calling of low-abundance bin pairs with high enrichment scores due to empty neighborhoods. Filtering on the diagonals prevents calling of high-abundance short-range interactions that are usually uninteresting. If either `min.count` or `min.diag` are `NULL`, no filtering will be performed on the average counts and diagonals, respectively.

Value

A logical vector indicating whether or not each bin pair is to be considered as a peak.

Author(s)

Aaron Lun

See Also

squareCounts, enrichedPairs, neighborCounts

Examples

```
# Setting up the object.
a <- 10
b <- 20
regions <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)))

set.seed(23943)
all.anchors <- sample(length(regions), 50, replace=TRUE)
all.targets <- as.integer(runif(50, 1, all.anchors+1))
data <- DIList(matrix(rnbinom(200, mu=10, size=10), 50, 4), anchors=all.anchors,
targets=all.targets, regions=regions, exptData=List(width=1))

# Getting peaks.
enrichment <- enrichedPairs(data)
summary(filterPeaks(data, enrichment, min.enrich=0.5))
summary(filterPeaks(data, enrichment, min.enrich=0.5, min.count=10))
summary(filterPeaks(data, enrichment, min.enrich=0.5, min.diag=NULL))
```

---

getArea *Get interaction area*

Description

Compute area in the interaction space for each pair of regions.

Usage

```
getArea(data, bp=TRUE)
```

Arguments

data	a DIList object
bp	a logical scalar indicating whether areas should be reported in base-pair terms

Details

The getArea function returns the area in the interaction space for each pair of regions. If bp=TRUE, the area is reported in terms of squared base pairs. This tends to be the easiest to interpret. Otherwise, the area is reported as the number of pairs of restriction fragments. This may be more relevant to the actual resolution of the Hi-C experiment.

Some special consideration is required for areas overlapping the diagonal. This is because counting is only performed on one side of the diagonal, to avoid redundancy. Base-pair areas are automatically adjusted to account for this feature, based on the presence of partial overlaps between interacting regions.

For fragment-based areas, some additional work is required to properly compute areas around the diagonal for partially overlapping regions. This is only necessary when data is produced by connectCounts. This is because bins will not partially overlap in any significant manner when counts are generated with squareCounts.

Value

A numeric vector is returned containing the area in the interaction space for each pair of regions in data.

Author(s)

Aaron Lun

See Also

squareCounts, connectCounts

Examples

```
# Making up a DIList for binned data.
nfrags <- 50
frag.sizes <- as.integer(runif(nfrags, 5, 10))
ends <- cumsum(frag.sizes)
cuts <- GRanges("chrA", IRanges(c(1, ends[-nfrags]+1), ends))
param <- pairParam(cuts)

regions <- diffHic:::.getBinID(cuts, 20)$region
all.combos <- combn(length(regions), 2)
y <- DIList(matrix(0, ncol(all.combos), 1), anchors=all.combos[2,],
    targets=all.combos[1,], regions=regions, exptData=List(param=param, width=20))

# Generating partially overlapping regions.
set.seed(3424)
re <- sample(nfrags, 20)
rs <- as.integer(runif(20, 1, re+1))
regions <- GRanges("chrA", IRanges(start(cuts)[rs], end(cuts)[re]))
regions$nfrags <- re - rs + 1L
regions <- sort(regions)
all.combos <- combn(length(regions), 2)
y2 <- DIList(matrix(0, ncol(all.combos), 1), anchors=all.combos[2,],
```

```
    targets=all.combos[1,], regions=regions, exptData=List(param=param))

#### Getting areas. ####

getArea(y)
getArea(y, bp=FALSE)

getArea(y2)
getArea(y2, bp=FALSE)
```

---

getDistance *Get the linear distance for each interaction*

Description

Compute the distance between interacting regions on the linear genome.

Usage

```
getDistance(data, type=c("mid", "gap", "span"))
```

Arguments

data	a DIList object
type	a character string specifying the type of distance to compute

Details

The getDistance function examines each interaction in data$pairs and computes the distance between the corresponding regions on the linear genome. Options to compute the distances are mid, for the distance between the midpoints of the regions; gap, for the distance of the interval lying between the regions (set to zero, if regions are overlapping); and span, for the total distance spanned by the interaction, including both regions. Interchromosomal interactions are marked with NA.

Value

An integer vector is returned containing the distances between interacting regions for each pair in data.

Author(s)

Aaron Lun

See Also

squareCounts, connectCounts

Examples

```
# Making up a DIList.
nregs <- 20
reg.sizes <- as.integer(runif(nregs, 5, 10))
ends <- cumsum(reg.sizes)
regions <- GRanges("chrA", IRanges(c(1, ends[-nregs]+1), ends))

all.combos <- combn(length(regions), 2)
y <- DIList(matrix(0, ncol(all.combos), 1), anchors=all.combos[2,],
targets=all.combos[1,], regions=regions, exptData=List(width=1))

# Collating to count combinations.
getDistance(y)
getDistance(y, type="gap")
getDistance(y, type="span")
```

---

getPairData *Get read pair data*

Description

Extract diagnostics for each read pair from an index file

Usage

```
getPairData(file, param)
```

Arguments

`file`	character string, specifying the path to the index file produced by `preparePairs`
`param`	a `pairParam` object containing read extraction parameters

Details

This is a convenience function to extract read pair diagnostics from an index file, generated from a Hi-C library with `preparePairs`. The aim is to examine the distribution of each returned value to determine the appropriate cutoffs for `prunePairs`.

The `length` refers to the length of the DNA fragment used in sequencing. It is computed for each read pair by adding the distance of each read to the closest restriction site in the direction of the read.

The `insert` simply refers to the insert size for each read pair. This is defined as the distance between the extremes of each read on the same chromosome. Values for interchromosomal pairs are set to `NA`.

For `orientation`, setting 0x1 or 0x2 means that the reads designated as the "anchor" or "target" respectively are on the reverse strand. For intrachromosomal reads, an orientation value of 1 represents inward-facing reads whereas a value of 2 represents outward-facing reads.

Note that a `pairParam` object is only used here for consistency. Specifically, the restriction fragment coordinates in `param$fragments` are required. No removal of read pairs will be performed here, so the values of `param$restrict` or `param$discard` will be ignored.

Value

A dataframe is returned containing integer fields for `length`, `orientation` and `insert` for each read pair.

