Package 'clustComp'

May 3, 2026

Version 1.41.0

Date 2017-07-16

Title Clustering Comparison Package

Author Aurora Torrente and Alvis Brazma.

Maintainer Aurora Torrente <aurora@ebi.ac.uk>

Depends R (>= 3.3)

Imports sm, stats, graphics, grDevices

Suggests Biobase, colonCA, RUnit, BiocGenerics

biocViews GeneExpression, Clustering, Visualization

Description clustComp is a package that implements several techniques for the comparison and visualisation of relationships between different clustering results, either flat versus flat or hierarchical versus flat. These relationships among clusters are displayed using a weighted bi-graph, in which the nodes represent the clusters and the edges connect pairs of nodes with non-empty intersection; the weight of each edge is the number of elements in that intersection and is displayed through the edge thickness. The best layout of the bi-graph is provided by the barycentre algorithm, which minimises the weighted number of crossings. In the case of comparing a hierarchical and a non-hierarchical clustering, the dendrogram is pruned at different heights, selected by exploring the tree by depth-first search, starting at the root. Branches are decided to be split according to the value of a scoring function, that can be based either on the aesthetics of the bi-graph or on the mutual information between the hierarchical and the flat clusterings. A mapping between groups of clusters from each side is constructed with a greedy algorithm, and can be additionally visualised.

License GPL (>= 2)

NeedsCompilation no

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

git_branch devel

git_last_commit b2bace8

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-03

Contents

barycentre 2
drawTreeGraph 3
dyn.cross 6
flatVSflat 7
flatVShier 9
insert 12
SCmapping 13
score.crossing 15
score.it 16

Index **18**

`barycentre` *Computation of the barycentre-coordinate of a node connected to others in a bigraph*

Description

`barycentre` provides an updated coordinate value for a node that is connected to nodes in a different layer of a bigraph. It is computed as the average of coordinates of the adjacent nodes, where weighted edges are considered as multi-edges collapsed into one.

Usage

```
barycentre(edge.weight, coordinates = NULL)
```

Arguments

`edge.weight`	a vector containing the intersection sizes (edge weights) between a given node (from one of the partitionings) and all nodes in the other.
`coordinates`	a vector indicating the coordinates of the adjacent nodes. If it is not provided, then they are evenly spaced assuming a layout from top downwards.

Details

The node under consideration, from a given partitioning, is assigned a new position using the barycentre algorithm. The coordinates of the incident nodes are considered as many times as the corresponding edge weights indicate, and the new position for the node is given by the average over this set of coordinates.

Value

`position`	a nsumber indicating the barycentre-coordinate of the node under consideration.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

flatVSflat, flatVShier

Examples

```
# simulated data
clustering1 <- c(rep(1, 5), rep(2, 10), rep(3, 10))
clustering2 <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
weights <- table(clustering1, clustering2)
barycentre(weights[1, ], 1:4)
barycentre(weights[1, ])
barycentre(weights[2, ], 1:4)
apply(weights, 1, barycentre, 1:4)
```

---

drawTreeGraph *Plot the bi-graph determined by the branches in the tree and the flat clusters*

Description

drawTreeGraph plots both a hierarchical tree, either complete or pruned, and a flat clustering, connected with edges whose thickness is proportional to the number of elements shared by branches and clusters, to form a weighted bi-graph. Its usage is mainly internal, as part of the visualisation of the flatVShier function. The size of each cluster is also displayed.

Usage

```
drawTreeGraph(weight, current.order, coordinates, tree, dot = TRUE,
    line.wd = 3, main = NULL, expanded = FALSE, hclust.obj = NULL,
    flat.obj = NULL, labels = NULL, cex.labels = 1, expression = NULL,
    layout = NULL, ramp = NULL, bar1.col = NULL, bar2.col = NULL)
```

