Package 'CliquePercolation'

July 21, 2025

Version 0.4.0

Date 2022-11-08

Title Clique Percolation for Networks

Description Clique percolation community detection for weighted and unweighted networks as well as threshold and plotting functions. For more information see Farkas et al. (2007) <doi:10.1088/1367-2630/9/6/180> and Palla et al. (2005) <doi:10.1038/nature03607>.

Maintainer Jens Lange <lange.jens@outlook.com>

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

Imports colorspace, graphics, igraph, magrittr, Matrix, methods, Polychrome, qgraph, stats, utils, parallel, lessR, ohenery, pbapply

RoxygenNote 7.2.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Jens Lange [aut, cre], Janis Zickfeld [ctb], Alexander P. Christensen [ctb], Pedro Henrique Ribeiro Santiago [ctb]

Repository CRAN

Date/Publication 2022-11-09 08:40:37 UTC

Contents

`cpAlgorithm` *Clique Percolation Community Detection*

Description

Function for clique percolation community detection algorithms for weighted and unweighted networks.

Usage

```
cpAlgorithm(W, k, method = c("unweighted", "weighted", "weighted.CFinder"), I)
```

Arguments

`W`	A qgraph object or a symmetric matrix; see also qgraph
`k`	Clique size (number of nodes that should form a clique)
`method`	A string indicating the method to use (`"unweighted"`, `"weighted"`, or `"weighted.CFinder"`); see Details
`I`	Intensity threshold for weighted networks

Details

`method = "unweighted"` conducts clique percolation for unweighted networks as described in Palla et al. (2005). `method = "weighted"` conducts clique percolation for weighted graphs with inclusion of cliques if their Intensity is higher than the specified Intensity (`I`), which is the method described in Farkas et al. (2007). `method = "weighted.CFinder"` conducts clique percolation as in the CFinder program. The Intensity (`I`) threshold is applied twice, namely first to the Intensity of the cliques (as before) and then also to their `k-1` overlap with other cliques (e.g., in the case of `k = 3`, it is applied to the edge that two cliques share).

For weighted networks, the absolute value of the edge weights is taken. Therefore, negative edges are treated like positive edges just like in the CFinder program. Thus, the Intensity threshold `I` can only be positive.

cpAlgorithm produces a solution for all networks, even if there are no communities or communities have no overlap. The respective output is empty in such cases.

Value

A list object with the following elements:

list.of.communities.numbers list of communities with numbers as identifiers of nodes

list.of.communities.labels list of communities with labels from qgraph object or row or column names of matrix as identifiers of nodes

shared.nodes.numbers vector with all nodes that belong to multiple communities with numbers as identifiers of nodes

shared.nodes.labels vector with all nodes that belong to multiple communities with labels from qgraph object or row or column names of matrix as identifiers of nodes

isolated.nodes.numbers vector with all nodes that belong to no community with numbers as identifiers of nodes

isolated.nodes.labels vector with all nodes that belong to no community with labels from qgraph object or row or column names of matrix as identifiers of nodes

k user-specified `k`

method user-specified method

I user-specified `I` (if method was `"weighted"` or `"weighted.CFinder"`)

Author(s)

Jens Lange, <lange.jens@outlook.com>

References

Farkas, I., Abel, D., Palla, G., & Vicsek, T. (2007). Weighted network modules. *New Journal of Physics, 9*, 180-180. http://doi.org/10.1088/1367-2630/9/6/180

Palla, G., Derenyi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature, 435*, 814-818. http://doi.org/10.1038/nature03607