Author(s)

Aaron Lun

See Also

`preparePairs`, `prunePairs`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

tmpf <- "gunk.h5"
invisible(preparePairs(hic.file, param, tmpf))
getPairData(tmpf, param)
```

---

loadData *Load data from an index file*

Description

Load read pair data and chromosome names from a HDF5 index file.

Usage

```
loadChromos(file)
loadData(file, anchor, target)
```

Arguments

`file`	a character string containing a path to a index file
`anchor`	a character string, specifying the name of one chromosome in a pair
`target`	a character string, specifying the name of the other chromosome in the pair

Details

The purpose of these function is to allow users to perform custom analyses by extracting the data manually from each index file. This may be desirable, e.g., when preparing data for input into other tools. To extract all data, users are advised to run `loadData` iteratively on each pair of chromosomes as obtained with `loadChromos`.

Note that `loadData` will successfully operate even if the anchor/target specification is mixed up. In this case, it will return a warning to inform the user that the names should be switched.

Value

The `loadChromos` function will return a dataframe with character fields `anchors` and `targets`. Each row represents a pair of chromosomes, the names of which are stored in the fields. The presence of a row indicates that the data for the corresponding pair exists in the `file`.

The `loadData` function will return a dataframe where each row contains information for one read pair. Refer to `preparePairs` for more details on the type of fields that are included.

Author(s)

Aaron Lun

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

tmpf <- "gunk.h5"
preparePairs(hic.file, param, tmpf)

loadChromos(tmpf)
loadData(tmpf, "chrA", "chrA")
loadData(tmpf, "chrB", "chrA")
loadData(tmpf, "chrA", "chrB")
try(loadData(tmpf, "chrA2", "chrB2"))
```

---

marginCounts *Collect marginal counts for each bin*

Description

Count the number of read pairs mapped to each bin across multiple Hi-C libraries.

Usage

```
marginCounts(files, param, width=50000)
```

Arguments

files	a character vector containing paths to the index files
param	a pairParam object containing read extraction parameters
width	an integer scalar specifying the width of each bin

Details

The genome is first split into non-overlapping adjacent bins of size width, which are rounded to the nearest restriction site. The marginal count for each bin is defined as the number of reads in the library mapped to the bin. This acts as a proxy for genomic coverage by treating Hi-C data as single-end.

Even though counts do not correspond to interactions, a DIList object is still used to store the output for convenience. Bin coordinates can be extracted as the anchor or target intervals in the output. Each row of the output refers to a single bin in the linear genome, instead of a bin pair in the interaction space.

Larger marginal counts can be collected by increasing the width value. However, this comes at the cost of spatial resolution as adjacent events in the same bin can no longer be distinguished. Note that filtering is automatically performed to remove empty bins.

Counting will consider the values of restrict, discard and cap in param. See pairParam for more details.

Value

A DIList object containing the marginal counts for each bin. Anchor and target regions are guaranteed to be identical.

Author(s)

Aaron Lun

See Also

squareCounts, DIList-class

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(fragments=cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, fout))

# Collating to count combinations.
mar <- marginCounts(fout, param, width=10)
head(counts(mar))
mar <- marginCounts(fout, param, width=50)
head(counts(mar))
```

```
mar <- marginCounts(fout, param, width=100)
head(counts(mar))

# Attempting with other parameters.
mar <- marginCounts(fout, reform(param, restrict="chrA"), width=50)
head(counts(mar))
mar <- marginCounts(fout, reform(param, cap=1), width=50)
head(counts(mar))
mar <- marginCounts(fout, reform(param, discard=GRanges("chrA", IRanges(1, 50))), width=50)
head(counts(mar))
```

`mergePairs` *Merge read pairs*

Description

Merge index files for multiple Hi-C libraries into a single output file.

Usage

```
mergePairs(files, file.out)
```

Arguments

`files`	a character vector containing the paths to the index files to be merged
`file.out`	a character string specifying the path to the output index file

Details

Hi-C libraries are often split into technical replicates. This function facilitates the merging of said replicates into a single library for downstream processing. Index files listed in `files` should be produced by `preparePairs`, with or without pruning by `prunePairs`.

Value

A merged index file is produced at the specified location. A `NULL` object is invisibly returned.

Author(s)

Aaron Lun

See Also

`preparePairs`, `prunePairs`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <-readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

fout <- "temp.h5"
fout2 <- "temp2.h5"
fout3 <- "temp3.h5"
invisible(preparePairs(hic.file, param, fout))
invisible(prunePairs(fout, param, fout2))
invisible(prunePairs(fout, param, fout3, max.frag=50))

mout <- "merged"
mergePairs(c(fout2, fout3), mout)
require(rhdf5)
h5read(fout2, "chrA/chrA")
h5read(fout3, "chrA/chrA")
h5read(mout, "chrA/chrA")
```

---

neighborCounts *Load Hi-C interaction counts*

Description

Collate count combinations for interactions between pairs of bins across multiple Hi-C libraries.

Usage

```
neighborCounts(files, param, width=50000, filter=1L, flank=NULL,
    exclude=NULL, prior.count=NULL)
```

Arguments

files	a character vector containing paths to the index files generated from each Hi-C library
param	a pairParam object containing read extraction parameters
width	an integer scalar specifying the width of each square in base pairs
filter	an integer scalar specifying the minimum count for each square
flank	an integer scalar, specifying the number of bins to consider as the local neighborhood
exclude	an integer scalar, specifying the number of bins to exclude from the neighborhood
prior.count	a numeric scalar indicating the prior count to use in computing the log-fold increase

Details

This function combines the functionality of squareCounts and enrichedPairs. The idea is to allow calculation of local enrichment values when there is insufficient memory to load all bin pairs with filter=1L in squareCounts. Here, the interaction space around each bin pair is examined as the counts are loaded for that bin pair, avoiding the need to hold the entire interaction space at once. Only the counts and local enrichment values for those bin pairs with row sums above filter are reported to save memory. The returned enrichment values are equivalent to that computed with enrichedPairs with the default settings.

Value

A list object is returned containing interaction, a DIList object with the number of read pairs for each bin pair across all libraries; and enrichment, a numeric vector of enrichment statistics for each bin pair in interaction.

Author(s)

Aaron Lun

References

Rao S et al. (2014). A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*. 159, 1665-1690.

