Package 'diffHic'

May 5, 2024

Version 1.36.0

Date 2021-11-16

Title Differential Analyis of Hi-C Data

Depends R (>= 3.5), GenomicRanges, InteractionSet, SummarizedExperiment

Imports Rsamtools, Rhtslib, Biostrings, BSgenome, rhdf5, edgeR, limma, csaw, locfit, methods, IRanges, S4Vectors, GenomeInfoDb, BiocGenerics, grDevices, graphics, stats, utils, Rcpp, rtracklayer

Suggests BSgenome.Ecoli.NCBI.20080805, Matrix, testthat

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

LinkingTo Rhtslib (>= 1.13.1), zlibbioc, Rcpp

SystemRequirements C++11, GNU make

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

git_branch RELEASE_3_19

git_last_commit 4910e38

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-05

Author Aaron Lun [aut, cre], Gordon Smyth [aut]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

`annotatePairs` *Annotate bin pairs*

Description

Annotate bin pairs based on features overlapping the anchor regions.

Usage

```
annotatePairs(data.list, regions, rnames=names(regions), indices, ...)
```

Arguments

data.list	An InteractionSet or a list of InteractionSet objects containing bin pairs.
regions	A GRanges object containing coordinates for the regions of interest.
rnames	A character vector containing names to be used for annotation.
indices	An integer vector or list of such vectors, indicating the cluster identity for each interaction in data.list.
...	Additional arguments to pass to findOverlaps.

Details

Entries in regions are identified with any overlap to anchor regions for interactions in data.list. The names for these entries are concatenated into a comma-separated string for easy reporting. Typically, gene symbols are used in names, but other values can be supplied depending on the type of annotation. This is done separately for the first and second anchor regions so that potential interactions between features of interest can be identified.

If indices is supplied, all interactions corresponding to each unique index are considered to be part of a single cluster. Overlaps with all interactions in the cluster are subsequently concatenated into a single string. Cluster indices should range from [1, nclusters] for any given number of clusters. This means that the annotation for a cluster corresponding to a certain index can be obtained by subsetting the output vectors with that index. Otherwise, if indices is not set, all interactions are assumed to be their own cluster, i.e., annotation is returned for each interaction separately.

Multiple InteractionSet objects can be supplied in data.list, e.g., if the cluster consists of bin pairs of different sizes. This means that indices should also be a list of vectors where each vector indicates the cluster identity of the entries in the corresponding InteractionSet of data.list.

Value

A list of two character vectors anchor1 and anchor2 is returned, containing comma-separated strings of names for entries in regions overlapped by the first and second anchor regions respectively. If indices is not specified, overlaps are identified to anchor regions of each interaction in data.list. Otherwise, overlaps are identified to anchor regions for any interaction in each cluster.

Author(s)

Aaron Lun

See Also

findOverlaps, 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)
y <- InteractionSet(matrix(0, ncol(all.combos), 1),
   GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,], regions=cuts, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param, width=1))

regions <- GRanges(rep(c("chrA", "chrB"), c(3,2)), IRanges(c(1,5,8,3,3), c(1,5,8,3,4)))
names(regions) <- LETTERS[seq_along(regions)]
out <- annotatePairs(y, regions=regions)

# Again, with indices:
indices <- sample(20, length(y), replace=TRUE)
out <- annotatePairs(y, regions=regions, indices=indices)

# Again, with multiple InteractionSet objects:
out <- annotatePairs(list(y, y[1:10,]), regions=regions, indices=list(indices, indices[1:10]))
```

`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, index.only=FALSE)
```

Arguments

`...`	One or more named InteractionSet 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.
`index.only`	A logical scalar indicating whether only indices 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 InteractionSet 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 InteractionSet 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 InteractionSet in the ellipsis. In each vector, each entry represents an index for a single input bin pair in the corresponding InteractionSet. This index points to the entries in `interactions` 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 InteractionSet that are nested within bin pairs in another InteractionSet. This can be done by supplying both InteractionSet objects in the ellipsis, and leaving `reference` unspecified. The value of `reference` will be automatically selected as the largest width of the supplied InteractionSet objects. Nesting can be identified by `match`ing the output indices for the smaller bin pairs to those of the larger bin pairs.

If `minbox=TRUE`, the coordinates in `interactions` 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.

If `index.only=TRUE`, only the indices are returned and coordinates are not computed. This is largely for efficiency purposes when `boxPairs` is called by internal functions.

Value

If `index.only=FALSE`, a named list is returned containing:

`indices`**:** a named list of integer vectors for every InteractionSet in the ellipsis, see Details.

`interactions`**:** A ReverseStrictGInteractions object containing the coordinates of the parent bin pair or, if `minbox=TRUE`, the minimum bounding box.

If `index.only=TRUE`, the `indices` are returned directly without computing coordinates.

Author(s)

Aaron Lun

See Also

`squareCounts`, `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 <- InteractionSet(matrix(0, ncol(all.combos), 1),
   GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,], regions=cuts, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param, width=1))

