Package 'TrajectoryUtils'

November 12, 2026

Version 1.20.0

Date 2024-02-03

Title Single-Cell Trajectory Analysis Utilities

Description Implements low-level utilities for single-cell trajectory analysis, primarily intended for re-use inside higher-level packages. Include a function to create a cluster-level minimum spanning tree and data structures to hold pseudotime inference results.

License GPL-3

biocViews GeneExpression, SingleCell

Depends SingleCellExperiment

Imports methods, stats, Matrix, igraph, S4Vectors, SummarizedExperiment

Suggests BiocNeighbors, DelayedArray, DelayedMatrixStats, BiocParallel, testthat, knitr, BiocStyle, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.1

BugReports https://github.com/LTLA/TrajectoryUtils/issues

URL https://bioconductor.org/packages/TrajectoryUtils

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

git_branch RELEASE_3_23

git_last_commit 72c185e

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Aaron Lun [aut, cre], Kelly Street [aut]

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

Contents

averagePseudotime 2
createClusterMST 3
defineMSTPaths 8
guessMSTRoots 9
PseudotimeOrdering 10
rowmean 13
splitByBranches 14

Index **16**

`averagePseudotime` *Compute the average pseudotime*

Description

Compute the average pseudotime for each cell across all paths in which it is involved.

Usage

```
averagePseudotime(x, i = 1L)
```

Arguments

`x`	A numeric matrix-like object containing pseudotime orderings. Alternatively, a PseudotimeOrdering object containing such a matrix.
`i`	Integer scalar or string specifying the entry of `pathStats(x)` containing the pseudotime matrix, if `x` is a PseudotimeOrdering object.

Details

Averaging the pseudotime is a convenient way to consolidate multiple paths into a single ordering for, e.g., visualization. It is permissible as cells involved in multiple paths should generally have similar pseudotimes in each path, under assumption that the cell is involved in the part of the trajectory that is common to those paths. In such cases, the average is just a way of compressing those pseudotimes into a single value. Conversely, for cells that are unique to a single path, the average collapses to that path's pseudotime (assuming that all other values are `NA`).

Value

A numeric vector containing the average pseudotime for each cell.

Author(s)

Aaron Lun

Examples

```
pseudotimes <- matrix(rnorm(200), ncol=2)
pseudotimes[1:40,1] <- NA
pseudotimes[61:100,2] <- NA
pseudotimes[41:60,] <- runif(20)
averagePseudotime(pseudotimes)

pto <- PseudotimeOrdering(pseudotimes)
averagePseudotime(pto)
```

`createClusterMST` *Minimum spanning trees on cluster centroids*

Description

Build a MST where each node is a cluster centroid and each edge is weighted by the Euclidean distance between centroids. This represents the most parsimonious explanation for a particular trajectory and has the advantage of being directly intepretable with respect to any pre-existing clusters.

Usage

```
createClusterMST(x, ...)

## S4 method for signature 'ANY'
createClusterMST(
  x,
  clusters,
  use.median = FALSE,
  outgroup = FALSE,
  outscale = 1.5,
  endpoints = NULL,
  columns = NULL,
  dist.method = c("simple", "scaled.full", "scaled.diag", "slingshot", "mnn"),
  with.mnn = FALSE,
  mnn.k = 50,
  BNPARAM = NULL,
  BPPARAM = NULL
)

## S4 method for signature 'SummarizedExperiment'
createClusterMST(x, ..., assay.type = "logcounts")

## S4 method for signature 'SingleCellExperiment'
createClusterMST(
  x,
  clusters = colLabels(x, onAbsence = "error"),
```

```
  ...,
  use.dimred = NULL
)
```