See Also

squareCounts, enrichedPairs

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(fragments=cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, file=fout))

# Collating to count combinations.
y <- neighborCounts(fout, param, width=50, filter=2,
    flank=5, prior.count=2)
y$interaction
y$enrichment

# Practically identical to a more memory-intensive call.
ref <- squareCounts(fout, param, width=50, filter=1)
keep <- rowSums(counts(ref)) >= 2
enriched <- enrichedPairs(ref, flank=5, prior.count=2)

stopifnot(identical(ref[keep,], y$interaction))
stopifnot(all(abs(enriched[keep] - y$enrichment) < 1e-6))
```

normalizeCNV *Normalize CNV biases*

Description

Compute normalization offsets to remove CNV-driven and abundance-dependent biases

Usage

```
normalizeCNV(data, margins, prior.count=3, span=0.3, maxk=500, ...)
matchMargins(data, margins)
```

Arguments

data	a DIList object produced by squareCounts
margins	a DIList object produced by marginCounts
prior.count	a numeric scalar specifying the prior count to use in computing marginal log-ratios
span	a numeric scalar between 0 and 1, describing the span of the fit
maxk	a integer scalar specifying the number of vertices to use during local fitting
...	other arguments to pass to locfit

Details

Each bin pair in data is associated with three covariates. The first two are the marginal log-ratios of the corresponding bins, i.e., the log-ratio of the marginal counts between two libraries. These represent the relative CNVs in the interacting regions between libraries. To avoid redundancy, the first covariate is the larger marginal log-ratio whereas the second is the smaller. The third covariate is the average abundance across all libraries.

Each bin pair is also associated with a response, i.e., the log-ratio of the interaction counts between two libraries. A loess-like surface is fitted to the response against the three covariates, using the locfit function. The aim is to eliminate systematic differences between libraries at any combination of covariate values. This removes CNV-driven biases as well as trended biases with respect to the abundance. The fitted value can then be used as a GLM offset for each bin pair.

The DIList objects in data and margins should be constructed with the same parameters in their respective functions. This ensures that the regions are the same, so that the marginal counts can be directly used. Matching of the bins in each bin pair in data to indices of margins is performed using matchMargins. Note that the marginal counts are not directly computed from data as filtering of bin pairs may be performed beforehand.

In practice, normalization offsets are computed for each library relative to a single reference "average" library. This average library is constructed by using the average abundance as the (log-)count for both the bin pair and marginal counts. The space of all pairs of CNV log-ratios is also rotated

by 45 degrees prior to smoothing. This improves the performance of the approximations used by locfit.

The fit parameters can be changed by varying span, maxk and additional arguments in locfit. Higher values of span will increase smoothness, at the cost of sensitivity. Increases in maxk may be required to obtain a more accurate approximation when fitting large datasets. In all cases, a loess fit of degree 1 is used.

Value

For normalizeCNV, a numeric matrix is returned with the same dimensions as counts(data). This contains log-based GLM offsets for each bin pair in each library.

For matchMargins, a data frame is returned with integer fields amatch and tmatch. Each field specifies the index in margins corresponding to the anchor or target bin for each bin pair in data.

Author(s)

Aaron Lun

See Also

locfit, lp, squareCounts, marginCounts

Examples

```
# Dummying up some data.
set.seed(3423746)
npts <- 100
npairs <- 5000
nlibs <- 4
anchors <- sample(npts, npairs, replace=TRUE)
targets <- sample(npts, npairs, replace=TRUE)
data <- DIList(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs),
totals=runif(nlibs, 1e6, 2e6), anchors=pmax(anchors, targets), targets=pmin(anchors, targets),
regions=GRanges("chrA", IRanges(1:npts, 1:npts)))
margins <- DIList(counts=matrix(rpois(npts*nlibs, 100), nrow=npts),
totals=data$totals, anchors=1:npts, targets=1:npts, regions=regions(data))

# Running normalizeCNV.
head(normalizeCNV(data, margins))
head(normalizeCNV(data, margins, prior.count=1))
head(normalizeCNV(data, margins, span=0.5))

# Occasionally locfit will complain; increase maxk to compensate.
data <- DIList(counts=matrix(rpois(npairs*nlibs, 20), nrow=npairs),
totals=runif(nlibs, 1e6, 2e6), anchors=pmax(anchors, targets), targets=pmin(anchors, targets),
regions=GRanges("chrA", IRanges(1:npts, 1:npts)))
tryCatch(head(normalizeCNV(data, margins, maxk=100)), error=function(e) e)
head(normalizeCNV(data, margins, maxk=1000))

# Matching margins.
matched <- matchMargins(data, margins)
```

```
head(matched)
anchor.counts <- margins[matched$amatch,]
target.counts <- margins[matched$tmatch,]
```

---

pairParam *pairParam class and methods*

Description

Class to specify read pair loading parameters

Details

Each pairParam object stores a number of parameters to extract reads from a BAM file. Slots are defined as:

`fragments`: a `GRanges` object containing the coordinates of the restriction fragments

`restrict`: a character vector or 1-by-2 matrix containing the names of allowable chromosomes from which reads will be extracted

`discard`: a `GRanges` object containing intervals in which any alignments will be discarded

`cap`: an integer scalar, specifying the maximum number of read pairs per pair of restriction fragments

The `fragments` object defines the genomic interval spanned by each restriction fragment. All reads are generated around restriction sites, so the spatial resolution of the experiment depends on such sites. The object can be can be obtained by applying `cutGenome` on an appropriate `BSgenome` object.

If `restrict` is supplied, reads will only be extracted for the specified chromosomes. This is useful to restrict the analysis to interesting chromosomes, e.g., no contigs/scaffolds or mitochondria. `restrict` can also be a `n`-by-2 matrix, specifying `n` pairs of chromosomes over which read pairs are to be counted.

If `discard` is set, a read will be removed if the corresponding alignment is wholly contained within the supplied ranges. Any pairs involving reads discarded in this manner will be ignored. This is useful for removing unreliable alignments in repeat regions.

If `cap` is set to a non-`NA` value, an upper bound will be placed on the number of read pairs that are counted for each fragment pair (after any removal due to `discard`). This protects against spikes in the read pair density throughout the interaction space. Such spikes may be caused by technical artifacts like PCR duplication or repeats, which were not successfully removed in prior processing steps.

Constructor

`pairParam(fragments, discard=GRanges(), restrict=NULL, cap=NA)`: Creates a pairParam object. Each argument is placed in the corresponding slot, with coercion into the appropriate type.

Subsetting

In the code snippets below, x is a pairParam object.

`x$name`: Returns the value in slot `name`.

Other methods

In the code snippets below, x is a pairParam object.

`show(x)`: Describes the parameter settings in plain English.

`reform(x, ...)`: Creates a new pairParam object, based on the existing `x`. Any named arguments in `...` are used to modify the values of the slots in the new object, with type coercion as necessary.

Author(s)

Aaron Lun

See Also

`cutGenome`, `squareCounts`

Examples