Examples

```
## Example for unweighted networks

# create qgraph object
W <- matrix(c(0,1,1,1,0,0,0,0,
              0,0,1,1,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,1,1,1,0,
              0,0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,0,1,0,
              0,0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0), nrow = 8, ncol = 8, byrow = TRUE)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# run clique percolation for unweighted networks
results <- cpAlgorithm(W = W, k = 3, method = "unweighted")
```

```
# print results overview
results

# extract more information about communities
summary(results)

## Example for weighted networks

# create qgraph object
W <- matrix(c(0,1,1,1,0,0,0,0,
              0,0,1,1,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,1,1,1,0,
              0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,1,0,
              0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0), nrow = 8, ncol = 8, byrow = TRUE)
set.seed(4186)
rand_w <- stats::rnorm(length(which(W == 1)), mean = 0.3, sd = 0.1)
W[which(W == 1)] <- rand_w
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# run clique percolation for weighted networks
results <- cpAlgorithm(W = W, k = 3, method = "weighted", I = 0.1)

# print results overview
results

# extract more information about communities
summary(results)

## Example with Obama data set (see ?Obama)

# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# run clique percolation algorithm with specific k and I
cpk3I.16 <- cpAlgorithm(net, k = 3, I = 0.16, method = "weighted")

# print results overview
cpk3I.16

# extract more information about communities
summary(cpk3I.16)
```

cpColoredGraph *Colored Network According To Clique Percolation Communities*

Description

Function for plotting the original network with nodes colored according to the community partition identified via the clique percolation community detection algorithm, taking predefined sets of nodes into account.

Usage

```
cpColoredGraph(
  W,
  list.of.communities,
  list.of.sets = NULL,
  larger.six = FALSE,
  h.cp = c(0, 360 * (length(cplist) - 1)/length(cplist)),
  c.cp = 80,
  l.cp = 60,
  set.palettes.size = NULL,
  own.colors = NULL,
  avoid.repeated.mixed.colors = FALSE,
  ...
)
```

Arguments

`W`	A qgraph object or a symmetric matrix; see also qgraph
`list.of.communities`	List object taken from results of cpAlgorithm function; see also cpAlgorithm
`list.of.sets`	List object specifying predefined groups of nodes in original network; default is NULL; see Details
`larger.six`	Integer indicating whether `length(list.of.communities)` is larger six (if `list.of.sets = NULL`) or `length(list.of.sets)` is larger six (if `list.of.sets` is not NULL); default is FALSE; see Details
`h.cp`	Vector of integers indicating the range of hue from which colors should be drawn for elements in `list.of.communities` (if `list.of.sets = NULL`) or for elements in `list.of.sets` (if `list.of.sets` is not NULL); default is the value specified in colorspace::qualitative_hcl(); see Details
`c.cp`	Integer indicating the chroma from which colors should be drawn for elements in `list.of.communities` (if `list.of.sets = NULL`) or for elements in `list.of.sets` (if `list.of.sets` is not NULL); default is 80 as specified in colorspace::qualitative_hcl(); see Details
`l.cp`	Integer indicating the luminance from which colors should be drawn for elements in `list.of.communities` (if `list.of.sets = NULL`) or for elements in

list.of.sets (if list.of.sets is not NULL); default is 60 as specified in colorspace::qualitative_hcl(); see Details

set.palettes.size
: Integer indicating the number of sets for which smooth gradients of a set color should be generated using colorspace::sequential_hcl(); default is the number of pure communities of a set plus one; see Details

own.colors
: Vector of hex code colors of length of list.of.communities (if list.of.sets = NULL) or list.of.sets (if list.of.sets is not NULL); if specified, colors are used for coloring the communities and no other colors are generated; if NULL (default), reasonable colors are generated; see Details

avoid.repeated.mixed.colors
: Logical indicating whether it should be avoided that multiple mixed communities are assigned the same color; default is FALSE; see Details