Arguments

x	A numeric matrix of coordinates where each row represents a cell/sample and each column represents a dimension (usually a PC or another low-dimensional embedding, but features or genes can also be used). Alternatively, a SummarizedExperiment or SingleCellExperiment object containing such a matrix in its assays, as specified by assay.type. This will be transposed prior to use. Alternatively, for SingleCellExperiments, this matrix may be extracted from its reducedDims, based on the use.dimred specification. In this case, no transposition is performed. Alternatively, if clusters=NULL, a numeric matrix of coordinates for cluster centroids, where each row represents a cluster and each column represents a dimension Each row should be named with the cluster name. This mode can also be used with assays/matrices extracted from SummarizedExperiments and SingleCellExperiments.
...	For the generic, further arguments to pass to the specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method (if use.dimred is specified) or the ANY method (otherwise).
clusters	A factor-like object of the same length as nrow(x), specifying the cluster identity for each cell in x. If NULL, x is assumed to already contain coordinates for the cluster centroids. Alternatively, a matrix with number of rows equal to nrow(x), containing soft assignment weights for each cluster (column). All weights should be positive and sum to 1 for each row.
use.median	A logical scalar indicating whether cluster centroid coordinates should be computed using the median rather than mean.
outgroup	A logical scalar indicating whether an outgroup should be inserted to split unrelated trajectories. Alternatively, a numeric scalar specifying the distance threshold to use for this splitting.
outscale	A numeric scalar specifying the scaling of the median distance between centroids, used to define the threshold for outgroup splitting. Only used if outgroup=TRUE.
endpoints	A character vector of clusters that must be endpoints, i.e., nodes of degree 1 or lower in the MST.
columns	A character, logical or integer vector specifying the columns of x to use. If NULL, all provided columns are used by default.
dist.method	A string specifying the distance measure to be used, see Details.
with.mnn	Logical scalar, deprecated; use dist.method="mnn" instead.

mnn.k	An integer scalar specifying the number of nearest neighbors to consider for the MNN-based distance calculation when dist.method="mnn". See findMutualNN for more details.
BNPARAM	A BiocNeighborParam object specifying how the nearest-neighbor search should be performed when dist.method="mnn", see the **BiocNeighbors** package for more details.
BPPARAM	A BiocParallelParam object specifying whether the nearest neighbor search should be parallelized when dist.method="mnn", see the **BiocNeighbors** package for more details.
assay.type	An integer or string specifying the assay to use from a SummarizedExperiment x.
use.dimred	An integer or string specifying the reduced dimensions to use from a SingleCellExperiment x.

Value

A graph object containing an MST computed on centers. Each node corresponds to a cluster centroid and has a numeric vector of coordinates in the coordinates attribute. The edge weight is set to the Euclidean distance and the confidence is stored as the gain attribute.

Computing the centroids

By default, the cluster centroid is defined by taking the mean value across all of its cells for each dimension. If clusters is a matrix, a weighted mean is used instead. This treats the column of weights as fractional identities of each cell to the corresponding cluster.

If use.median=TRUE, the median across all cells in each cluster is used to compute the centroid coordinate for each dimension. (With a matrix-like clusters, a weighted median is calculated.) This protects against outliers but is less stable than the mean. Enabling this option is advisable if one observes that the default centroid is not located near any of its points due to outliers. Note that the centroids computed in this manner is not a true medoid, which was too much of a pain to compute.

Introducing an outgroup

If outgroup=TRUE, we add an outgroup to avoid constructing a trajectory between "unrelated" clusters (Street et al., 2018). This is done by adding an extra row/column to the distance matrix corresponding to an artificial outgroup cluster, where the distance to all of the other real clusters is set to $\omega/2$. Large jumps in the MST between real clusters that are more distant than ω will then be rerouted through the outgroup, allowing us to break up the MST into multiple subcomponents (i.e., a minimum spanning forest) by removing the outgroup.

The default ω value is computed by constructing the MST from the original distance matrix, computing the median edge length in that MST, and then scaling it by outscale. This adapts to the magnitude of the distances and the internal structure of the dataset while also providing some margin for variation across cluster pairs. The default outscale=1.5 will break any branch that is 50% longer than the median length.

Alternatively, outgroup can be set to a numeric scalar in which case it is used directly as ω.