Arguments

weight	a contingency matrix containing the intersection sizes (edge weights) between branches in the tree and clusters from the flat partitioning.
current.order	a list of two components; the first one is a vector with the branches (rows of the matrix weight) in the ordering in which they are drawn; the second one provides the ordering for flat clusters (columns of weight). Both are drawn from bottom upwards.
coordinates	a list of two components; the first one is a vector providing the Y-coordinates, from bottom upwards, for the branches, whereas the second one provides the Y-coordinates, from bottom upwards, for the flat clusters.
tree	a list with two components: $heights, a vector describing the heights at which the different branches in the tree are agglomerated, and $branches, a matrix of 3 columns; the i-th row contains as first element the branch split at the i-th allowed splitting, and as second and third elements, the corresponding children.

dot	a Boolean parameter; if TRUE then the last split in the children-tree is shown with a green open circle.
line.wd	a number indicating the width of the thickest edge(s) in the bigraph.
main	a character string for the plot title.
expanded	a Boolean parameter indicating whether the hierarchical tree should be plotted complete or with its branches collapsed.
hclust.obj	an hclust object describing the how the leaves are merged and the ordering of the branches, which might have been changed by the gravity-centre algorithm.
flat.obj	a vector indicating the flat cluster each gene belongs to.
labels	a vector indicating the labels for the leaves in the expanded tree.
cex.labels	a number indicating the magnification used for the labels of the leaves.
expression	a matrix containing the expression data from which the dendrogram and the flat partioning were obtained.
layout	a vector containing 3 or 2 components, depending on whether the heatmap is plotted or not. Each coordinate provides the value on the X axis for locating the heatmap, the colour bar representing pruned branches, and the bigraph. Provided vectors are increasingly ordered to keep the layout of the plot.
ramp	a vector with two components containing the two colours used to define the palette for the heatmap.
bar1.col	a vector of integers or character strings indicating the colours to be used in the hierarchical coloured bar, drawn on the left hand side.
bar2.col	a vector of integers or character strings indicating the colours to be used in the flat coloured bar, drawn on the right hand side.

Details

The drawTreeGraph allows visualising the comparison of a hierarchical clustering, drawn on the left hand side of the plot, and a flat clustering, represented on the right hand side. The tree branches are labelled by their original labels preceded by 'B'; if the function is called as part of the flatVShier algorithm, then the standard notation for initial branch labels is that of hclust objects: if the label is a negative integer it corresponds to a leaf; if it is a positive integer, then it corresponds to a branch that agglomerates at least two elements, and the number represents the stage at which the branch was formed. The last splitting can be optionally highlighted with a green open circle upon the parent-node. The flat clusters are labelled by their original labels preceded by 'F'. If the dendrogram is fully expanded, the corresponding hclust object hclust.obj must be provided. Then, branches to be collapsed are marked with a red solid dot and a coloured bar shows branch sizes. If the gene expression matrix is provided as argument expression, a heatmap for the expression levels can be optionally added, using the colours defined in ramp. If the flat clustering is given as parameter flat.obj a second coloured bars shows how genes are distributed across flat clusters.

Value

a list of components including:

b.coord	a vector indicating the Y coordinates of the nodes in the bi-graph representing the branches of the hierarchical tree.
f.coord	a vector indicating the Y coordinates of the nodes in the bi-graph representing the flat clusters.
x.coords	a vector of two components indicating the X coordinates at which each layer of the bi-graph is represented.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

flatVShier

Examples

```
### simulated data
parent.clustering <- c(rep(1, 15), rep(2, 10))
# replace the branch '2' by its children '3' and '4'
children.clustering <- c(rep(1, 15), rep(3, 5), rep(4, 5))
flat.clustering <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
split <- rbind(c(0, 1, 2), c(2, 3, 4))
weight <- table(children.clustering, flat.clustering)
current.order <- list(c(3, 4, 1), 1:4)
coordinates <- list(c(-1, 0, 1), c(-1.5, -0.5, 0.5, 1.5))
tree <- list(heights = c(1, 0.8), branches = split)
drawTreeGraph(weight, current.order, coordinates, tree)

### expanded tree
set.seed(0)
myData <- matrix(rnorm(50), 10, 5)
myData[1:5,] <- myData[1:5, ] + 2  # two groups
flat.clustering <- kmeans(myData, 2)$cluster
hierar.clustering <- hclust(dist(myData))
weight <- matrix(c(5, 0, 0, 5), 2, 2)
colnames(weight) <- 1:2; rownames(weight) <- c(6,8)
current.order <- list(c(6, 8), 1:2)
coordinates <- list(c(0.25, 0.75), c(0.25, 0.75))
tree <- list(heights = hierar.clustering$height[9],
    branches = matrix(c(9, 6, 8), 1, 3))

# without heatmap
drawTreeGraph(weight, current.order, coordinates, tree,
    expanded = TRUE, hclust.obj = hierar.clustering,
    dot = FALSE)

# with heatmap
drawTreeGraph(weight, current.order, coordinates, tree,
    expanded = TRUE, hclust.obj = hierar.clustering,
    flat.obj = flat.clustering, expression = myData,
    dot = FALSE)
```