...
: any additional argument from qgraph; see also qgraph

Details

The function takes the results of cpAlgorithm (see also cpAlgorithm), that is, either the list.of.communities.numbers or the list.of.communities.labels and plots the original network, coloring the nodes according to the community partition. If there are no predefined sets of nodes (list.of.sets = NULL; the default), each community is assigned a color by using a palette generation algorithm from the package colorspace, which relies on HCL color space. Specifically, the function qualitative_hcl (see also colorspace::qualitative_hcl() is used, which generates a balanced set of colors over a range of hue values, holding chroma and luminance constant. This method is preferred over other palette generating algorithms in other color spaces (Zeileis et al., subm.). The default values recommended in qualitative_hcl are used, adapted to the current context in the case of hue. Yet, h.cp, c.cp, and l.cp can be used to overwrite the default values. Each node gets the color of the community it belongs to. Shared nodes are split equally in multiple colors, one for each community they belong to. Isolated nodes are colored white.

If there are predefined sets of nodes, the qualitatively different colors are assigned to the sets specified in list.of.sets. Then, it is checked whether communities are pure (they contain nodes from only one set) or they are mixed (they contain nodes from multiple sets). For pure communities of each set, the assigned color is taken and faded towards white with another function from colorspace, namely sequential_hcl (see also colorspace::sequential_hcl(). For instance, if there are three pure communities with nodes that are only from Set 1, then the basic color assigned to Set 1 is taken, and faded toward white in 3 + 1 steps. There is one color generated more than needed (here four colors for three communities), because the last color in the fading is always white, which is reserved for isolated nodes. The three non-white colors are then assigned to each community, with stronger colors being assigned to larger communities. In that sense, all communities that entail nodes from only one specific set, will have rather similar colors (only faded towards white). All communities that entail nodes from only one specific other set, will also have similar colors, yet they will differ qualitatively from the colors of the communities that entail items from other sets. For communities that entail items from multiple sets, the basic colors assigned to these sets are mixed in proportion to the number of nodes from each set. For instance, if a community entails two nodes from Set 1 and one node from Set 2, then the colors of Sets 1 and 2 are mixed 2:1.

The mixing of colors is subtractive (how paint mixes). Subtractive color mixing is difficult to implement. An algorithm proposed by Scott Burns is used (see http://scottburns.us/subtractive-color-mixture/ and http://scottburns.us/fast-rgb-to-spectrum-conversion-for-reflectances/). Each color is

transformed into a corresponding reflectance curve via the RGBC algorithm. That is, optimized reflectance curves of red, green, and blue are adapted according to the RGB values of the respective color. The reflectance curves of the colors that need to be mixed are averaged via the weighted geometric mean. The resulting mixed reflectance curve is transformed back to RGB values by multiplying the curve with a derived matrix. The algorithm produces rather good color mixing and is computationally efficient. Yet, results may not always be absolutely precise.

The mixing of colors for mixed communities can lead to multiple communities being assigned the same color. For instance, two communities with two nodes each from Sets 1 and 2 would have the same color, namely the colors assigned to the sets mixed in the same proportion. This is reasonable, because these communities are structurally similar. However, it can be confusing to have two actually different communities with the same color. To avoid this, set `avoid.repeated.mixed.colors = TRUE`. Doing so slightly alters the ratio with which the color of a mixed community is determined, if the community would have been assigned a color that was already assigned. This slight variation of the ratio is random. To reproduce results from a previous run, set a seed.

The fading of pure communities via sequential_hcl is a function of the number of sets. If there are more pure communities from a specific set, more faded colors will be generated. This makes coloring results hard to compare across different networks, if such a comparison is desired. For instance, if one network has 12 nodes that belong to three communities sized 6, 3, and 3, all of them pure (having nodes from only one set), then their colors will be strong, average, and almost white respectively. If the same 12 nodes belong to two communities size 6 and 6, both of them pure, then their colors will be strong and average to almost white. Different numbers of pure communities therefore change the color range. To circumvent that, one can specify `set.palettes.size` to any number larger than the number of pure communities of a set plus one. For all sets, sequential_hcl then generates as many shades towards white of a respective color as specified in `set.palettes.size`. Colors for each community are then picked from the strongest towards whiter colors, with larger communities being assigned stronger colors. Note that in this situation, the range of colors is always the same for all sets in a network, making them comparable across different sets. When there are more pure communities of one set than from another their luminance will be lower. Moreover, also across networks, the luminance of different sets of nodes or of the same set can be compared.