Forcing endpoints

If certain clusters are known to be endpoints (e.g., because they represent terminal states), we can specify them in `endpoints`. This ensures that the returned graph will have such clusters as nodes of degree 1, i.e., they terminate the path. The function uses an exhaustive search to identify the MST with these constraints. If no configuration can be found, an error is raised - this will occur if all nodes are specified as endpoints, for example.

If `outgroup=TRUE`, the function is allowed to connect two endpoints together to create a two-node subcomponent. This will result in the formation of a minimum spanning forest if there are more than two clusters in `x`. Of course, if there are only two nodes and both are specified as endpoints, a two-node subcomponent will be formed regardless of `outgroup`.

Note that edges involving endpoint nodes will have infinite confidence values (see below). This reflects the fact that they are forced to exist during graph construction.

Confidence on the edges

For the MST, we obtain a measure of the confidence in each edge by computing the distance gained if that edge were not present. Ambiguous parts of the tree will be less penalized from deletion of an edge, manifesting as a small distance gain. In contrast, parts of the tree with clear structure will receive a large distance gain upon deletion of an obvious edge.

For each edge, we divide the distance gain by the length of the edge to normalize for cluster resolution. This avoids overly penalizing edges in parts of the tree involving broad clusters while still retaining sensitivity to detect distance gain in overclustered regions. As an example, a normalized gain of unity for a particular edge means that its removal requires an alternative path that increases the distance travelled by that edge's length.

The normalized gain is reported as the `"gain"` attribute in the edges of the MST from `createClusterMST`. Note that the `"weight"` attribute represents the edge length.

Distance measures

Distances between cluster centroids may be calculated in multiple ways:

- The default is `"simple"`, which computes the Euclidean distance between cluster centroids.
- With `"scaled.diag"`, we downscale the distance between the centroids by the sum of the variances of the two corresponding clusters (i.e., the diagonal of the covariance matrix). This accounts for the cluster "width" by reducing the effective distances between broad clusters.
- With `"scaled.full"`, we repeat this scaling with the full covariance matrix. This accounts for the cluster shape by considering correlations between dimensions, but cannot be computed when there are more cells than dimensions.
- The `"slingshot"` option will typically be equivalent to the `"scaled.full"` option, but switches to `"scaled.diag"` in the presence of small clusters (fewer cells than dimensions in the reduced dimensional space).
- For `"mnn"`, see the more detailed explanation below.

If `clusters` is a matrix with `"scaled.diag"`, `"scaled.full"` and `"slingshot"`, a weighted covariance is computed to account for the assignment ambiguity. In addition, a warning will be raised if `use.median=TRUE` for these choices of `dist.method`; the Mahalanobis distances will not be correctly computed when the centers are medians instead of means.

Alternative distances with MNN pairs

While distances between centroids are usually satisfactory for gauging cluster "closeness", they do not consider the behavior at the boundaries of the clusters. Two clusters that are immediately adjacent (i.e., intermingling at the boundaries) may have a large distance between their centroids if the clusters themselves span a large region of the coordinate space. This may preclude the obvious edge from forming in the MST.

In such cases, we can use an alternative distance calculation based on the distance between mutual nearest neighbors (MNNs). An MNN pair is defined as two cells in separate clusters that are each other's nearest neighbors in the other cluster. For each pair of clusters, we identify all MNN pairs and compute the median distance between them. This distance is then used in place of the distance between centroids to construct the MST. In this manner, we focus on cluster pairs that are close at their boundaries rather than at their centers.

This mode can be enabled by setting `dist.method="mnn"`, while the stringency of the MNN definition can be set with `mnn.k`. Similarly, the performance of the nearest neighbor search can be controlled with `BPPARAM` and `BSPARAM`. Note that this mode performs a cell-based search and so cannot be used when `x` already contains aggregated profiles.

Author(s)

Aaron Lun

References

Ji Z and Ji H (2016). TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis. *Nucleic Acids Res.* 44, e117

Street K et al. (2018). Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics*, 477.