a5 <- a/5
b5 <- b/5
all.combos2 <- combn(length(cuts)/5, 2) # Bin size of 5.
y2 <- InteractionSet(matrix(0, ncol(all.combos2), 1),
    GInteractions(anchor1=all.combos2[2,], anchor2=all.combos2[1,],
     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))), mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param, width=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, index.only=FALSE)
```

Arguments

`...`	One or more InteractionSet objects, optionally named.
`tol`	A numeric scalar specifying the maximum distance between bin pairs in base pairs.
`upper`	A numeric scalar specifying the maximum size of each cluster in base pairs.
`index.only`	A logical scalar indicating whether only indices should be returned.

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 interactions. The cluster ID in each indices vector represents the row index for these coordinates.

If index.only=TRUE, only the indices are returned and coordinates of the bounding box are not computed. This is largely for efficiency purposes when clusterPairs is called by internal functions.

Value

If index.only=FALSE, a named list is returned containing:

indices: A named list of integer vectors where each vector contains a cluster ID for each interaction in the corresponding input InteractionSet object.

interactions: A ReverseStrictGInteractions object containing the coordinates of the sides of the bounding box for each cluster.

If index.only=TRUE, the indices are returned directly without computing coordinates.

Author(s)

Aaron Lun

See Also

squareCounts, diClusters, boxPairs

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.anchor1 <- sample(length(regions), 50, replace=TRUE)
all.anchor2 <- as.integer(runif(50, 1, all.anchor1+1))
y <- InteractionSet(matrix(0, 50, 1),
   GInteractions(anchor1=all.anchor1, anchor2=all.anchor2, regions=regions, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=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
rep.regions <- GRanges(rep(c("chrA", "chrB"), c(a2, b2)),
    IRanges(c(1:a2*2, 1:b2*2), c(1:a2*2, 1:b2*2)))
rep.anchor1 <- sample(length(rep.regions), 10, replace=TRUE)
rep.anchor2 <- as.integer(runif(10, 1, rep.anchor1+1))
y2 <- InteractionSet(matrix(0, 10, 1),
   GInteractions(anchor1=rep.anchor1, anchor2=rep.anchor2, regions=rep.regions, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=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`	an InteractionSet 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 ContactMatrix object, 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 ContactMatrix object containing (normalized) contact frequencies for the intra-chromosomal space. Entries in `compartment` 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
anchor1 <- sample(npts, npairs, replace=TRUE)
anchor2 <- sample(npts, npairs, replace=TRUE)
data <- InteractionSet(
    list(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs)),
    GInteractions(anchor1=anchor1, anchor2=anchor2,
        regions=GRanges(c(rep("chrA", 80), rep("chrB", 20)),
            IRanges(c(1:80, 1:20), c(1:80, 1:20))), mode="reverse"),
    colData=DataFrame(totals=runif(nlibs, 1e6, 2e6)), metadata=List(width=1))
data <- unique(data)

# 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, restrict.regions=FALSE)
```

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

`restrict.regions` A logical scalar indicating whether the output regions should be limited to entries in `param$restrict`.

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. To save memory, pairs of intervals 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 if nested intervals are present in `regions`.

For typical Hi-C experiments, mapping of read pairs into intervals is performed at the level of restriction fragments. 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 expands the effective boundaries of each entry of `regions` to the nearest restriction site. In contrast, setting `type="within"` would contract each interval.

For DNase Hi-C experiments, the interval spanned by the alignment of each read is overlapped against the intervals in `regions`. This uses the `linkOverlaps` function, which responds to any specification of `type`. The boundaries of `regions` are not modified as no restriction fragments are involved.

Counting will consider the values of `restrict`, `discard` and `cap` in `param` - see `pairParam` for more details. In all cases, strandedness of the intervals is ignored in input and set to `"*"` in the output object. Any element metadata in the input `regions` is also removed in the output.

Value

An InteractionSet containing the number of read pairs in each library that are mapped between pairs of `regions`, or between `regions` and `second.regions`. Interacting regions are returned as a ReverseStrictGInteractions object containing the concatenated `regions` and `second.regions`.

Matching to a second set of regions

The `second.regions` argument allows specification of a second set of regions. 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. Note that the sets are not parallel, and

any pairing is considered if as long as it contains one region from the first set and another from the second set.

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.

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. This is useful for 4C-like experiments where interactions between viewpoints and the rest of the genome are of interest.

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

Format of the output regions

For standard Hi-C experiments, all supplied regions are expanded or contracted to the nearest restriction site. These modified regions can be extracted from the regions slot in the output InteractionSet object, which 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.

For DNase-C experiments, no expansion of the regions is performed, so the coordinates in the output regions slot are the same as those in the input regions. However, reordering may still be necessary, in which case the original field will specify the original index of each entry. All nfrags are set to zero as no restriction fragments are involved.

If second.regions is specified, the output regions slot will contain both the input regions and the second.regions (though not necessarily in that order). Entries that were originally in second.regions 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. Conversely, if is.second=FALSE, the index will point towards the corresponding entry in the original regions.

If second.regions is an integer scalar, the entries in the output regions slot will contain the coordinates for the resulting bins. Note that the original metadata field is set to NA for these bins, as no original GRanges existed for these intervals.

If restrict.regions=TRUE and param$restrict is not NULL, only bins on the chromosomes in param$restrict will be reported in the output regions slot. This avoids the overhead of constructing many bins when only a small subset of them are used. By default, restrict.regions=FALSE to ensure that the anchor IDs of the output object are directly comparable between different settings of param$restrict,

Author(s)

Aaron Lun

See Also

squareCounts, findOverlaps, InteractionSet-class, ReverseStrictGInteractions-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(assay(con))
con <- connectCounts(fout, param, regions=regions, filter=1L, type="within")
head(assay(con))

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

---

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 DI 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 <- InteractionSet(matrix(0, ncol(all.combos), 1),
   GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,], regions=cuts, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param, width=1))

a5 <- a/5
b5 <- b/5
all.combos2 <- combn(length(cuts)/5, 2) # Bin size of 5.
y2 <- InteractionSet(matrix(0, ncol(all.combos2), 1),
    GInteractions(anchor1=all.combos2[2,], anchor2=all.combos2[1,],
    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))), mode="reverse"),
     colData=DataFrame(lib.size=1000), metadata=List(param=param, width=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 <- InteractionSet(matrix(0, ncol(all.combos3), 1),
    GInteractions(anchor1=all.combos3[2,], anchor2=all.combos3[1,],
     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))),
        mode="reverse"),
     colData=DataFrame(lib.size=1000), metadata=List(param=param, width=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, assay=1)
```

Arguments

data	an InteractionSet 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
assay	a string or integer scalar specifying the matrix to use from data

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 biases for each of the two anchor bins and the true interaction probability. The bias represents the ease of sequencing/mapping/other for the genome sequence in each bin.

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 first and second anchor 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
anchor1 <- sample(npts, npairs, replace=TRUE)
anchor2 <- sample(npts, npairs, replace=TRUE)
data <- InteractionSet(
    list(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs)),
    GInteractions(anchor1=anchor1, anchor2=anchor2,
        regions=GRanges("chrA", IRanges(1:npts, 1:npts)), mode="reverse"),
colData=DataFrame(totals=runif(nlibs, 1e6, 2e6)))

# 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.
anchor1.bias <- stuff$bias[anchors(data, type="first", id=TRUE),]
anchor2.bias <- stuff$bias[anchors(data, type="second", id=TRUE),]
offsets <- log(anchor1.bias * anchor2.bias)



# 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$bias[anchors(data, type="first", id=TRUE),]) +
    log(stuff$bias[anchors(data, type="second", id=TRUE),])
```