In all cases, qualitatively different colors are assigned to either the elements in `list.of.communities` (when `list.of.sets = NULL`) or the elements in `list.of.sets` (when `list.of.sets` is not NULL) with qualitative_hcl. Zeileis et al. (subm.) argue that this function can generate up to six different colors that people can still distinguish. For a larger number of qualitative colors, other packages can be used. Specifically, if the argument `larger.six = TRUE` (default is `FALSE`), the qualitatively different colors are generated via the package Polychrome (Coombes et al., 2019) with the function createPalette (see also createPalette). This function generates maximally different colors in HCL space and can generate a higher number of distinct colors. With these colors, the rest of the procedure is identical. The seedcolors specified in Polychrome are general red, green, and blue. As the procedure relies on randomness, you have to set a seed to reproduce the results of a previous run. Note that the Polychrome palettes are maximally distinct, thus they are most likely not as balanced as the palettes generated with colorspace. In general, the function cpColoredGraph is recommended only for very small networks anyways, for which `larger.six = FALSE` makes sense. For larger networks, consider plotting the community network instead (see cpCommunityGraph).

When own.colors are specified, these colors are assigned to the elements in `list.of.communities` (if `list.of.sets = NULL`) or to the elements in `list.of.sets` (if `list.of.sets` is not NULL). The rest of the procedure is identical.

Value

The function primarily plots the original network and colors the nodes according to the communities, taking predefined sets into account. Additionally, it returns a list with the following elements:

basic.colors.sets Vector with colors assigned to the elements in `list.of.sets`, if `list.of.sets` is not NULL. Otherwise NULL is returned.

colors.communities Vector with colors of the communities, namely assigned colors if `list.of.sets = NULL` or shaded and mixed colors if `list.of.sets` is not NULL.

colors.nodes List with all colors assigned to each node. Isolated nodes are white. Shared nodes have a vector of colors from each community they belong to.

Author(s)

Jens Lange, <lange.jens@outlook.com>

References

Coombes, K. R., Brock, G., Abrams, Z. B., & Abruzzo, L. V. (2019). Polychrome: Creating and assessing qualitative palettes with many colors. *Journal of Statistical Software, 90*, 1-26. https://doi.org/10.18637/jss.v090.c01

Zeileis, A., Fisher, J. C., Hornik, K., Ihaka, R., McWhite, C. D., Murrell, P., Stauffer, R., & Wilke, C. O. (subm.). *colorspace: A toolbox for manipulating and assessing colors and palettes*. https://arxiv.org/abs/1903.06490