Examples

```
# Mocking up a Y-shaped trajectory.
centers <- rbind(c(0,0), c(0, -1), c(1, 1), c(-1, 1))
rownames(centers) <- seq_len(nrow(centers))
clusters <- sample(nrow(centers), 1000, replace=TRUE)
cells <- centers[clusters,]
cells <- cells + rnorm(length(cells), sd=0.5)

# Creating the MST:
mst <- createClusterMST(cells, clusters)
plot(mst)

# We could also do it on the centers:
mst2 <- createClusterMST(centers, clusters=NULL)
plot(mst2)

# Works if the expression matrix is in a SE:
library(SummarizedExperiment)
se <- SummarizedExperiment(t(cells), colData=DataFrame(group=clusters))
mst3 <- createClusterMST(se, se$group, assay.type=1)
```

```
plot(mst3)
```

`defineMSTPaths` *Define paths through the MST*

Description

Define paths through the MST, either from pre-specified root nodes or based on external timing information.

Usage

```
defineMSTPaths(g, roots, times = NULL, clusters = NULL, use.median = FALSE)
```

Arguments

`g`	A graph object containing a minimum spanning tree, e.g., from `createClusterMST`.
`roots`	A character vector specifying the root node for each component in `g`.
`times`	A numeric vector of length equal to the number of nodes in `g`, specifying the external time associated with each node. This should be named with the name of each node. Alternatively, a numeric vector of length equal to the number of cells, in which case `clusters` must be specified.
`clusters`	A vector or factor specifying the assigned cluster for each cell, where each cluster corresponds to a node in `g`. Alternatively, a matrix with number of rows equal to `nrow(x)`, containing soft assignment weights for each cluster (column). All weights should be positive and sum to 1 for each row. This only has an effect if `times` is set to a vector of length equal to the number of cells.
`use.median`	Logical scalar indicating whether the time for each cluster is defined as the median time across its cells. The mean is used by default. This only has an effect if `clusters` is specified.

Details

When `roots` is specified, a path is defined from the root to each endpoint node (i.e., with degree 1) in `g`. We expect one root node to be specified for each component in `g`.

When `times` is specified, a path is defined from each local minima in time to the nearest local maxima within each component of `g`. Timing information can be defined from experimental metadata or with computational methods like RNA velocity.

Value

A list of character vectors. Each vector contains the names of nodes in `g` and defines a path through the MST from a root to an endpoint node.

Author(s)

Aaron Lun

See Also

guessMSTRoots, to obtain roots without any prior information.

splitByBranches, for a root-free way of obtaining paths.

Examples

```
library(igraph)
test.g <- make_graph(c("A", "B", "B", "C", "B", "D"), directed=FALSE)
defineMSTPaths(test.g, roots="A")
defineMSTPaths(test.g, roots="B")

defineMSTPaths(test.g, times=c(A=0, B=1, C=2, D=3))
defineMSTPaths(test.g, times=c(A=0, B=-1, C=2, D=3))
defineMSTPaths(test.g, times=c(A=0, B=5, C=2, D=3))
```

---

guessMSTRoots *Guess the roots of a MST*

Description

Pick nodes to use as the root(s) of an MST using a variety of ad hoc methods.

Usage

```
guessMSTRoots(
  g,
  method = c("degree1", "maxstep", "maxlen", "minstep", "minlen")
)
```

Arguments

g	A graph object containing a MST. All nodes should be named.
method	String specifying the method to use to pick the root.

Details

When method="degree1", an arbitrary node of degree 1 is chosen as the root for each component. This aims to reduce the number of branch events by starting from an already-terminal node.

When method="maxstep", we pick the node of degree 1 that has the highest average number of steps to reach all other nodes of degree 1. This aims to maximize the number of shared clusters between different paths when traversing g from the root to the other terminal nodes, under the philosophy that branch events should occur as late as possible. When method="maxlen", we instead

pick the node of degree 1 that has the highest average distance to reach all other nodes of degree 1. This also considers the distance spanned by each cluster.