---

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 containing a path to a FASTA file.
pattern	A character vector containing one or more recognition sites.
overhang	An integer vector specifying the length of the 5' overhang for each element in pattern.

Details

This function simulates a restriction digestion of a specified genome, given the recognition site and 5' overhang of the restriction enzyme. The total sequence spanned by each fragment is recorded

including the two sticky ends after they are filled in. 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.

Multiple restriction enzymes can be specified by passing vectors to `pattern` and `overhang`. All recognition sites are expected to be inverse palindromes, and for any given enzyme, the width of `pattern` and `overhang` must be both odd or even.

No attempt is made to remove fragments that cannot be physically formed, e.g., from recognition sites that overlap with themselves or each other. This generally is not problematic for downstream analysis, as they are short and will not have many assigned reads. If they are a concern, most of them can be removed by simply applying a suitable threshold (e.g., 10 bp) on the fragment width. However, the best solution is to simply choose (combinations of) restriction enzymes that do not overlap.

Value

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

Note on FASTA sequence names

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)

# Multiple sites with different overhangs are supported.
cutGenome(x, c("AGCT", "AGNCT"), overhang=c(4, 3))
```

`diClusters` *Cluster significant bin pairs to DIs*

Description

Cluster significant bin pairs to DIs with post-hoc cluster-level FDR control.

Usage

```
diClusters(data.list, result.list, target, equiweight=TRUE, cluster.args=list(),
    pval.col="PValue", fc.col=NA, grid.length=21, iterations=4)
```

Arguments

`data.list`	an InteractionSet or a list of InteractionSet objects containing bin pairs.
`result.list`	a data frame or a list of data frames containing the DI test results for each bin pair.
`target`	a numeric scalar specifying the desired cluster-level FDR.
`equiweight`	a logical scalar indicating whether equal weighting from each input object should be enforced
`cluster.args`	a list of parameters to supply to `clusterPairs`.
`pval.col`	a character string or integer scalar specifying the column of p-values for elements in `result.list`.
`fc.col`	a character string or integer scalar specifying the column of log-fold changes for elements in `result.list`.
`grid.length, iterations`	Parameters to supply to `controlClusterFDR`.

Details

Bin pairs are identified as being significant based on the adjusted p-values in the corresponding data frame of `result.list`. Only these significant bin pairs are clustered together via `clusterPairs`. This identifies DIs consisting only of significant bin pairs. By default, the `tol` parameter in `clusterPairs` is set to 1 bp, i.e., all adjacent bin pairs are clustered together. If `fc.col` is specified, all clusters consist of bin pairs that are changing in the same direction.

The aim is to avoid very large clusters from blind clustering in areas of the interaction space that have high interaction intensity. This includes interactions within structural domains, or in data sets where interactions are difficult to define due to high levels of noise. Post-hoc control of the cluster-level FDR is performed using the `controlClusterFDR` function. This is necessary as clustering is not blind to the test results. By default, the cluster-level FDR is controlled at 0.05 if `target` is not specified.

Some effort is required to equalize the contribution of the results from each element of `result.list`. This is done by setting `equiweight=TRUE`, where the weight of each bin pair is inversely proportional to the number of bin pairs from that analysis. These weights are used as frequency weights for bin pair-level FDR control, when identifying significant bin pairs prior to clustering. Otherwise, the final results would be dominated by large number of small bin pairs.

Value

A list of cluster indices and minimum bounding boxes is returned as described in clusterPairs. An additional FDR field is also present, containing the estimate of the cluster-level FDR.

Author(s)

Aaron Lun

See Also

clusterPairs, controlClusterFDR

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.
y1 <- InteractionSet(matrix(0, ncol(all.combos), 1),
   GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,], regions=cuts, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param, width=1))

set.seed(1000)
result1 <- data.frame(logFC=rnorm(nrow(y1)), PValue=runif(nrow(y1)), logCPM=0)
result1$PValue[sample(nrow(result1), 50)] <- 0

# Consolidating with post-hoc control.
out <- diClusters(y1, result1, target=0.05, cluster.args=list(tol=1))
out
```

---

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()
```

DNaseHiC *Methods for processing DNase Hi-C data*

Description

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

Usage

```
emptyGenome(bs)

# Deprecated
segmentGenome(bs)

# Deprecated
prepPseudoPairs(bam, param, file, dedup=TRUE, minq=NA, ichim=TRUE,
    chim.span=1000, output.dir=NULL, storage=5000L)
```

Arguments

bs	a BSgenome object, or a character string pointing to a FASTA file, or a named integer vector of chromosome lengths
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
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
output.dir	a character string specifying a directory for temporary files
storage	an integer scalar specifying the maximum number of pairs to store in memory before writing to file

Details

DNase Hi-C uses DNase to randomly fragment the genome, rather than using restriction fragments. This requires some care to handle in **diffHic**, as most functions rely on fragment assignments in many functions. To specify that the data are from a DNase Hi-C experiment, an empty GRanges object should be supplied as the fragments in pairParam. Most functions will automatically recognise that the data are DNase Hi-C and behave appropriately. This reflects the fact that no restriction fragments are involved in this analysis. Genome information will instead be extracted from the seqlengths of the GRanges object.

prepPseudoPairs and segmentGenome are deprecated in favour of preparePairs and emptyGenome, respectively. In the case of preparePairs, it will automatically detect if the BAM file is to be treated as DNase Hi-C based on param$fragments.

Value

For emptyGenome, an empty GRanges object is produced containing the sequence names and lengths.

Author(s)

Aaron Lun

See Also

preparePairs, cutGenome

Examples