```
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))

blah <- pairParam(cuts)
blah <- pairParam(cuts, discard=GRanges("chrA", IRanges(1, 10)))
blah <- pairParam(cuts, restrict='chr2')
blah$fragments
blah$restrict
blah$cap

# Use 'reform' if only some arguments need to be changed.
blah
reform(blah, restrict='chr3')
reform(blah, discard=GRanges())
reform(blah, cap=10)

# Different restrict options.
pairParam(cuts, restrict=c('chr2', 'chr3'))
pairParam(cuts, restrict=cbind('chr2', 'chr3'))
pairParam(cuts, restrict=cbind(c('chr1', 'chr2'), c('chr3', 'chr4')))
```

plotDI *Construct a plaid plot of differential interactions*

Description

Plot differential interactions in a plaid format with informative coloring.

Usage

```
plotDI(data, fc, first.region, second.region=first.region,
    col.up="red", col.down="blue", background="grey70",
    zlim=NULL, xlab=NULL, ylab=NULL, diag=TRUE, ...)
rotDI(data, fc, region, col.up="red", col.down="blue",
    background="grey70", zlim=NULL, xlab=NULL, ylab="Gap", ...)
```

Arguments

data	a DIList object
fc	a numeric vector of log-fold changes
first.region	a GRanges object of length 1 specifying the first region
second.region	a GRanges object of length 1 specifying the second region
region	a GRanges object of length 1 specifying the region of interest
col.up	any type of R color to describe the maximum color for positive log-fold changes
col.down	any type of R color to describe the maximum color for negative log-fold changes
background	any type of R color, specifying the background color of the interaction space
zlim	a numeric scalar indicating the maximum absolute log-fold change
xlab	character string for the x-axis label on the plot, defaults to the first chromosome name
ylab	character string for the y-axis label on the plot, defaults to the second chromosome name in plotDI
diag	a logical scalar specifying whether boxes should be shown above the diagonal for intra-chromosomal plots in plotDI
...	other named arguments to be passed to plot

Details

The plotDI function constructs a plaid plot on the current graphics device. The intervals of first.region and second.region are represented by the x- and y-axes, respectively. Each bin pair is represented by a box in the plotting space, where each side of the box represents a bin. Plotting space that is not covered by any bin pair is shown in background.

The color of the box depends on the magnitude and sign of the log-fold change in fc. Positive log-FCs will range from white to col.up, whereas negative log-FCs will range from white to col.down.

The chosen color is proportional to the magnitude of the log-FC, and the most extreme colors are only obtained at the maximum absolute log-FC in `fc`. The maximum value can be capped at `zlim` for better resolution of small log-FCs.

If `diag=TRUE`, boxes will also be plotted above the diagonal for intra-chromosomal plots. This is set as the default to avoid confusion when `first.region` is not set as the anchor range, i.e., it has a lower sorting order than `second.region`. However, this can also be turned off to reduce redundancy in visualization around the diagonal.

The `rotDI` function constructs a rotated plot of differential interactions, for visualization of local changes. See `rotPlaid` for more details.

Value

A (rotated) plaid plot of differential interactions is produced on the current graphics device. A function is also invisibly returned that converts log-FCs into colors. This is useful for coordinating the colors, e.g., when constructing a separate color bar.

Author(s)

Aaron Lun

References

Lieberman-Aiden E et al. (2009). Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. *Science* 326, 289-293.

See Also

`plotPlaid`, `rotPlaid`, `squareCounts`

Examples

```
# Setting up the objects.
a <- 10
b <- 20
regions <- GRanges(rep(c("chrA", "chrB"), c(a, b)), IRanges(c(1:a, 1:b), c(1:a, 1:b)),
    seqinfo=Seqinfo(seqlengths=c(chrA=a, chrB=b), seqnames=c("chrA", "chrB")))

set.seed(3423)
all.anchors <- sample(length(regions), 500, replace=TRUE)
all.targets <- as.integer(runif(500, 1, all.anchors+1))
out <- DIList(matrix(0, 500, 1), anchors=all.anchors, targets=all.targets,
    regions=regions, exptData=List(width=1))
fc <- runif(nrow(out), -2, 2)

# Constructing intra-chromosomal DI plots around various regions
plotDI(out, fc, first.region=GRanges("chrA", IRanges(1, 10)),
    second.region=GRanges("chrA", IRanges(1, 10)), diag=TRUE)
plotDI(out, fc, first.region=GRanges("chrA", IRanges(1, 10)),
    second.region=GRanges("chrA", IRanges(1, 10)), diag=FALSE)

# Constructing inter-chromosomal DI plots around various regions
```

```
xxx <- plotDI(out, fc, first.region=GRanges("chrB", IRanges(1, 10)),
    second.region=GRanges("chrA", IRanges(1, 20)), diag=TRUE)
plotDI(out, fc, first.region=GRanges("chrB", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), diag=TRUE, zlim=5)

# Making colorbars.
xxx((-10):10/10)
xxx((-20):20/20)

# Rotated.
rotDI(out, fc, region=GRanges("chrA", IRanges(1, 200)))
rotDI(out, fc, region=GRanges("chrB", IRanges(1, 200)))
```

`plotPlaid` *Construct a plaid plot of interactions*

Description

Plot interactions between two sequences in a plaid format with informative coloring.

Usage