When `method="minstep"`, we pick the node that has the lowest average number of steps to reach all nodes of degree 1. This aims to minimize the number of shared clusters between different paths under the philosophy that branch events should occur as early as possible. When `method="minlen"`, we instead pick the node that has the highest average distance to reach all nodes of degree 1.

Value

A character vector containing the identity of the root for each component in g.

Author(s)

Aaron Lun, based on code by Kelly Street

Examples

```
library(igraph)
edges <- c("A", "B", "B", "C", "C", "D", "C", "E")
g <- make_graph(edges, directed=FALSE)

guessMSTRoots(g)
guessMSTRoots(g, method="maxstep")
guessMSTRoots(g, method="minstep")

# Works with multiple components.
edges2 <- c(edges, "F", "G", "G", "H")
g2 <- make_graph(edges2, directed=FALSE)
guessMSTRoots(g2)
```

PseudotimeOrdering *The PseudotimeOrdering class*

Description

The PseudotimeOrdering class defines a two-dimensional object where rows represent cells and columns represent paths through a trajectory (i.e., "lineages"). It is expected to contain a numeric matrix of pseudotime orderings for each cell (row) in each path (column). If a cell is on a path, it should have a valid pseudotime for the corresponding column; otherwise its entry should be set to `NA`. Cells may lie on multiple paths if those paths span shared regions of the trajectory.

Constructor

`PseudotimeOrdering(pathStats, cellData=NULL, pathData=NULL, metadata=list()` will construct a PseudotimeOrdering object given:

- pathStats, a (usually numeric) matrix-like object of pseudotime orderings as described above. Alternatively, a list of such matrices can be supplied if multiple statistics are associated with each cell/path combination. By convention, the first matrix in such a list should contain the pseudotime orderings.
- cellData, a DataFrame of cell-level metadata. This should have number of rows equal to the number of cells.
- pathData, a DataFrame of path-level metadata. This should have number of rows equal to the number of paths.
- metadata, a list of any additional metadata to be stored in the object.

Getting/setting path statistics

In the following code chunks, x is a PseudotimeOrdering object.

`pathStat(x, i=1L, withDimnames=TRUE)`**:** Returns a (usually numeric) matrix-like object containing some path statistics. The default of `i=1L` will extract the first matrix of path statistics - by convention, this should contain the pseudotime orderings. `i` may also be a string if the path statistics in `x` are named. The dimnames of the output matrix are guaranteed to be the same as `dimnames(x)` if `withDimnames=TRUE`.

`pathStat(x, i=1L, withDimnames=TRUE) <- value`**:** Replaces the path statistics at `i` in `x` with the matrix `value`. This should have the same dimensions as `x`, and if `withDimnames=TRUE`, it should also have the same dimnames.

`pathStats(x, withDimnames=TRUE)`**:** Returns a list of matrices containing path statistics. The dimnames of each matrix are guaranteed to be the same as `dimnames(x)` if `withDimnames=TRUE`.

`pathStats(x, withDimnames=TRUE) <- value`**:** Replaces the path statistics in `x` with those in the list `value`. Each entry of `value` should have the same dimensions as `x`. The dimnames of each matrix should also be the same as `dimnames(x)` if `withDimnames=TRUE`.

`pathStatNames(x)`**:** Returns a character vector containing the names for each matrix of path statistics.

`pathStatNames(x) <- value`**:** Replaces the names of the path statistics with those in the character vector `value`.

Getting/setting path metadata

In the following code chunks, x is a PseudotimeOrdering object.

`npaths(x)`**:** Returns an integer scalar containing the number of paths in `x`. This is the same as `ncol(x)`.

`pathnames(x)`**:** Returns a character vector containing the names of paths in `x` (or `NULL`, if no names are available). This is the same as `colnames(x)`.