```
require(BSgenome.Ecoli.NCBI.20080805)
emptyGenome(Ecoli)
emptyGenome(c(chrA=100, chrB=200))
```

`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 number of read pairs between that bin and the `span` upstream bins (i.e., those with higher genomic coordinates). This is repeated for the `span` downstream bins, thus yielding two counts (up and down) per bin.

A RangedSummarizedExperiment is returned containing the coordinates of each bin and two matrices of counts, named `"up"` and `"down"`. Each row of the matrix corresponds to a bin, while each column corresponds to a library in `files`. Each entry of the matrix stores the total count of read pairs to upstream or downstream bins.

The total up- and downstream counts 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. Alternatively, it can be used to identify differential domains - see the user's guide for more details.

Value

A RangedSummarizedExperiment object with one row for each bin in the genome. It contains two integer matrices named `"up"` and `"down"`, containing the counts to upstream and downstream bins respectively.

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 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, file=fout))

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

# Calculating directionality log-FC with a large prior.
up.counts <- assay(out, "up")
down.counts <- assay(out, "down")
dir.logFC <- log2((up.counts+10)/(down.counts+10))
dir.logFC

# Calculating directionality index with Dixon's method.
dixon.stat <- sign(up.counts-down.counts)*2*(
    (up.counts-down.counts)/(up.counts+down.counts))^2
dixon.stat
```

---

enrichedPairs *Collect local enrichment statistics for bin pairs*

Description

Determine the count for a variety of local neighborhoods around a bin pair, for use in computing peak enrichment statistics.

Usage

```
enrichedPairs(data, flank=5, exclude=0, assay.in=1, assay.out=NULL)
```

Arguments

data	an InteractionSet 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
assay.in	a string or integer scalar, specifying the assay containing bin pair counts in data
assay.out	a character vector containing 4 unique names for the neighborhood regions A-D, see below

Value

An object of the same type as data is returned, containing additional matrices in the assays slot. Each matrix contains the counts for one neighborhood for each bin pair in each library. The area of each neighborhood is also returned in the mcols and named with the "N." prefix.

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 (named "quadrant") 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 B (named "vertical") is a vertical rectangle with dimensions (1, flank*2+1), containing the target bin pair at the center. Region C (named "horizontal") is the horizontal counterpart to B. Region D (named "surrounding") is a square with side lengths equal to flank*2+1, where the target bin pair is positioned in the center.

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 size of the neighborhood counts 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 count size is inflated. On the other hand, if flank is too small, there will not be enough neighborhood bin pairs to dilute the increase in counts 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.

For each bin pair, the other bin pairs in data that belong to its neighborhood are identified. The sum of counts across these bin pairs is computed for each library and stored in a matrix. This is repeated for each type of neighborhood (A-D), and the matrices are named based on assay.out. The area of each neighborhood is also computed in terms of the number of bin pairs contained by the neighborhood. Note that the neighborhood area includes bin pairs that are missing from data, as these are assumed to have a count of zero. See filterPeaks for how these neighborhood counts are used to assess the "peak-ness" of each bin pair.

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, neighborCounts, 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.anchor1 <- sample(length(regions), 50, replace=TRUE)
all.anchor2 <- as.integer(runif(50, 1, all.anchor1+1))
data <- InteractionSet(matrix(as.integer(rnbinom(200, mu=10, size=10)), 50, 4),
    GInteractions(anchor1=all.anchor1, anchor2=all.anchor2,
        regions=regions, mode="reverse"),
    colData=DataFrame(lib.size=1:4*1000), metadata=List(width=1))
data$totals <- colSums(assay(data))

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

---

extractPatch *Extract a patch of the interaction space*

Description

Extract and count read pairs into bin pairs for a subset of the interaction space.

Usage

```
extractPatch(file, param, first.region, second.region=first.region,
    width=10000, restrict.regions=FALSE)
```

Arguments

file	character string specifying the path to an index file produced by prepare Pairs
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
width	an integer scalar specifying the width of each bin in base pairs
restrict.regions	A logical scalar indicating whether the output regions should be limited to entries in param$restrict.

Details

This function behaves much like squareCounts, but only for the "path" of the interaction space defined by first.region and second.region. Read pairs are only counted into bin pairs where one end overlaps first.region and the other end overlaps second.region. This allows for rapid extraction of particular regions of interest without having to count across the entire interaction space.

Note that the first anchor region (i.e., bin) in each bin pair is not necessarily the bin that overlaps first.region. In each pair, the bins are sorted so that the first bin has a higher genomic coordinate than the second bin. The flipped flag in the metadata of the output object indicates whether this order is flipped. If TRUE, the first bin in each pair corresponds to second.region, and vice versa.

If restrict.regions=TRUE, only bins on the chromosomes in first.region and second.region will be reported in the regions slot of the output object. This avoids the overhead of constructing many bins when only a small subset of them are used. By default, restrict.regions=FALSE to ensure that the anchor IDs of the output object are directly comparable between different calls to extractPatch.

Value

An InteractionSet object containing the number of read pairs for each bin pair in the specified patch.