dyn.cross *Computation of the number of crossings in the bi-graph in a particular layout*

Description

dyn.cross dynamically computes the number of edge crossings for a given node ordering of both layers of the bi-graph. This ordering is given by the rownames and colnames of the matrix of weights provided as argument. The use of this function is mainly internal.

Usage

```
dyn.cross(weights)
```

Arguments

weights	a matrix containing the intersection sizes (edge weights) between clusters (nodes) from the first partitioning (left or top layer of the bi-graph) and the second partitioning (right or bottom layer of the bi-graph). The rownames and colnames of the matrix provide the ordering of the nodes in the first and second layer, respectively.

Details

The number of crossings in the weighted bigraph is computed by considering multiedges between connected nodes. The specific implementation uses a modification, adapted to the case of having multiple edges, of the dynamic programming algorithm developed by Nagamochi and Yamada for counting the number of crossings in non-weighted graphs, which has reduced computational cost.

Value

crossings	the number of weighted crossings in the layout provided as argument.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Nagamochi H. and Yamada N. (2004) Counting edge crossings in a 2-layered drawing. *Information Processing Letters*. 91, 221-225.

See Also

barycentre, flatVSflat, flatVShier

Examples

```
# simulated data
    clustering1 <- c(rep(1, 5), rep(2, 10), rep(3, 10))
    clustering2 <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
    dyn.cross(table(clustering1, clustering2)) # no crossings
    dyn.cross(table(clustering1, clustering2)[c(2, 1, 3), ])
```

`flatVSflat` *Comparison of two flat clusterings*

Description

`flatVSflat` carries out the comparison and visualisation of two flat clusterings. The nodes in each partitioning are represented as nodes in the two layers of a bi-graph. The sizes of the intersection between clusters are reflected in the edge thickness. The number of edge crossings is minimised heuristically using the barycentre algorithm alternatively on each side.

Usage

```
flatVSflat(flat1, flat2, coord1 = NULL, coord2 = NULL, max.iter = 24,
    h.min = 0.1, plotting = TRUE, horiz = FALSE, offset = 0.1, line.wd = 3,
    point.sz = 2, evenly = FALSE, greedy = TRUE, greedy.colours = NULL,
    main = "", xlab = "", ylab = "", col = NULL, ...)
```

Arguments

`flat1`	a vector of length N containing the labels for each of the N genes clustered according to the first flat method.
`flat2`	a vector of length N containing the labels for each of the N genes clustered according to the second flat method.
`coord1`	a vector indicating the coordinates of the nodes in the first layer of the bi-graph. If not provided, then the nodes are initially equally spaced.
`coord2`	a vector indicating the coordinates of the nodes in the second layer of the bi-graph. If not provided, then the nodes are initially equally spaced.
`max.iter`	an integer stating the maximum number of runs of the barycentre heuristic on both layers of the bi-graph.
`h.min`	minimum separation between nodes in the same layer; if the barycentre algorithm sets two nodes to be less than this distance apart, then the second node and the following ones are shifted (downwards, in the vertical layout, and to the right, in the horizontal layout).
`plotting`	a Boolean parameter which yields the bi-graph if TRUE.
`horiz`	a Boolean argument for displaying a vertical (default) or horizontal layout.
`offset`	a numerical parameter that sets the separation between the nodes and their labels. It is set to 0.1 by default.
`line.wd`	a numerical parameter that fixes the width of the thickest edge(s); the rest are drawn proportionally to their weights; 3 by default.
`point.sz`	a numerical parameter that fixes the size of the nodes in the bigraph; 2 by default.
`evenly`	a Boolean parameter; if TRUE the coordinate values are ignored, and the nodes are drawn evenly spaced, according to the ordering obtained by the algorithm. It is set to FALSE by default.
`greedy`	a Boolean argument; if set to TRUE the greedy algorithm is used to construct a one-to-one mapping between superclusters on both sides.