`pathnames(x) <- value`**:** Replaces the path names in `x` with those in the character vector `value` (or `NULL`, to unname the paths). This is the same as `colnames(x) <- value`.

`pathData(x, use.names=TRUE)`**:** Returns a DataFrame containing the path-level metadata of `x`. This has the same number of rows as the number of columns in `x`. Row names are guaranteed to be equal to `pathnames(x)` if `use.names=TRUE`.

`pathData(x) <- value`**:** Replaces the path-level metadata of `x` with a DataFrame `value` containing the same number of rows.

Getting/setting cell metadata

In the following code chunks, x is a PseudotimeOrdering object.

`ncells(x)`: Returns an integer scalar containing the number of cells in x. This is the same as `nrow(x)`.

`cellnames(x)`: Returns a character vector containing the names of cells in x (or `NULL`, if no names are available). This is the same as `rownames(x)`.

`cellnames(x) <- value`: Replaces the cell names in x with those in the character vector `value` (or `NULL`, to unname the cells). This is the same as `rownames(x) <- value`.

`cellData(x, use.names=TRUE)`: Returns a DataFrame containing the cell-level metadata of x. This has the same number of rows as x. Row names are guaranteed to be equal to `cellnames(x)` if `use.names=TRUE`.

`cellData(x) <- value`: Replaces the cell-level metadata of x with a DataFrame `value` containing the same number of rows.

Further operations

In the following code chunks, x is a PseudotimeOrdering object.

`x$name` and `x$name <- value` will get and set, respectively, the named field of the `cellData`. This is primarily provided for convenience.

Subsetting operations (e.g., `x[i, j]`) and combining operations (`rbind(x, ...)`, `cbind(x, ...)`) will return the expected PseudotimeOrdering object.

`metadata(x)` and `metadata(x) <- value` will get and set, respectively, the metadata of x.

Comments on advanced usage

The PseudotimeOrdering class is actually just a reskin of the widely-used SummarizedExperiment class. We re-use the same underlying data structure and simply rename `row` and `col` to cell and path, respectively. This means that any method that operates on a SummarizedExperiment can also - in theory - be applied to PseudotimeOrdering.

We chose to do this reskinning to provide a clear conceptual break between the two classes. The PseudotimeOrdering's dimensions do not follow the SummarizedExperiment's conventional "samples as columns" philosophy, as each row instead represents a cell/sample. Similarly, it is hard to argue that the paths are really interpretable as "features" in any meaningful sense. By reskinning, we hide the SummarizedExperiment implementation from the end-user and avoid any confusion with the interpretation of PseudotimeOrdering's dimensions.

Of course, we could just transpose the inputs to get them to fit into a SummarizedExperiment. However, the use of rows as cells is convenient as we often have many cells but few paths; it is easier to inspect the pseudotime ordering matrix with this orientation. It also allows us to store the PseudotimeOrdering as a column in the `colData` of a SummarizedExperiment. In this manner, datasets can be easily annotated with pseudotime orderings from trajectory reconstruction methods.

Author(s)

Aaron Lun

Examples

```
# Make up a matrix of pseudotime orderings.
ncells <- 200
npaths <- 5
orderings <- matrix(rnorm(1000), ncells, npaths)

# Default constructor:
(pto <- PseudotimeOrdering(orderings))
(pto <- PseudotimeOrdering(list(ordering=orderings)))

# Adding some per-cell metadata:
pto$cluster <- sample(LETTERS, ncells, replace=TRUE)
table(pto$cluster)

# Adding some per-path metadata:
pathData(pto)$description <- c("EMT", "differentiatoin", "activation", "other", "?")
pathData(pto)

# Subsetting and combining works fine:
rbind(pto, pto)
cbind(pto, pto)
pto[1:10,]
pto[,1:2]
```

`rowmean` *Compute column means based on a grouping variable*

Description

Computes the column mean or median for each group of rows in a matrix.