Examples

```
## Example with fictitious data

# generate qgraph object with letters as labels
W <- matrix(c(0,0.10,0,0,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0.10,0,0,0.10,0.20,0,0,0,0,0.20,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0.10,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0.10,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0.20,0,0,0,0,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0.20,0,0,0,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0.20,0,0,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0.20,0,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0.20,0.20,0.30,0,0,0,0,0.30,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0.20,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0.30,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), nrow = 21, ncol = 21, byrow = TRUE)
```

```
W <- Matrix::forceSymmetric(W)
rownames(W) <- letters[seq(from = 1, to = nrow(W))]
colnames(W) <- letters[seq(from = 1, to = ncol(W))]

W <- qgraph::qgraph(W, layout = "spring", edge.labels = TRUE)

# run clique percolation algorithm; three communities; two shared nodes, one isolated node
cp <- cpAlgorithm(W, k = 3, method = "weighted", I = 0.09)

# color original graph according to community partition
# all other arguments are defaults; qgraph arguments used to return same layout

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          layout = "spring", edge.labels = TRUE)


# own colors (red, green, and blue) assigned to the communities

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          own.colors = c("#FF0000","#00FF00","#0000FF"),
                          layout = "spring", edge.labels = TRUE)


# define sets of nodes; nodes a to o are in Set 1 and letters p to u in Set 2
list.of.sets <- list(letters[seq(from = 1, to = 15)],
                     letters[seq(from = 16, to = 21)])

# color original graph according to community partition, taking sets of nodes into account
# two communities are pure and therefore get shades of set color; smaller community is more white
# one community is mixed, so both set colors get mixed

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          list.of.sets = list.of.sets,
                          layout = "spring", edge.labels = TRUE)


# graph as before, but specifying the set palette size to 6
# from a range of 6 colors, the pure communities get the darker ones
# in a different network with also two pure communities, luminance would therefore be equal

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          list.of.sets = list.of.sets, set.palettes.size = 6,
                          layout = "spring", edge.labels = TRUE)


# graph as before, but colors sampled only form yellow to blue range, less chroma, more luminance

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          list.of.sets = list.of.sets, set.palettes.size = 6,
                          h.cp = c(50, 210), c.cp = 70, l.cp = 70,
                          layout = "spring", edge.labels = TRUE)
```

```
# own colors (red and green) assigned to the sets
# two communities in shades of red and one community is mix of green and red (brown)

results <- cpColoredGraph(W, list.of.communities = cp$list.of.communities.labels,
                          list.of.sets = list.of.sets,
                          own.colors = c("#FF0000","#00FF00"),
                          layout = "spring", edge.labels = TRUE)


## Example with Obama data set (see ?Obama)

# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# run clique percolation algorithm with specific k and I
cpk3I.16 <- cpAlgorithm(net, k = 3, I = 0.16, method = "weighted")

# color original graph according to community partition
# all other arguments are defaults

results <- cpColoredGraph(net, list.of.communities = cpk3I.16$list.of.communities.labels,
                          layout = "spring", theme = "colorblind")
```

---

cpCommunityGraph *Plotting Clique Percolation Community Network*

Description

Function for plotting a network with nodes representing communities from clique percolation community detection and edges representing the number of shared nodes of the communities.

Usage

```
cpCommunityGraph(
  list.of.communities,
  node.size.method = c("proportional", "normal"),
  max.node.size = 10,
  ...
)
```

Arguments

`list.of.communities`
: List object taken from results of cpAlgorithm function; see also cpAlgorithm

`node.size.method`
: String indicating how node size is plotted (`"proportional"` or `"normal"`); see Details

`max.node.size`
: Integer indicating size of the node representing the largest community, if `node.size.method = "proportional"`

`...`
: any additional argument from qgraph; see also qgraph

Details

The function takes the results of cpAlgorithm (see also cpAlgorithm), that is, either the `list.of.communities.numbers` or the `list.of.communities.labels` and plots the community network. Each node represents a community. Edges connecting two nodes represent the number of shared nodes between the two communities.

The nodes can be plotted proportional to the sizes of the communities (`node.size.method = "proportional"`). The node representing the largest community is then plotted with the size specified in `max.node.size`. All other nodes are plotted relative to this largest node. Alternatively, all nodes can have the same size (`node.size.method = "normal"`).

For the plotting, all isolated nodes will be ignored. If there are less than two communities in the list, plotting the network is useless. Therefore, an error is printed in this case.

Value

The function primarily plots the community network. Additionally, it returns a list with the weights matrix (`community.weights.matrix`) of the community network.