```
plotPlaid(file, param, first.region, second.region=first.region,
    width=10000, col="black", max.count=20, xlab=NULL, ylab=NULL,
diag=TRUE, count=FALSE, count.args=list(), ...)
rotPlaid(file, param, region, width=10000, col="black",
    max.count=20, xlab=NULL, ylab="Gap", ...)
```

Arguments

`file`	character string specifying the path to an index file produced by `prepairPairs`
`param`	a `pairParam` object containing read extraction parameters
`first.region`	a `GRanges` object of length 1 specifying the first region
`second.region`	a `GRanges` object of length 1 specifying the second region
`region`	a `GRanges` object of length 1 specifying the region of interest
`width`	an integer scalar specifying the width of each bin in base pairs
`col`	any type of R color to describe the color of the plot elements
`max.count`	a numeric scalar specifying the count for which the darkest color is obtained
`xlab`	character string for the x-axis label on the plot, defaults to the chromosome name of `first.region`
`ylab`	character string for the y-axis label on the plot, defaults to the chromosome name of `second.regeion` in `plotPlaid`
`diag`	a logical scalar specifying whether boxes should be shown above the diagonal for intra-chromosomal plots in `plotPlaid`

count	a logical scalar specifying whether the count for each bin should be plotted in plotPlaid
count.args	a named list of arguments to be passed to text for plotting of bin counts, if count=TRUE
...	other named arguments to be passed to plot

Details

The plotPlaid function constructs a plaid plot on the current graphics device. The intervals of the first.region and second.region are represented by the x- and y-axes, respectively. Each region is partitioned into bins of size width. Each bin pair is represented by a box in the plotting space, where each side of the box represents a bin. The color of the box depends on the number of read pairs mapped between the corresponding bins.

The resolution of colors can be controlled by varying max.count. All boxes with counts above max.count will be assigned the maximum intensity. Other boxes will be assigned a color of intensity proportional to the size of the count, such that a count of zero results in white space. Smaller values of max.count will improve contrast at low counts at the cost of contrast at higher counts. Scaling max.count is recommended for valid comparisons between libraries of different sizes (e.g., larger max.count for larger libraries).

If count=TRUE, the number of read pairs will be shown on top of each bin. This will be slower to plot but can be useful in some cases, e.g., when more detail is required, or when the range of colors is not sufficient to capture the range of counts in the data. If diag=TRUE, boxes will also be plotted above the diagonal for intra-chromosomal plots. This is set as the default to avoid confusion when first.region is not set as the anchor range, i.e., it has a lower sorting order than second.region. However, this can also be turned off to reduce redundancy in visualization around the diagonal.

The rotPlaid function constructs a plaid plot that has been rotated by 45 degrees. This is useful for visualizing local interactions within a specified region. In a rotated plot, the x-coordinate of a box in the plotting space represents the midpoint between two interacting bins, while the y-coordinate represents the distance between bins. More simply, the interacting bins of a box can be identified by tracing diagonals from the edges of the box to the x-axis.

Note that the plotted boxes for the bin pairs may overwrite the bounding box of the plot. This can be fixed by running box() after each plotPlaid call.

Value

A (rotated) plaid plot is produced on the current graphics device. For both functions, a function is invisibly returned that converts counts into colors. This is useful for coordinating the colors, e.g., when constructing a separate color bar.

Author(s)

Aaron Lun

References

Lieberman-Aiden E et al. (2009). Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. *Science* 326, 289-293.

See Also

preparePairs

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
originals <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(originals)

# Setting up parameters
fout <- "temp_saved.h5"
invisible(preparePairs(hic.file, param, fout))

# Constructing intra-chromosomal plaid plots around various regions.
plotPlaid(fout, param, first.region=GRanges("chrA", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=50, diag=TRUE)
box()
xxx <- plotPlaid(fout, param, first.region=GRanges("chrA", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=50, diag=FALSE)

# Making colorbars.
xxx(1:2)
xxx(1:5)
xxx(1:10)

# Constructing inter-chromosomal plaid plots around various regions
plotPlaid(fout, param, first.region=GRanges("chrB", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=50)
plotPlaid(fout, param, first.region=GRanges("chrB", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=100)

# For a hypothetical second library which is half the size of the previous one:
plotPlaid(fout, param, first.region=GRanges("chrB", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=100, max.count=20, count=TRUE)
plotPlaid(fout, param, first.region=GRanges("chrB", IRanges(1, 100)),
    second.region=GRanges("chrA", IRanges(1, 200)), width=100, max.count=40,
    count=TRUE, count.args=list(col="blue"))

# Rotated
rotPlaid(fout, param, region=GRanges("chrA", IRanges(1, 200)), width=50)
rotPlaid(fout, param, region=GRanges("chrA", IRanges(1, 200)), width=100)
```

---

preparePairs *Prepare Hi-C pairs*

Description

Identifies the interacting pair of restriction fragments corresponding to each read pair in a Hi-C library.

Usage

```
preparePairs(bam, param, file, dedup=TRUE, minq=NA, yield=1e7, ichim=TRUE, chim.dist=NA)
```

Arguments

bam	a character string containing the path to a name-sorted BAM file
param	a pairParam object containing read extraction parameters
file	a character string specifying the path to an output index file
dedup	a logical scalar indicating whether marked duplicate reads should be removed
minq	an integer scalar specifying the minimum mapping quality for each read
yield	a numeric scalar specifying the number of reads to extract at every iteration
ichim	a logical scalar indicating whether invalid chimeras should be counted
chim.dist	an integer scalar specifying the maximum distance between segments for a valid chimeric read pair

Value

Multiple dataframe objects are stored within the specified file using the HDF5 format. Each object corresponds to a pair of anchor/target chromosomes. Each row of the dataframe contains information for a read pair, with one read mapped to each chromosome. The dataframe contains several integer fields:

anchor.id: index of the anchor restriction fragment

target.id: index of the target restriction fragment

anchor.pos: mapping coordinate of the read (or for chimeras, the 5' segment thereof) on the anchor fragment

target.pos: mapping coordinate of read on the target fragment

anchor.len: length of the alignment on the anchor fragment, set to a negative value if the alignment is on the reverse strand

target.len: length of the alignment on the target fragment, set to a negative value if reverse stranded

See getPairData for more details on length, orientation and gap.