greedy.colours	a vector of integers or character strings defining the colours that will be used to visualise the superclusters. If not provided colours are selected from the default palette of size 8. If the length of this vector is smaller than that of the number of resulting superclusters p, then it is recycled, and different symbols are used.
main	graphical parameter as in plot.
xlab	graphical parameter as in plot.
ylab	graphical parameter as in plot.
col	graphical parameter as in plot.
...	further graphical parameters.

Details

As the iterations of the algorithm run the coordinates of the nodes in a single layer are updated. For a given partition, each node is assigned a new position, the gravity-centre, using the barycentre algorithm; then, the nodes in the corresponding layer are reordered according to the new positions. If the gravity-centres cause two consecutive nodes to be less than h.min apart, the coordinates of the second and all the following ones are shifted. Additionally, to improve the results of the algorithm the following strategy is also used after running the barycentre algorithm on each side: consecutive nodes are swapped if this transposition leads to a reduction in the number of edge crossings. The algorithm runs until there is no improvement in the number of crossings or until the maximum number of iterations is reached. The rownames and colnames of matrix weights contain the cluster labels. The ordering in the layout is over-imposed by the coordinate values, therefore, the names (in the coordinates) and row-/col-names (in the contingency table) should coincide.

Value

a list of components including:

icross	the number of edge crossings before running the barycentre algorithm.
fcross	the number of edge crossings after running the barycentre algorithm.
coord1	a vector containing the coordinates for each node in the first layer.
coord2	a vector containing the coordinates for each node in the second layer.
s.clustering1	a vector of length N stating the supercluster in the first layer each element belongs to. If greedy=FALSE this component is not returned.
s.clustering2	a vector of length N stating the supercluster in the second layer each element belongs to. If greedy=FALSE this component is not returned.
merging1	a list of p components; the j-th element contains the labels from the first clustering that have been merged to produce the j-th supercluster. If greedy=FALSE this component is not returned.
merging2	a list of p components; the j-th element contains the labels from the second clustering that have been merged to produce the j-th supercluster. If greedy=FALSE this component is not returned.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Eades, P. *et al.* (1986). On an edge crossing problem. Proc. of 9th Australian Computer Science Conference, pp. 327-334.

Gansner, E.R. *et al.* (1993). A technique for drawing directed graphs. IEEE Trans. on Software Engineering, 19 (3), 214-230.

Garey, M.R. *et al.* (1983). Crossing number in NP complete. SIAM J. Algebraic Discrete Methods, 4, 312-316.

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

flatVShier, barycentre

Examples

```
### simulated data; two superclusters
clustering1 <- c(rep(1, 5), rep(2, 10), rep(3, 10))
clustering2 <- c(rep(1, 4), rep(2,15), rep(1, 6))
# two colours
flatVSflat(clustering1, clustering2, greedy.colours = c("red","blue"))
# one colour, two symbols
flatVSflat(clustering1, clustering2, greedy.colours = c("red"))
# optimal bi-graph without greedy algorithm
flatVSflat(clustering1, clustering2, greedy = FALSE)


# simulated data; only one supercluster
clustering1 <- c(rep(1, 5), rep(2, 10), rep(3, 10))
clustering2 <- c(rep(1:4, 5), rep(1, 5))
flatVSflat(clustering1, clustering2, greedy.colours = c("red"))
```

`flatVShier` *Comparison of a hierarchical and a flat clusterings*

Description

`flatVShier` carries out the comparison and visualisation of the relationships between a hierarchical and a flat clusterings. The hierarchical one is shown either as a complete or pruned tree, whose collapsed branches are nodes on the left hand side layer of a bi-graph. The flat clusters are represented on the right hand side. Branches and flat clusters are connected with edges, whose thickness represents the number of elements common to both sets. The number of edge crossings is minimised using the barycentre algorithm on the right hand side; also, the children corresponding to the last split in the dendrogram when exploring it by depth-first search are swapped if this decreases the number of crossings.