Author(s)

Jens Lange, <lange.jens@outlook.com>

Examples

```
## Example with fictitious data

# create qgraph object
W <- matrix(c(0,1,1,0,0,0,0,
              0,0,1,0,0,0,0,
              0,0,0,1,1,1,0,
              0,0,0,0,1,1,0,
              0,0,0,0,0,1,0,
              0,0,0,0,0,0,1,
              0,0,0,0,0,0,0), nrow = 7, ncol = 7, byrow = TRUE)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# run clique percolation for unweighted networks
cp.results <- cpAlgorithm(W = W, k = 3, method = "unweighted")

# plot community network; proportional; maximum size is 7
cp.network1 <- cpCommunityGraph(cp.results$list.of.communities.numbers,
                                node.size.method = "proportional",
```

```
                                max.node.size = 7)

# plot community network; proportional; maximum size is 7
# change shape of nodes to triangle via qgraph argument
cp.network2 <- cpCommunityGraph(cp.results$list.of.communities.numbers,
                                node.size.method = "proportional",
                                max.node.size = 7,
                                shape = "triangle")

## Example with Obama data set (see ?Obama)

# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# run clique percolation algorithm with specific k and I
cpk3I.16 <- cpAlgorithm(net, k = 3, I = 0.16, method = "weighted")

# plot community network; normal
Obama.network <- cpCommunityGraph(cpk3I.16$list.of.communities.numbers,
                                  node.size.method = "proportional",
                                  theme = "colorblind")
```

`cpCommunitySizeDistribution` *Plotting Clique Percolation Community Size Distribution*

Description

Function for plotting the frequency distribution of community sizes from clique percolation community detection and testing for power-law.

Usage

```
cpCommunitySizeDistribution(
  list.of.communities,
  color.line = "#bc0031",
  test.power.law = FALSE
)
```

Arguments

`list.of.communities`	List object taken from results of cpAlgorithm function; see also cpAlgorithm
`color.line`	string indicating the color of the line in the plot as described in par; default is `"#bc0031"`

`test.power.law` Logical indicating whether fit of power-law should be tested; default is FALSE; see Details

Details

The function takes the results of cpAlgorithm (see also cpAlgorithm), that is, either the `list.of.communities.numbers` or the `list.of.communities.labels` and plots the community size distribution. If there are no communities, no plot can be generated. An error is printed indicating this.

If `test.power.law = TRUE`, test of a fit of a power-law is performed with the function fit_power_law (see also fit_power_law). Fit is tested for the entire distribution from the smallest community size onward (i.e., typically k as specified in cpAlgorithm). Moreover, test uses the `plfit` implementation of fit_power_law. For other arguments, default values are used.

Value

The function primarily plots the community size distribution. Additionally, it returns a list with a data frame containing all community sizes and their frequencies (`size.distribution`). If `test.power.law = TRUE`, a test of fit of a power-law distribution is also returned as a list object with results from fit_power_law (see also fit_power_law).

Author(s)

Jens Lange, <lange.jens@outlook.com>

Examples

```
## Example with fictitious data

# create qgraph object; 150 nodes; 1/7 of all edges are different from zero
W <- matrix(c(0), nrow = 150, ncol = 150, byrow = TRUE)
set.seed(4186)
W[upper.tri(W)] <- sample(c(rep(0,6),1), length(W[upper.tri(W)]), replace = TRUE)
rand_w <- stats::rnorm(length(which(W == 1)), mean = 0.3, sd = 0.1)
W[which(W == 1)] <- rand_w
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W, DoNotPlot = TRUE)

# run clique percolation for weighted networks
cp.results <- cpAlgorithm(W, k = 3, method = "weighted", I = 0.38)

# plot community size distribution with blue line
cp.size.dist <- cpCommunitySizeDistribution(cp.results$list.of.communities.numbers,
                                            color.line = "#0000ff")
# test for power-law distribution
cp.size.dist <- cpCommunitySizeDistribution(cp.results$list.of.communities.numbers,
                                            color.line = "#0000ff",
                                            test.power.law = TRUE)
cp.size.dist$fit.power.law

## Example with Obama data set (see ?Obama)
```

```
# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# run clique percolation algorithm with specific k and I
cpk3I.16 <- cpAlgorithm(net, k = 3, I = 0.16, method = "weighted")

# plot community size distribution
#the distribution is not very informative with four equally-sized communities
Obama.size.dist <- cpCommunitySizeDistribution(cpk3I.16$list.of.communities.numbers)
```

`cpPermuteEntropy` *Confidence Intervals Of Entropy Based On Random Permutations Of Network*

Description

Function for determining confidence intervals of entropy values calculated for community partition from clique percolation based on randomly permuted networks of original network.