An integer vector is also returned from the function, containing various diagnostics:

pairs: an integer vector containing total, the total number of read pairs; marked, read pairs with at least one marked 5' end; filtered, read pairs where the MAPQ score for either 5' end is below minq; mapped, read pairs considered as successfully mapped (i.e., not filtered, and also not marked if dedup=TRUE)

same.id: an integer vector containing dangling, the number of read pairs that are dangling ends; and self.circles, the number of read pairs forming self-circles

singles: an integer scalar specifying the number of reads without a mate

chimeras: an integer vector containing total, the total number of read pairs with one chimeric read; mapped, chimeric read pairs with both 5' ends mapped; multi, mapped chimeric pairs with at least one successfully mapped 3' segment; and invalid, read pairs where the 3' location of one read disagrees with the 5' location of the mate

Converting to restriction fragment indices

The resolution of a Hi-C experiment is defined by the distribution of restriction sites across the genome. Thus, it makes sense to describe interactions in terms of restriction fragments. This function identifies the interacting fragments corresponding to each pair of reads in a Hi-C library. To save space, it stores the indices of the interacting fragments for each read pair, rather than the fragments themselves.

Indexing is performed by matching up the mapping coordinates for each read with the restriction fragment boundaries in `param$fragments`. Needless to say, the boundary coordinates in `param$fragments` must correspond to the reference genome being used. In most cases, these can be generated using the `cutGenome` function from any given `BSgenome` object. If, for any reason, a modified genome is used for alignment, then the coordinates of the restriction fragments on the modified genome are required.

Each read pair subsequently becomes associated with a pair of restriction fragments. The anchor fragment is that with the higher genomic coordinate, i.e., the larger index in `param$fragments`. The target fragment is that with the smaller coordinate/index. This definition avoids the need to consider both permutations of indices in a pair during downstream processing.

Handling of read pairs

For pairs with chimeric reads, the alignment of the 5' end of the read is used to identify the interacting fragments. Invalid chimeras arise when the position of the 3' segment of a chimeric read is not consistent with that of the mate read. Invalid chimeric pairs are generally indicative of mapping errors but can also form due to non-specific ligation events. While they can be explicitly removed, setting `ichim=TRUE` is recommended to avoid excessive filtering of reads when alignment of short chimeric segments is inaccurate.

By default, invalid chimeras are defined as all pairs where the 3' segment and the mate do not map onto the same restriction fragment in an inward-facing orientation. Alternatively, a distance-based threshold can be used by setting `chim.dist` to some reasonable value, e.g., 1000 bp. Chimeras are only considered invalid if the distance between the segment and mate is greater than `chim.dist` (or if the alignments are not inward-facing). This may be more relevant in situations involving inefficient cleavage, where the mapping locations are broadly consistent but do not fall in the same restriction fragment.

Self-circles are outward-facing read pairs mapped to the same restriction fragment. These are formed from inefficient cross-linking and are generally uninformative. Dangling ends are inward-facing read pairs mapped to the same fragment, and are generated from incomplete ligation of blunt ends. Both constructs are detected and discarded within the function. Note that this does not consider dangling ends or self-circles formed from incompletely digested fragments, which must be removed with `prunePairs`.

In all cases, if the 5' end of either read is unavailable (e.g. unmapped, mapping quality score below `minq`, marked as a duplicate), the read pair is recorded as unpaired and discarded. By default, no MAPQ filtering is performed when `minq` is set to `NA`. Any duplicate read must be marked in the bit field of the BAM file using a tool like Picard's `MarkDuplicates` if it is to be removed with `dedup=TRUE`. For chimeric reads, the recommended approach is to designate the 5' end as the primary alignment. This ensures that the duplicate computations are performed on the most relevant alignments for each read pair.

Miscellaneous information

The value of `yield` simply controls the amount of memory used by regulating the number of read pairs that are loaded in at any given time. Large values will require more memory but will reduce the computation time. Input files must be sorted by read name for proper execution of the function. Read pair information need not be synchronised so long as the pairing, first read and second read flags are appropriately set.

Users should note that the use of a `pairParam` object for input is strictly for convenience. Any non-empty values of `param$discard` and `param$restrict` will be ignored here. Reads will not be discarded if they lie outside the specified chromosomes, or if they lie within blacklisted regions.

Author(s)

Aaron Lun

References

Imakaev M et al. (2012). Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nat. Methods* 9, 999-1003.

Belton, RP et al. (2012). Hi-C: a comprehensive technique to capture the conformation of genomes. *Methods* 58, 268-276.

See Also

`cutGenome`, `prunePairs`, `mergePairs`, `getPairData`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

tmpf <- "gunk.h5"
preparePairs(hic.file, param, tmpf)
preparePairs(hic.file, param, tmpf, minq=50)
preparePairs(hic.file, param, tmpf, ichim=TRUE)
preparePairs(hic.file, param, tmpf, dedup=FALSE)
```

---

prunePairs *Prune read pairs*

Description

Prune the read pairs that represent potential artifacts in a Hi-C library

Usage

```
prunePairs(file.in, param, file.out=file.in, max.frag=NA, min.inward=NA, min.outward=NA)
```

Arguments

file.in	a character string specifying the path to the index file produced by prepairePairs
param	a pairParam object containing read extraction parameters
file.out	a character string specifying a path to an output index file
max.frag	an integer scalar specifying the maximum length of any sequenced DNA fragment
min.inward	an integer scalar specifying the minimum distance between inward-facing reads on the same chromosome
min.outward	an integer scalar specifying the minimum distance between outward-facing reads on the same chromosome

Details

This function removes potential artifacts from the input index file, based on the coordinates of the reads in each pair. A pairParam object is only used here for consistency; only param$fragments will be used. Any values of param$restrict or param$discard will be ignored.

Non-NA values for min.inward and min.outward are designed to protect against dangling ends and self-circles, respectively. This is particularly true when digestion is incomplete, as said structures do not form within a single restriction fragment and cannot be identified earlier. These can be removed by discarding inward- and outward-facing read pairs that are too close together.

A finite value for max.frag also protects against non-specific cleavage. This refers to the length of the actual DNA fragment used in sequencing and is computed from the distance between each read and its nearest downstream restriction site. Off-target cleavage will result in larger distances than expected.

Note the distinction between *restriction* fragments and *sequencing* fragments. The former is generated by pre-ligation digestion, and is of concern when choosing min.inward and min.outward. The ltter is generated by post-ligation shearing and is of concern when choosing max.frag.

Suitable values for each parameter can be obtained with the output of getPairData. For example, values for min.inward can be obtained by setting a suitable lower bound on the distribution of non-NA values for gap with orientation values of 1.

Value

An integer vector is invisibly returned, containing total, the total number of read pairs; length, the number of read pairs with fragment lengths greater than max.frag; inward, the number of inward-facing read pairs with gap distances less than min.inward; and outward, the number of outward-facing read pairs with gap distances less than min.outward.

Multiple data frame objects are also produced within the specified out file, for each corresponding data frame object in file.in. For each object, the number of rows may be reduced due to the removal of read pairs corresponding to potential artifacts.

Author(s)

Aaron Lun

References

Jin F et al. (2013). A high-resolution map of the three-dimensional chromatin interactome in human cells. *Nature* doi:10.1038/nature12644.

See Also

preparePairs, getPairData, squareCounts

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

require(rhdf5)
fout <- "temp.h5"
fout2 <- "temp2.h5"
invisible(preparePairs(hic.file, param, fout))
x <- prunePairs(fout, param, fout2)
h5read(fout2, "chrA/chrA")

x <- prunePairs(fout, param, fout2, max.frag=50)
h5read(fout2, "chrA/chrA")

x <- prunePairs(fout, param, fout2, min.inward=50)
h5read(fout2, "chrA/chrA")

x <- prunePairs(fout, param, fout2, min.outward=50)
h5read(fout2, "chrA/chrA")
```