Usage

```
flatVShier(tree, flat.clustering, flat.order = NULL, look.ahead = 2,
    score.function = "crossing", expanded = FALSE, expression = NULL,
    greedy = TRUE, layout = NULL, pausing = TRUE, verbose = TRUE,
    greedy.colours = NULL, h.min = 0.04, labels = NULL, max.branches = 100,
    line.wd = 3, cex.labels = 1, main = NULL, ramp = NULL, bar1.col = NULL,
    bar2.col = NULL)
```

Arguments

tree	an hclust object, or structure that can be converted to hclust object, corresponding to a data set of size N.
flat.clustering	a vector of length N containing the labels for each of the N objects clustered.
flat.order	an optional vector containing the initial ordering for the flat clusters (from bottom upwards).
look.ahead	the number of steps allowed to look further after finding a parent-node whose score is better that that of its children.
score.function	a string specifying whether the decision to split a given branch is based on the aesthetics ('crossing') or on the information theory ('it').
expanded	a Boolean parameter to state whether the hierarchical tree is displayed complete or pruned. FALSE by default.
expression	a matrix containing the expression data from which the dendrogram and the flat partioning were obtained. If provided, and argument expanded is set to TRUE, the heatmap of the data is represented.
greedy	a Boolean argument; if TRUE, the branches produced by the optimal cutoffs are used to construct superclusters that will be mapped to superclusters on the flat side with the greedy algorithm in SCmapping.
layout	a vector containing 3 or 2 components, depending on whether the heatmap is plotted or not. Each coordinate provides the value on the X axis for locating the heatmap, the colour bar representing pruned branches, and the bigraph. Provided vectors are increasingly ordered to keep the layout of the plot. By default, the X values are 1, 3, (5).
pausing	a Boolean argument; if TRUE, each step in the comparison is plotted and followed by a pause.
verbose	a Boolean argument; if TRUE, the situation in each iteration is described.
greedy.colours	an optional vector containing the colours for each of the superclusters if the greedy algorithm is used; if the length of this vector is smaller than that of the number of resulting superclusters p, then it is recycled.
h.min	minimum separation between nodes in the flat layer; if the barycentre algorithm sets two nodes to be less than this distance apart, then the second node and the following ones are shifted (upwards).
labels	an optional vector containing labels for the leaves of the tree.
max.branches	an integer stating the maximum number of branches allowed in the dendogram.
line.wd	a numerical parameter that fixes the width of the thickest edge(s); the rest are drawn proportionally to their weights; 3 by default.
cex.labels	a number indicating the magnification for the labels of the flat clusters and the branches in the hierarchical tree.

main	an optional character string for the title of the plot.
ramp	a vector with two components containing the two colours used to define the palette for the heatmap.
bar1.col	a vector of integers or character strings indicating the colours to be used in the hierarchical coloured bar, drawn on the left hand side, if the dendrogram is fully expanded.
bar2.col	a vector of integers or character strings indicating the colours to be used in the flat coloured bar, drawn on the right hand side, if the dendrogram is fully expanded.