Author(s)

Aaron Lun

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 <- tempfile(fileext=".h5")
```

```
invisible(preparePairs(hic.file, param, file=fout))

stuff <- extractPatch(fout, param, GRanges("chrA:1-100"))
interactions(stuff)

stuff <- extractPatch(fout, param, GRanges("chrA:1-100"), GRanges("chrB:1-20"))
interactions(stuff)
```

`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`	an InteractionSet 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 `pairdist`, 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

`pairdist`

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 <- tempfile(fileext=".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=pairdist(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, assay.data=1, assay.ref=1)

filterTrended(data, span=0.25, prior.count=2, reference=NULL, assay.data=1, assay.ref=1)
```

Arguments

data	an InteractionSet 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 InteractionSet object, usually containing data for larger bin pairs
assay.data	a string or integer scalar specifying the count matrix to use from data
assay.ref	a string or integer scalar specifying the count matrix to use from reference

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.anchor1 <- sample(length(regions), npairs, replace=TRUE)
all.anchor2 <- as.integer(runif(npairs, 1, all.anchor1+1))
counts <- matrix(rnbinom(npairs*4, mu=10, size=10), npairs, 4)
y <- InteractionSet(list(counts=counts), GInteractions(anchor1=all.anchor1,
        anchor2=all.anchor2, regions=regions, mode="reverse"),
    colData=DataFrame(totals=colSums(counts)), metadata=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
rep.regions <- 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
rep.anchor1 <- sample(length(rep.regions), npairs2, replace=TRUE)
rep.anchor2 <- as.integer(runif(npairs2, 1, rep.anchor1+1))
y2 <- InteractionSet(list(counts=matrix(rnbinom(npairs2*4, mu=10*w^2, size=10), npairs2, 4)),
   GInteractions(anchor1=rep.anchor1, anchor2=rep.anchor2, regions=rep.regions, mode="reverse"),
    colData=DataFrame(totals=y$totals), metadata=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, assay.bp=1, assay.neighbors=NULL, get.enrich=FALSE,
    min.enrich=log2(1.5), min.count=5, min.diag=2L, ...)
```

Arguments

`data`	an InteractionSet object produced by `enrichedPairs` or `neighborCounts`
`enrichment`	a numeric vector of enrichment values
`assay.bp`	a string or integer scalar specifying the assay containing bin pair counts
`assay.neighbors`	a character vector containing names for the neighborhood regions, see `enrichedPairs` for details
`get.enrich`	a logical scalar indicating whether enrichment values should be returned
`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.

To compute enrichment values, we assume that the number of read pairs in neighborhood areas have been counted using `enrichedPairs` or `neighborCounts`. For a given bin pair in `data`, this function computes the mean abundance across libraries for each surrounding neighborhood, scaled by the neighborhood area (i.e., the number of bin pairs it contains). 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.

Value

If `get.enrich=TRUE`, a numeric vector of enrichment values for each bin pair. Otherwise, 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.anchor1 <- sample(length(regions), 50, replace=TRUE)
all.anchor2 <- as.integer(runif(50, 1, all.anchor1+1))
data <- InteractionSet(
    list(counts=matrix(as.integer(rnbinom(200, mu=10, size=10)), 50, 4)),
   GInteractions(anchor1=all.anchor1, anchor2=all.anchor2, regions=regions, mode="reverse"),
    colData=DataFrame(totals=runif(4, 1e6, 2e6)), metadata=List(width=1))

# Getting peaks.
enrichment <- enrichedPairs(data)
summary(filterPeaks(enrichment, min.enrich=0.5))
summary(filterPeaks(enrichment, min.enrich=0.5, min.count=10))
summary(filterPeaks(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	an InteractionSet 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 an InteractionSet 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:::.assignBins(param, 20)$region
all.combos <- combn(length(regions), 2)
y <- InteractionSet(matrix(0, ncol(all.combos), 1),
    GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,],
```

```
        regions=regions, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=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 <- InteractionSet(matrix(0, ncol(all.combos), 1),
    GInteractions(anchor1=all.combos[2,], anchor2=all.combos[1,],
        regions=regions, mode="reverse"),
    colData=DataFrame(lib.size=1000), metadata=List(param=param))

#### Getting areas. ####

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

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

`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. This will be set to `NA` if the fragment IDs are non-positive, e.g., for DNase Hi-C data (where the concept of fragments is irrelevant anyway).

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 read mapped into the first or second anchor fragment respectively is 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.

getPairData will now respect any settings of restrict, discard or cap in the input pairParam object. Statistics will not be reported for read pairs that lie outside of restricted chromosomes, within discarded regions or exceed the cap for a restriction fragment pair. Note that cap will be ignored for DNase-C experiments as this depends on an unknown bin size.

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 <- tempfile(fileext=".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, anchor1, anchor2)
```

Arguments

`file`	a character string containing a path to a index file
`anchor1`, `anchor2`	a character string, specifying the name of the chromosomes in a 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 anchor1/anchor2 specification is permuted. 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 `anchor1` and `anchor2`. 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 <- tempfile(fileext=".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, restrict.regions=FALSE)
```

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
`restrict.regions`	A logical scalar indicating whether the output regions should be limited to entries in `param$restrict`.

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.

Each row of the output RangedSummarizedExperiment refers to a single bin in the linear genome, instead of a bin pair in the interaction space. The count matrix for all row can be extracted using the `assay` method. Bin coordinates can be extracted using the `rowRanges` method.

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 *no* filtering is performed to remove empty bins. This is meant to make it easier to match up results with the output of `squareCounts`, as anchor IDs are directly comparable. If `restrict.regions=TRUE`, only counts for bins in chromosomes in `param$fragments` are returned.

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

Value

A RangedSummarizedExperiment object containing the marginal counts for each bin.

Author(s)

Aaron Lun

See Also

squareCounts, RangedSummarizedExperiment-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 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, fout))

# Collating to count combinations.
mar <- marginCounts(fout, param, width=10)
head(assay(mar))
mar <- marginCounts(fout, param, width=50)
head(assay(mar))
mar <- marginCounts(fout, param, width=100)
head(assay(mar))

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

---

mergeCMs *Merge ContactMatrix objects*

Description

Merge ContactMatrix objects into an InteractionSet object containing counts for pairs of interacting regions.

Usage