Usage

```
rowmean(x, group)

rowmedian(x, group)
```

Arguments

`x`	A numeric matrix or matrix-like object.
`group`	A vector or factor specifying the group assignment for each row of `x`. Alternatively, a matrix of soft assignments for each row to each group (column).

Details

The naming scheme here is somewhat inspired by the rowsum function. Admittedly, it is rather confusing when rowMeans computes the mean for a row across all columns while rowmean computes the mean for a column across a subset of rows, but there you have it.

If group is a matrix, it is expected to contain soft assignment weights for each row in x. Each row of group should contain non-negative values that sum to unity. These are used to compute weighted means or medians via **MatrixGenerics** functions.

Value

A numeric matrix with one row per level of group, where the value for each column contains the mean or median across the subset of rows corresponding that level.

Author(s)

Aaron Lun

Examples

```
x <- matrix(runif(100), ncol = 5)
group <- sample(1:8, 20, TRUE)
(xmean <- rowmean(x, group))
(xmeds <- rowmedian(x, group))
```

---

splitByBranches *Split a graph into branch-free paths*

Description

Split a graph (usually a MST, at least a DAG) into branch-free components, i.e., paths between branch points or terminal nodes.

Usage

```
splitByBranches(g)
```

Arguments

g	A graph object such as that produced by createClusterMST.

Details

This function implements another strategy to define paths through the MST, by simply considering all linear sections (i.e., connected by nodes of degree 2) between branch points or terminii. The idea is to enable characterisation of the continuum without external information or assumptions statements about where the root is positioned, which would otherwise be required in functions such as defineMSTPaths.

Value

A list of character vectors containins unbranched paths between branch points or terminal nodes.

Author(s)

Aaron Lun

See Also

defineMSTPaths, for the root-based method of defining paths.

Examples

```
library(igraph)
test.g <- make_graph(c("A", "B", "B", "C", "C", "D",
    "D", "E", "D", "F", "F", "G"), directed=FALSE)
splitByBranches(test.g)
```

Index

$,PseudotimeOrdering-method
(PseudotimeOrdering), 10
$<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10

assays, *4*
averagePseudotime, 2

cellData (PseudotimeOrdering), 10
cellData<- (PseudotimeOrdering), 10
cellData<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
cellnames (PseudotimeOrdering), 10
cellnames<- (PseudotimeOrdering), 10
cellnames<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
colData, *12*
createClusterMST, 3, *6*, *8*, *14*
createClusterMST,ANY-method
(createClusterMST), 3
createClusterMST,SingleCellExperiment-method
(createClusterMST), 3
createClusterMST,SummarizedExperiment-method
(createClusterMST), 3

DataFrame, *11*
defineMSTPaths, 8, *14, 15*

findMutualNN, *5*

graph, *5*, *8, 9*, *14*
guessMSTRoots, *9*, 9

ncells (PseudotimeOrdering), 10
npaths (PseudotimeOrdering), 10

pathData (PseudotimeOrdering), 10
pathData<- (PseudotimeOrdering), 10
pathData<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
pathnames (PseudotimeOrdering), 10
pathnames<- (PseudotimeOrdering), 10
pathnames<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
pathStat (PseudotimeOrdering), 10
pathStat<- (PseudotimeOrdering), 10
pathStat<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
pathStatNames (PseudotimeOrdering), 10
pathStatNames<- (PseudotimeOrdering), 10
pathStatNames<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
pathStats, *2*
pathStats (PseudotimeOrdering), 10
pathStats<- (PseudotimeOrdering), 10
pathStats<-,PseudotimeOrdering-method
(PseudotimeOrdering), 10
PseudotimeOrdering, *2*, 10
PseudotimeOrdering-class
(PseudotimeOrdering), 10

reducedDims, *4*
rowmean, 13
rowMeans, *14*
rowmedian (rowmean), 13
rowsum, *14*

show,PseudotimeOrdering-method
(PseudotimeOrdering), 10
SingleCellExperiment, *4*
splitByBranches, *9*, 14
SummarizedExperiment, *4*, *12*