Details

The method cuts different branches of the tree at 'optimal' levels, which may be different at different branches, to find the best matches with the flat clustering. The method explores the tree depth-first, starting from the root. In each iteration the goal is to decide whether the branch under consideration (the parent node) is to be split. To that end, the user selects a scoring function based on the bi-graph aesthetics ('crossing') or an alternative based on mutual information between the flat and hierarchical clusterings ('it'). The selected score is first computed for the parent-node. Next it is replaced by its children; the barycentre algorithm, with the swapping strategy, is used on the flat side of the bi- graph. Later, the children in the tree are swapped and the positions on the flat side are likewise updated. The best score obtained by any of these layouts in the children tree is compared to the score of the parent-tree, sp. If it is better, then the splitting is allowed and the tree is subsequently explored. Otherwise, the splitting is discarded, unless it is allowed to look ahead. In that case, the score for the tree with one of the children of the parent-node replaced by its own children is compared to sp; this is repeated until we get a better score for the children or until the maximum number of looking-ahead steps is reached. After the optimal cut-offs are found, it is possible to run a greedy algorithm to determine sets of clusters from each side which have a large overlap. These sets, referred to as superclusters, determine the mapping between the two clusterings.

Value

a list of components including:

tree.partition	a vector of length N stating the branch each element belongs to.
tree.s.clustering	a vector of length N stating the supercluster on the tree side each element belongs to. If greedy=FALSE this component is not returned.
flat.s.clustering	a vector of length N stating the supercluster on the flat side each element belongs to. If greedy=FALSE this component is not returned.
tree.merging	a list of p components; the j-th element contains the labels of the tree that have been merged to produce the j-th supercluster. If greedy=FALSE this component is not returned.
flat.merging	a list of p components; the j-th element contains the labels of the flat clusters that have been merged to produce the j-th supercluster. If greedy=FALSE this component is not returned.
dendrogram	an hclust object with the appropriate ordering of the branches to minimise the number of crossings.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

flatVSflat, barycentre, score.crossing, score.it, SCmapping

Examples

```
# simulated data
set.seed(0)
dataset <- rbind(matrix(rnorm(20), 5, 4), sweep(matrix(rnorm(24), 6, 4),
    2, 1:4, "+"))
tree <- hclust(dist(dataset))
# two clusters
flat <- kmeans(dataset,2)$cluster
collapsed1 <- flatVShier(tree, flat, pausing = FALSE)
# four clusters
flat<-kmeans(dataset, 4)$cluster
collapsed2 <- flatVShier(tree, flat)

## expanded tree
# no heatmap
expanded1 <- flatVShier(tree, flat, pausing = FALSE, score.function = "it",
    expanded = TRUE, bar1.col = c("red", "blue", "cyan", "magenta"),
    bar2.col = c("gray", "yellow", "orange", "purple"))
# with heatmap
expanded2 <- flatVShier(tree, flat, pausing = FALSE, score.function = "it",
    expanded = TRUE, expression = dataset,
    bar1.col = c("red", "blue", "cyan", "magenta"),
    bar2.col = c("gray", "yellow", "orange", "purple"))
```

---

insert *Insert a set of values at a given position of a vector*

Description

`insert` introduces a vector at a given position of another vector, displacing to the right all values from that position onwards.

Usage

```
insert(vect, position, value)
```

Arguments

`vect`	the vector in which to insert additional values.
`position`	the position of vector `vect` at which to insert additional values.
`value`	the values to be inserted in the vector `vect`.

Details

The value of `position` does not need to be a number smaller than or equal to the length of `vect`, as the missing values will be denoted as NA.

Value

`new.vector` the vector resulting after inserting `value` at the position determined by `position`.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

Examples

```
# simulated data
v1 <- 1:10
v2 <- insert(v1, 4, 0)
v2 <- insert(v1, 1, rep(0, 5))
v2 <- insert(v1, 11, "A")
v2 <- insert(v1, 12, "A")
```

SCmapping *Construction of the superclusters and the one-to-one mapping between them*

Description

`SCmapping` identifies groups of clusters from two flat partitionings that have the largest common intersections. These groups are found by following a greedy strategy: all edges incident to each cluster are removed except for the one(s) with highest weight; then the connected components in the resulting bi-graph define the correspondences of superclusters.