```
mergeCMs(..., deflate.args=list())
```

Arguments

...	ContactMatrix objects containing read pair counts for the same area of the interaction space, usually defined as pairs of bins along a genomic interval.
deflate.args	A list of arguments to pass to deflate.

Details

This function facilitates the conversion of multiple ContactMatrix objects into a single InteractionSet object. Each ContactMatrix corresponds to a sample and should contain read pair counts between the corresponding row/column regions. The dimensions of all ContactMatrix objects should be the same and the rows/columns should represent the same genomic regions.

The idea is to produce an object equivalent to the output of `squareCounts` when contact matrices are available instead of BAM files. This is done via the `deflate` method, where each ContactMatrix is converted to an InteractionSet using `deflate.args`. Entries of the ContactMatrix are equivalent to paired regions in an InteractionSet (which, in most cases, are bins of constant width).

The InteractionSet objects for all supplied samples are then combined into a single object for downstream input. This step will throw errors if the original ContactMatrix objects do not cover the same area of the interaction space. Column names are set to any names for `...`, and the total number of read pairs in each ContactMatrix is stored in `totals`.

The `width` value in the `metadata` of the output InteractionSet is set to the median width of the interacting regions. The `totals` field in the output `colData` is also set to be equal to the sum of the counts in each ContactMatrix (after removing redundant regions). Note that this only makes sense if the ContactMatrix objects contain interactions between non-overlapping genomic bins.

The `param` value is not set in the `metadata` of the output object. This depends on how the ContactMatrix objects were constructed in the first place, which is not known to the function.

Value

An InteractionSet object containing counts for interacting regions.

Author(s)

Aaron Lun

See Also

`deflate`, `squareCounts`, `connectCounts`

Examples

```
example(ContactMatrix, echo=FALSE)
mergeCMs(x, x2)
```

`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 <- tempfile(fileext=".h5")
fout2 <- tempfile(fileext=".h5")
fout3 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, fout))
invisible(prunePairs(fout, param, fout2))
invisible(prunePairs(fout, param, fout3, max.frag=50))

# Note: don't save to a temporary file for actual data.
mout <- tempfile(fileext=".h5")
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)
```

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

Details

This function combines the functionality of squareCounts and enrichedPairs. The idea is to allow counting of neighborhoods 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 assay matrices are equivalent to that computed with enrichedPairs with the default settings.

Value

An InteractionSet object is returned with the number of read pairs for each bin pair across all libraries. For each bin pair, the number of read pairs in each neighborhood region is also returned in separate assay fields. mcols contains the size of each neighborhood in terms of the number of 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, 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 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, file=fout))

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

---

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,
    assay.data=1, assay.marg=1, ...)

matchMargins(data, margins)
```

Arguments

data	an InteractionSet object produced by squareCounts
margins	a RangedSummarizedExperiment 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
assay.data	a string or integer scalar specifying the matrix to use from data
assay.marg	a string or integer scalar specifying the matrix to use from margins
...	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 objects in data and margins should be constructed with the same width and param arguments 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.

For use by downstream functions, the offset matrix can be stored as an assay named "offset" in the data object.

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 anchor1 and anchor2. Each field specifies the index in margins corresponding to the bin regions 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
anchor1 <- sample(npts, npairs, replace=TRUE)
anchor2 <- sample(npts, npairs, replace=TRUE)

data <- InteractionSet(
    list(counts=matrix(rpois(npairs*nlibs, runif(npairs, 10, 100)), nrow=npairs)),
    GInteractions(anchor1=anchor1, anchor2=anchor2,
        regions=GRanges("chrA", IRanges(1:npts, 1:npts)), mode="reverse"),
colData=DataFrame(totals=runif(nlibs, 1e6, 2e6)))

margins <- SummarizedExperiment(matrix(rpois(npts*nlibs, 100), nrow=npts),
colData=DataFrame(totals=data$totals), rowRanges=regions(data))

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

# Store offsets as the 'offset' assay for use by, e.g., asDGEList.
assays(data)$offset <- out
data

# Occasionally locfit will complain; increase maxk to compensate.
data <- InteractionSet(matrix(rpois(npairs*nlibs, 20), nrow=npairs),
    GInteractions(anchor1=anchor1, anchor2=anchor2,
        regions=GRanges("chrA", IRanges(1:npts, 1:npts)), mode="reverse"),
colData=DataFrame(totals=runif(nlibs, 1e6, 2e6)))
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)
anchor1.counts <- margins[matched$anchor1,]
anchor2.counts <- margins[matched$anchor2,]
```

`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 2-column 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, max.height=NULL, ylab="Gap", ...)
```

Arguments

data	an InteractionSet 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
max.height	a numeric scalar indicating the y-axis limit for rotPlaid
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.anchor1 <- sample(length(regions), 500, replace=TRUE)
all.anchor2 <- as.integer(runif(500, 1, all.anchor1+1))
out <- InteractionSet(matrix(0, 500, 1), colData=DataFrame(lib.size=1000),
        GInteractions(anchor1=all.anchor1, anchor2=all.anchor2,
            regions=regions, mode="reverse"), metadata=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, max.height=NULL, ylab="Gap", ...)
```

Arguments

file	character string specifying the path to an index file produced by preparePairs
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
max.height	a numeric scalar indicating the y-axis limit for rotPlaid
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.

By default, `max.height` is chosen to include the interaction between the boundaries `region` in `rotPlaid`. This is equivalent to the width of `region`. Smaller values can be chosen to focus on interactions closer to the diagonal. Larger values can also be used, but this is less useful as the interacting bins cannot be easily traced (as they will lie outside the x-axis limits).