---

savePairs *Save Hi-C interactions*

Description

Save a dataframe of interactions into a directory structure for rapid chromosomal access.

Usage

```
savePairs(x, file, param)
```

Arguments

x	a sorted dataframe with integer fields anchor.id and fragment.id for each interaction
file	a character string specifying the path for the output index file
param	a pairParam object containing read extraction parameters

Details

This function facilitates the input of processed Hi-C data from other sources into the current pipeline. Each entry in x$anchor.id and x$target.id refers to the index of param$fragments to denote the interacting regions. The x object will be resorted by anchor.id, then target.id. If necessary, anchor and target IDs will be switched such that the former is never less than the latter.

The coordinates of the restriction fragment boundaries in param$fragments should correspond to the reference genome being used. In most cases, these can be generated using the cutGenome function from any given BSgenome object. Values of param$discard and param$restrict will not be used here and can be ignored.

Any additional fields in x will also be saved to file. Users are recommended to put in xxx.pos and xxx.len fields (where xxx is replaced with anchor or target). This will allow removal of reads in param$discard during counting (e.g., with squareCounts). It will also allow proper calculation of statistics with getPairData, and quality control with prunePairs. See the output of preparePairs for more details on the specification of the fields.

Value

An index file is produced at the specified file location, containing the interaction data. A NULL value is invisibly returned.

Author(s)

Aaron Lun

See Also

preparePairs, cutGenome

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <-readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

n <- 1000
all.a <- as.integer(runif(n, 1L, length(cuts)))
all.t <- as.integer(runif(n, 1L, length(cuts)))
x <- data.frame(anchor.id=pmax(all.a, all.t), target.id=pmin(all.a, all.t),
anchor.pos=runif(1:100), anchor.len=10,
target.pos=runif(1:100), target.len=-10)
x <- x[order(x$anchor.id, x$target.id),]
```

```
fout <- "temp2.h5"
savePairs(x, fout, param)
require(rhdf5)
head(h5read(fout, "chrA/chrA"))
```

`squareCounts` *Load Hi-C interaction counts*

Description

Collate count combinations for interactions between pairs of bins across multiple Hi-C libraries.

Usage

```
squareCounts(files, param, width=50000, filter=1L)
```

Arguments

`files`	a character vector containing paths to the index files generated from each Hi-C library
`param`	a `pairParam` object containing read extraction parameters
`width`	an integer scalar specifying the width of each bin in base pairs
`filter`	an integer scalar specifying the minimum count for each square

Details

The genome is first split into non-overlapping adjacent bins of size `width`. These bins are rounded to the nearest restriction site, forming the sides of potential squares. The number of restriction fragments in each bin is stored as `nfrags` in the metadata of the output `region`. Each square represents an interaction between the corresponding rounded bins on the genome. The number of read pairs between each pair of sides is counted for each library to obtain the count for the corresponding square.

Larger counts can be collected by increasing the value of `width`. This can improve detection power by increasing the evidence for significant differences. However, this comes at the cost of spatial resolution as adjacent events in the same bin or square can no longer be distinguished. This may reduce detection power if counts for differential interactions are contaminated by counts for non-differential interactions.

Low abundance squares with count sums below `filter` are not reported. This reduces memory usage for large datasets. These squares are probably uninteresting as detection power will be poor for low counts. Another option is to increase `width` to reduce the total number of bins in the genome (and hence, the possible number of bin pairs).

Counting will consider the values of `restrict`, `discard` and `cap` in `param`. See `pairParam` for more details.

Value

A `DIList` object is returned containing the number of read pairs for each bin pair across all libraries. The genomic coordinates for each bin are also recorded.

Author(s)

Aaron Lun

References

Imakaev M et al. (2012). Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nat. Methods* 9, 999-1003.

Lieberman-Aiden E et al. (2009). Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. *Science* 326, 289-293.

See Also

`preparePairs`, `cutGenome`, `DIList-class`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(fragments=cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, file=fout))

# Collating to count combinations.
y <- squareCounts(fout, param)
head(counts(y))
y <- squareCounts(fout, param, filter=1)
head(counts(y))
y <- squareCounts(fout, param, width=50, filter=1)
head(counts(y))
y <- squareCounts(fout, param, width=100, filter=1)
head(counts(y))

# Attempting with other parameters.
y <- squareCounts(fout, reform(param, restrict="chrA"), width=100, filter=1)
head(counts(y))
y <- squareCounts(fout, filter=1,
    param=reform(param, restrict=cbind("chrA", "chrB")))
head(counts(y))
y <- squareCounts(fout, filter=1,
    param=reform(param, cap=1), width=100)
head(counts(y))
y <- squareCounts(fout, width=100, filter=1,
    param=reform(param, discard=GRanges("chrA", IRanges(1, 50))))
head(counts(y))
```

`totalCounts` *Get the total counts*

Description

Get the total number of read pairs in a set of Hi-C libraries.

Usage

```
totalCounts(files, param)
```

Arguments

`files`	a character vector containing paths to the index files generated from each Hi-C library
`param`	a `pairParam` object containing read extraction parameters

Details

As the name suggests, this function counts the total number of read pairs in each index file prepared by `preparePairs`. Use of `param$fragments` ensures that the chromosome names in each index file are consistent with those in the desired genome (e.g., from `cutGenome`). Counting will also consider the values of `restrict`, `discard` and `cap` in `param`.