Usage

```
SCmapping(clustering1, clustering2, plotting = TRUE, h.min = 0.1, line.wd = 3,
    point.sz = 3, offset = 0.1, evenly = TRUE, horiz = FALSE, max.iter =24,
    node.col = NULL, edge.col = NULL,...)
```

Arguments

`clustering1`	a vector indicating the cluster in which each point is allocated in the first flat partitioning.
`clustering2`	a vector indicating the cluster in which each point is allocated in the second flat partitioning.
`plotting`	a Boolean parameter which leads to the representation of the bi-graph if TRUE.
`h.min`	the minimum separation between nodes in the same layer; if the barycentre algorithm sets two nodes to be less than this distance apart, then the second node and the following ones are shifted (downwards, in the vertical layout, and to the right, in the horizontal layout).
`line.wd`	a numerical parameter that fixes the width of the thickest edge, according to the weights; 3 by default.

point.sz	a numerical parameter that fixes the size of the nodes in the bi-graph; 2 by default.
offset	a numerical parameter that sets the separation between the nodes and their labels. It is set to 0.1 by default.
evenly	a Boolean parameter; if TRUE the coordinate values are ignored, and the nodes are drawn evenly spaced, according to the ordering obtained by the barycentre algorithm. It is set to FALSE by default.
horiz	a Boolean argument for vertical (default) or horizontal layout.
max.iter	an integer stating the maximum number of runs of the barycentre heuristic on both layers of the bi-graph.
node.col	defines the colour of nodes from both layers.
edge.col	sets the colour of the edges.
...	further graphical parameters can be passed to the function.

Details

The one-to-one mapping between groups of clusters from two different flat partitionings is computed with the greedy algorithm: firstly, for each node the edge with the highest weight is taken, and secondly, the connected components in the edge-reduced bi-graph are found, so that each connected component corresponds to a pair of superclusters with a large overlap.

Value

a list containing:

s.clustering1	a vector indicating the supercluster in which each point is allocated in the first superclustering.
s.clustering2	a vector indicating the supercluster in which each point is allocated in the second superclustering.
merging1	a list of p elements, whose j-th component contains the labels of the initial clusters from the first partitioning that have been merged to produce the j-th supercluster in the left layer of the bi- graph.
merging2	a list of p elements, whose j-th component contains the labels of the initial clusters from the second partitioning that have been merged to produce the j-th supercluster in the right layer of the bi- graph.
weights	a pxp matrix containing the size of the intersections between the superclusters.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

barycentre, flatVSflat, flatVShier

Examples

```
### computation and visualisation of superclusters
    # simulated data
    clustering1 <- c(rep(1, 5), rep(2, 10), rep(3, 10))
    clustering2 <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
    mapping <- SCmapping(clustering1, clustering2, horiz = TRUE)
```

`score.crossing` *Computation of the aesthetics-based score of the parent and the children trees*

Description

`score.crossing` computes the value of the scoring function based on the aesthetics of the bigraph formed when comparing a dendrogram and a flat clustering, for both the parent-tree and the children-tree; the children-tree consists of the same branches as the parent-tree, except for the parent node, that has been split and replaced by some of its descendants.

Usage

```
score.crossing(weight.1, weight.2, N.cross)
```

Arguments

`weight.1`	a matrix of dimension `(m-1)xn` containing the intersection sizes (edge weights) between branches in the parent-tree and clusters from the flat partitioning. The ordering of the rows and columns is irrelevant for the computation of the score.
`weight.2`	a matrix of dimension `(m+k)xn` containing the intersection sizes (edge weights) between branches in the children-tree and clusters from the flat partitioning. `k` takes on values in 0,1,...,L, where L is the maximum number of steps that the comparison algorithm is allowed to look ahead. The ordering of the rows corresponding to branches that are not descendants of the parent node must coincide with that of the matrix `weight.1` after discarding the parent node. The ordering of the columns is irrelevant for the computation of the score.
`N.cross`	the number of edge crossings induced in the subtree formed by the descendants of the parent-node.

Details

The decision to split a given parent-node is based on achieving a better score for the children-tree than for the parent-tree. In the case of `score.crossing`, a better score is reflected by a larger value of the scoring function, which rewards few thicker edges, penalises many smaller edges, and accounts for the number of edge crossings in the resulting bigraph. The descendants of the parent-node considered in the children-tree are its two children if no look-ahead is carried out; otherwise, the descendants will reach subsequent generations and their number will increase by one at each look-ahead step.