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 <- tempfile(fileext=".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 plot.
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, ichim=TRUE,
    chim.dist=NA, output.dir=NULL, storage=5000L)
```

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
`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
output.dir	a character string specifying a directory for temporary files
storage	an integer scalar specifying the maximum number of pairs to store in memory before writing to file

Value

Multiple dataframe objects are stored within the specified file using the HDF5 format. Each object corresponds to a pair of chromosomes, designated as the anchor1 (later) or anchor2 (earlier) chromosome based on the order of their names. Each row of the dataframe contains information for a read pair, with one read mapped to each chromosome. The dataframe contains several integer fields:

anchor1.id, anchor2.id: Index of the anchor1 or anchor2 restriction fragment to which each read was assigned.

anchor1.pos, anchor2.pos: 1-based genomic coordinate of the aligned read (or the 5' segment thereof, for chimeras) on the anchor1 or anchor2 fragment.

anchor1.len, anchor2.len: Length of the alignment on the anchor1 or anchor2 fragment. This is multiplied by -1 for alignments on the reverse strand.

A list 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 read or 5' segment; filtered, read pairs where the MAPQ score for either read or 5' segment 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 all 5' segments and non-chimeric reads 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

For DNase Hi-C data, the anchor1.id and anchor2.id fields are set to zero, and the same.id field in the output list is removed.

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 anchor1 fragment is that with the higher genomic coordinate, i.e., the larger index in param$fragments. The anchor2 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.

Details of read pair processing

A read pair is discarded if either read is unavailable, e.g., unmapped, mapping quality score below minq, marked as a duplicate. 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.

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.

For pairs with chimeric reads, the segment containing the 5' end of each chimeric read is used to assign the fragment index. Chimeric read pairs are discarded if the 5' segments of the chimeric reads are not available, regardless of what happens with the 3' segment. Note that, when running MarkDuplicates with chimeric reads, the recommended approach is to designate the 5' segment as the only primary or non-supplementary alignment. This ensures that the duplicate computations are performed on the most relevant alignments for each read pair.

Invalid chimeras arise when the index/position of the 3' segment of a chimeric read is not consistent with that of the mate read. These are generally indicative of mapping errors but can also form due to non-specific ligation events. Computationally, invalid chimeras can be defined in two ways:

- If chim.dist=NA, a chimeric pair is considered to be invalid if the 3' segment and the mate do not map onto the same restriction fragment in an inward-facing orientation. This reflects the resolution limits of the Hi-C protocol.
- If chim.dist is not NA, chimeras are defined based on distance. A pair is considered invalid if the distance between the segment and mate is greater than chim.dist, or if the alignments are not inward-facing.

The second approach is more relevant in situations involving inefficient cleavage, where the mapping locations are broadly consistent but do not fall in the same restriction fragment. The maximum size of the ligation products can be used as a reasonable value for chim.dist, e.g., 1000 bp. While invalid chimeras can be explicitly removed, keeping ichim=TRUE is recommended to avoid excessive filtering due to inaccurate alignment of short chimeric 3' segments.

Processing DNase Hi-C experiments

DNase Hi-C involves random fragmentation with DNase instead of restriction enzymes. To indicate that the data are generated by DNase Hi-C, an empty GRanges object should be supplied as the

fragments in pairParam. Genome information will instead be extracted from the seqlengths of the GRanges object. This empty object can be generated by emptyGenome.

The BAM file can be processed with preparePairs in a manner that is almost identical to that of normal Hi-C experiments. However, there are some key differences:

- No reporting or removal of self-circles or dangling ends is performed, as these have no meaning when restriction fragments are not involved.
- Chimeras are considered invalid if the 3' segment of one read and the 5' segment of the mate are not inward-facing or more than chim.dist away from each other. If chim.dist=NA, it will be set to a default of 1000 bp.
- All fragment IDs in the output HDF5 file will be set to zero. The first read of each pair is defined as the read on the chromosome that is ordered later in seqlengths(fragments). For pairs on the same chromosome, the first read is defined as that with a higher genomic coordinate for its 5' end.

Miscellaneous information

If output.dir is not specified, a directory name is constructed using the tempfile command. If it is specified, users should make sure that no file already exists with that name. Otherwise, an error will be raised. This directory is used to store intermediate files that will be eventually processed into the HDF5 output file.

For low-memory systems or in cases where there are many chromosome pairs, users may need to reduce the value of storage. This will write data to file more frequently, which reduces memory usage at the cost of speed.

Users should note that the use of a pairParam object for input is strictly for convenience. Only the value of param$fragments will be used. Any non-empty values of param$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, with use of Rhtslib based on comments from Alessandro Mammana.

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)

# Note: don't save to a temporary file for actual data.
tmpf <- tempfile(fileext=".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)

# Pretending it's DNase Hi-C:
eparam <- pairParam(cuts[0])
preparePairs(hic.file, eparam, tmpf)
preparePairs(hic.file, eparam, tmpf, dedup=FALSE)
preparePairs(hic.file, eparam, tmpf, minq=50)
preparePairs(hic.file, eparam, tmpf, chim.dist=20)
```

`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 `preparePairs`
`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. It will then produce a new HDF5 file containing only the retained read pairs.

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 restriction 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. However, `max.frag` should not be set for DNase Hi-C experiments where there is no concept of non-specific cleavage.

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 latter 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 `insert` with `orientation==1`.

`prunePairs` will now respect any settings of `restrict`, `discard` and `cap` in the `pairParam` input object. Reads will be correspondingly removed from the file if they lie outside of restricted chromosomes, within discarded regions or exceed the cap for a restriction fragment pair. Note that `cap` will be ignored for DNase-C experiments as this depends on an unknown bin size.