Value

An integer vector is returned containing the total number of read pairs in each library.

Author(s)

Aaron Lun

See Also

`preparePairs`, `cutGenome`, `pairParam`, `squareCounts`

Examples

```
hic.file <- system.file("exdata", "hic_sort.bam", package="diffHic")
cuts <- readRDS(system.file("exdata", "cuts.rds", package="diffHic"))
param <- pairParam(cuts)

# Setting up the parameters
fout <- "output.h5"
invisible(preparePairs(hic.file, param, file=fout))
```

```
# Counting totals, and comparing them.
totalCounts(fout, param)
squareCounts(fout, param, width=10)$totals

new.param <- reform(param, restrict="chrA")
totalCounts(fout, new.param)
squareCounts(fout, new.param, width=10)$totals

new.param <- reform(param, discard=GRanges("chrA", IRanges(1, 50)))
totalCounts(fout, new.param)
squareCounts(fout, new.param, width=10)$totals

new.param <- reform(param, cap=1)
totalCounts(fout, new.param)
squareCounts(fout, new.param, width=10)$totals
```

Index

∗Topic **clustering**
boxPairs, 2
clusterPairs, 4
compartmentalize, 6
∗Topic **counting**
connectCounts, 8
DIList-class, 17
marginCounts, 37
neighborCounts, 40
pairParam, 44
squareCounts, 57
totalCounts, 59
∗Topic **diagnostics**
getPairData, 35
∗Topic **documentation**
diffHicUsersGuide, 16
∗Topic **filtering**
enrichedPairs, 25
Filtering diagonals, 27
Filtering methods, 29
getArea, 32
getDistance, 34
∗Topic **normalization**
correctedContact, 12
DIList-wrappers, 20
normalizeCNV, 42
∗Topic **preprocessing**
cutGenome, 15
DNaseHiC, 21
loadData, 36
mergePairs, 39
preparePairs, 50
prunePairs, 53
savePairs, 55
∗Topic **testing**
consolidatePairs, 10
∗Topic **visualization**
plotDI, 46
plotPlaid, 48

[,DIList,ANY,ANY-method *(DIList-class)*, 17
$,DIList-method *(DIList-class)*, 17
$,pairParam-method *(pairParam)*, 44
$<-,DIList-method *(DIList-class)*, 17

anchors *(DIList-class)*, 17
anchors,DIList-method *(DIList-class)*, 17
as.matrix,DIList-method *(DIList-class)*, 17
asDGEList *(DIList-wrappers)*, 20
asDGEList,DIList-method *(DIList-wrappers)*, 20
aveLogCPM, *31*

boxPairs, 2, *11*

c,DIList-method *(DIList-class)*, 17
clusterPairs, 4, *11*
colData,DIList-method *(DIList-class)*, 17
combineTests, *11*
compartmentalize, 6
connectCounts, *4*, 8, *33, 34*
consolidatePairs, 10
correctedContact, 12
counts *(DIList-class)*, 17
counts,DIList-method *(DIList-class)*, 17
cutGenome, 15, *22, 23*, *44, 45*, *52, 53*, *56*, *58, 59*

DGEList, *20, 21*
diffHic *(diffHicUsersGuide)*, 16
diffHicUsersGuide, 16
DIList *(DIList-class)*, 17
DIList-class, 17
DIList-wrappers, 20
dim,DIList-method *(DIList-class)*, 17
dimnames,DIList-method *(DIList-class)*, 17
DNaseHiC, 21

domainDirections, 23

enrichedPairs, 25, *31, 32*, *41*
exptData,DIList-method *(*DIList-class*)*, 17
exptData<-,DIList,SimpleList-method *(*DIList-class*)*, 17

filterDiag *(*Filtering diagonals*)*, 27
filterDirect *(*Filtering methods*)*, 29
Filtering diagonals, 27
Filtering methods, 29
filterPeaks, *26*, 31
filterTrended, *6, 7*, *26*
filterTrended *(*Filtering methods*)*, 29
findOverlaps, *9*

getArea, 32
getDistance, *27, 28*, 34
getPairData, 35, *51*, *53–56*

kmeans, *6, 7*

loadChromos *(*loadData*)*, 36
loadData, 36
locfit, *42, 43*
loessFit, *29*
lp, *43*

marginCounts, 37, *42, 43*
match, *3*
matchMargins *(*normalizeCNV*)*, 42
matchPattern, *16*
mergePairs, 39, *53*
mglmOneGroup, *13, 14*

neighborCounts, *31, 32*, 40
normalize *(*DIList-wrappers*)*, 20
normalize,DIList-method *(*DIList-wrappers*)*, 20
normalizeCNV, 42
normOffsets, *21*
normOffsets *(*DIList-wrappers*)*, 20
normOffsets,DIList-method *(*DIList-wrappers*)*, 20

pairParam, *9*, *38*, 44, *57*, *59*
pairParam-class *(*pairParam*)*, 44
plot, *46*, *49*
plotDI, 46
plotPlaid, *47*, 48
preparePairs, *22, 23*, *35–37*, *39*, *48*, *50*, 50, *54–56*, *58, 59*
prepPseudoPairs *(*DNaseHiC*)*, 21
prunePairs, *35, 36*, *39*, *52, 53*, 53, *56*

reform *(*pairParam*)*, 44
reform,pairParam-method *(*pairParam*)*, 44
regions *(*DIList-class*)*, 17
regions,DIList-method *(*DIList-class*)*, 17
rotDI *(*plotDI*)*, 46
rotPlaid, *47*
rotPlaid *(*plotPlaid*)*, 48

savePairs, 55
scaledAverage, *30*
segmentGenome *(*DNaseHiC*)*, 21
show,DIList-method *(*DIList-class*)*, 17
show,pairParam-method *(*pairParam*)*, 44
squareCounts, *2–7*, *9*, *12–14*, *19–21*, *24–27*, *29–34*, *38*, *41–43*, *45*, *47*, *55, 56*, 57, *59*
Sweave, *17*
system, *17*

targets *(*DIList-class*)*, 17
targets,DIList-method *(*DIList-class*)*, 17
text, *49*
totalCounts, 59