Value

a list containing the following components:

`sc1`	the value of the scoring function for the parent-tree.
`sc2`	the value of the scoring function for the children-tree.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualizing relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

score.it, dyn.cross, flatVShier

Examples

```
### simulated data
parent.clustering <- c(rep("B1", 5), rep("B2", 10), rep("B3", 10))
# replace the branch 'B2' by its children 'B4' and 'B5'
children.clustering <- c(rep("B1", 5), rep("B4", 3), rep("B5", 7),
    rep("B3", 10))
flat.clustering <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
# the ordering of flat clusters is '1','2','3' and '4'.
parent.weights <- table(parent.clustering, flat.clustering)
children.weights <- table(children.clustering, flat.clustering)
descendant.cross <- dyn.cross(children.weights[c('B4', 'B5'), ])
score.crossing(parent.weights, children.weights, descendant.cross)
## better score for the parent.tree
```

---

score.it *Computation of the information theoretic-based score of the parent and the children trees*

Description

`score.it` computes the value of the scoring function based on information theory and the mutual information shared by a dendrogram and the flat clustering which is compared to, for both the parent-tree and the children-tree; the children-tree consists of the same branches as the parent-tree, except for the parent node, that has been split and replaced by some of its descendants.

Usage

```
score.it(weight.1, weight.2)
```

Arguments

`weight.1`	a matrix of dimension `(m-1)xn` containing the intersection sizes (edge weights) between branches in the parent-tree and clusters from the flat partitioning. The ordering of the rows and columns is irrelevant for the computation of the score.
`weight.2`	a matrix of dimension `(m+k)xn` containing the intersection sizes (edge weights) between branches in the children-tree and clusters from the flat partitioning. `k` takes on values in 0,1,...,L, where L is the maximum number of steps that the comparison algorithm is allowed to look ahead. The ordering of the rows corresponding to branches that are not descendants of the parent node must coincide with that of the matrix `weight.1` after discarding the parent node. The ordering of the columns is irrelevant for the computation of the score.

Details

The decision to split a given parent-node is based on achieving a better score for the children-tree than for the parent-tree. In the case of `score.it`, a better score is reflected by a smaller value of the scoring function, which is related to the average length of the messages that encode the information about one clustering contained in the other. The descendants of the parent-node considered in the children-tree are its two children if no look-ahead is carried out; otherwise, the descendants will reach subsequent generations and their number will increase by one at each look-ahead step.

Value

a list containing the following components:

`sc1`	the value of the scoring function for the parent-tree.
`sc2`	the value of the scoring function for the children-tree.

Author(s)

Aurora Torrente <aurora@ebi.ac.uk> and Alvis Brazma <brazma@ebi.ac.uk>

References

Torrente, A. *et al.* (2005). A new algorithm for comparing and visualising relationships between hierarchical and flat gene expression data clusterings. *Bioinformatics*, 21 (21), 3993-3999.

See Also

score.crossing, flatVShier

Examples

```
### simulated data
parent.clustering <- c(rep(1, 5), rep(2, 10), rep(3, 10))
# replace the branch '2' by children '4' and '5'
children.clustering<-c(rep(1,5),rep(4,3),rep(5,7),rep(3,10))
flat.clustering <- c(rep(1, 6), rep(2, 6), rep(3, 4), rep(4, 9))
score.it(table(parent.clustering, flat.clustering),
    table(children.clustering, flat.clustering))
## better score for the parent.tree
```

Index

∗ **branch split**
insert, 12
score.crossing, 15
score.it, 16
∗ **clustering comparison**
drawTreeGraph, 3
flatVSflat, 7
flatVShier, 9
SCmapping, 13
∗ **edge crossing**
barycentre, 2
dyn.cross, 6

barycentre, 2

drawTreeGraph, 3
dyn.cross, 6

flatVSflat, 7
flatVShier, 9

insert, 12

SCmapping, 13
score.crossing, 15
score.it, 16