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)

# Note: don't save to a temporary file for actual data.
fout <- tempfile(fileext=".h5")
fout2 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, fout))
x <- prunePairs(fout, param, fout2)

require(rhdf5)
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")
```

---

readMTX2IntSet *Create an InteractionSet from a BED file and Matrix Market files*

Description

Read a set of Matrix Market Exchange Format files from disk and create an InteractionSet object.

Usage

```
readMTX2IntSet(mtx, bed, as.integer=TRUE)
```

Arguments

mtx	A character vector containg paths to Matrix Market Exchange Format (MTX) files. Each file contains interaction read counts for one sample.
bed	String containing the path to the BED file specifying the genomic regions.
as.integer	Logical indicating whether the data should be read as integers. Otherwise values are stored in double-precision numbers.

Details

Each MTX file is assumed to contain read counts for symmetric matrix representing the two-dimensional interaction space. Each row and column is assumed to correspond to contiguous bins of the genome, with coordinates specified by bed in standard BED format. This function will aggregate counts from all files to create an InteractionSet object that mimics the output of squareCounts.

The `width` value in the `metadata` of the output InteractionSet is set to the median width of the regions. The `totals` field in the output `colData` is also set to be equal to the sum of the counts in each MTX file. Note that these settings only make sense if the ContactMatrix objects cover binned regions.

This function can, in principle, read and merge any number of MTX files. However, for large data sets, consider reading each MTX file separately, subsetting it to interactions of interest and then creating the InteractionSet object. For example, subsetted contact matrices can be used to create an InteractionSet via `mergeCMs`.

Value

An InteractionSet object containing interactions between regions from the BED file (in reverse-strict mode, see `GInteractions`). Each row corresponds to a unique interaction found in any of the MTX files, and contains the read counts across all files.

Author(s)

Gordon Smyth, with modifications by Aaron Lun

See Also

`mergeCMs`

Examples

```
library(Matrix)
tmp.loc <- tempfile()
dir.create(tmp.loc)

# Mocking up some MTX and BED files.
set.seed(110000)
A <- rsparsematrix(1000, 1000, density=0.1, symmetric=TRUE,
    rand.x=function(n) round(runif(n, 1, 100)))
A.name <- file.path(tmp.loc, "A.mtx")
writeMM(file=A.name, A)

B <- rsparsematrix(1000, 1000, density=0.1, symmetric=TRUE,
    rand.x=function(n) round(runif(n, 1, 100)))
B.name <- file.path(tmp.loc, "B.mtx")
writeMM(file=B.name, B)

GR <- GRanges(sample(c("chrA", "chrB", "chrC"), 1000, replace=TRUE),
    IRanges(start=round(runif(1000, 1, 10000)),
        width=round(runif(1000, 50, 500))))
GR <- sort(GR)
bed.name <- file.path(tmp.loc, "regions.bed")
rtracklayer::export.bed(GR, con=bed.name)

# Reading everything in.
iset <- readMTX2IntSet(c(A.name, B.name), bed.name)
iset
```

savePairs *Save Hi-C read pairs*

Description

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

Usage

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

Arguments

x	A dataframe with integer fields anchor1.id and anchor2.id. Each row corresponds to a single read pair.
file	A character string specifying the path for the output index file.
param	A pairParam object containing read extraction parameters. In particular, param$fragments should contain genomic regions corresponding to the anchor*.id values.

Details

This function facilitates the input of processed Hi-C data from other sources into the current pipeline. Each row of x corresponds to a read pair, and each entry in x$anchor1.id and x$anchor2.id contains an index for param$fragments. Thus, the pair of indices for each row denotes the the interacting regions for each read pair. These regions are generally expected to be restriction fragments in conventional Hi-C experiments.

Obviously, the coordinates of the restriction fragment boundaries in param$fragments should correspond to the genome to which the reads were aligned. These can be generated using the cutGenome function from any given BSgenome object or the FASTA files used for alignment. Values of param$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 anchor1.pos, anchor1.len, anchor2.pos and anchor2.len fields. These should mimic the output of preparePairs:

anchorY.pos: Integer field, containing the 1-based genomic position of the left-most aligned base of read Y.

anchorY.len: Integer field, containing the length of the alignment of read Y on the reference sequence. This should be multiplied by -1 if the alignment was on the negative strand.

These fields enable the use of more **diffHic** functions, e.g., removal of reads in param$discard during counting with squareCounts, correct calculation of statistics with getPairData, quality control with prunePairs.

For storing DNase Hi-C data, param$fragments should be empty but the seqinfo should contain the lengths and names of all chromosomes. Here, the input anchor1.id and anchor2.id should contain indices of the seqlengths. This specifies the chromosome to which each read is aligned, e.g., an anchor1.id of 2 means that read 1 is aligned to the second chromosome in seqlengths. Note that, for this type of data, it is essential to store the position and length fields mentioned above.

When constructing the output file, x will be resorted by anchor1.id, then anchor2.id. If necessary, anchor1 and anchor2 indices will be switched such that the former is never less than the latter. For DNase Hi-C data, both of these fields will ultimately be set to zero - see prepPseudoPairs for more details.

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(anchor1.id=all.a, anchor2.id=all.t,
anchor1.pos=runif(1:100), anchor1.len=10,
anchor2.pos=runif(1:100), anchor2.len=-10)

# Note: don't save to a temporary file for actual data.
fout <- tempfile(fileext=".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, restrict.regions=FALSE)
```

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
restrict.regions	A logical scalar indicating whether the output regions should be limited to entries in param$restrict.

Details

The genome is first split into non-overlapping adjacent bins of size width. In the two-dimensional space, squares are formed from pairs of bins and represent interactions between the corresponding intervals 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.

For standard Hi-C data, bins are rounded to the nearest restriction site. The number of restriction fragments in each bin is stored as nfrags in the metadata of the output region. For DNase Hi-C data, no rounding is performed as restriction fragments are irrelevant during DNase digestion. Each read is placed into a bin based on the location of its 5' end, and nfrags for all bins are set to zero.

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).

If restrict.regions=TRUE and param$restrict is not NULL, only bins on the chromosomes in param$restrict will be reported in the regions slot of the output InteractionSet object. This avoids the overhead of constructing many bins when only a small subset of them are used. By default, restrict.regions=FALSE to ensure that the anchor IDs of the output object are directly comparable between different settings of param$restrict, e.g., for merging the results of multiple squareCounts calls.

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

Value

An InteractionSet object is returned containing the number of read pairs for each bin pair across all libraries. Bin pairs are stored as a ReverseStrictGInteractions object.

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, InteractionSet-class, ReverseStrictGInteractions-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 <- tempfile(fileext=".h5")
invisible(preparePairs(hic.file, param, file=fout))

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

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