Usage

```
cpPermuteEntropy(
  W,
  cpThreshold.object,
  n = 100,
  interval = 0.95,
  CFinder = FALSE,
  ncores,
  seed = NULL
)
```

Arguments

`W`	A qgraph object or a symmetric matrix; see also qgraph
`cpThreshold.object`	A cpThreshold object; see also cpThreshold
`n`	number of permutations (default is 100)
`interval`	requested confidence interval (larger than zero and smaller 1; default is 0.95)
`CFinder`	logical indicating whether clique percolation for weighted networks should be performed as in CFinder ; see also cpAlgorithm
`ncores`	Numeric. Number of cores to use in computing results. Defaults to `parallel::detectCores() / 2` or half of your computer's processing power. Set to `1` to not use parallel computing
`seed`	Numeric. Set seed for reproducible results. Defaults to `NULL`

Details

The function generates n random permutations of the network specified in W. For each randomly permuted network, it runs cpThreshold (see cpThreshold for more information) with k and I values extracted from the cpThreshold object specified in cpThreshold.object. Across permutations, the confidence intervals of the entropy values are determined for each k separately.

The confidence interval of the entropy values is determined separately for each k. This is because larger k have to produce less communities on average, which will decrease entropy. Comparing confidence intervals of smaller k to those of larger k would therefore be disadvantageous for larger k.

In the output, one can check the confidence intervals of each k. Moreover, a data frame is produced that takes the cpThreshold object that was specified in cpThreshold.object and removes all rows that do not exceed the upper bound of the confidence interval of the respective k.

Value

A list object with the following elements:

Confidence.Interval a data frame with lower and upper bound of confidence interval for each k

Extracted.Rows rows extracted from cpThreshold.object that are larger than the upper bound of the specified confidence interval for each k

Settings user-specified settings

Author(s)

Jens Lange, <lange.jens@outlook.com>

Examples

```
## Example with fictitious data

# create qgraph object
W <- matrix(c(0,1,1,1,0,0,0,0,
              0,0,1,1,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,1,1,1,0,
              0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,1,0,
              0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0), nrow = 8, ncol = 8, byrow = TRUE)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# create cpThreshold object
cpThreshold.object <- cpThreshold(W = W, method = "unweighted", k.range = c(3,4),
                                  threshold = "entropy")

# run cpPermuteEntropy with 100 permutations and 95% confidence interval

results <- cpPermuteEntropy(W = W, cpThreshold.object = cpThreshold.object,
```

```
                            n = 100, interval = 0.95, ncores = 1, seed = 4186)

# check results
results


## Example with Obama data set (see ?Obama)

# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# create cpThreshold object

threshold <- cpThreshold(net, method = "weighted",
                         k.range = 3:4,
                         I.range = seq(0.1, 0.5, 0.01),
                         threshold = "entropy")


# run cpPermuteEntropy with 50 permutations and 99% confidence interval

permute <- cpPermuteEntropy(net, cpThreshold.object = threshold,
                            interval = 0.99, n = 50, ncores = 1, seed = 4186)

# check results
permute
```

---

cpThreshold *Optimizing* k *And* I *For Clique Percolation Community Detection*

Description

Function for determining threshold value(s) (ratio of largest to second largest community sizes, chi, entropy, fuzzy modularity, signed fuzzy modularity) of ranges of k and I values to help deciding for optimal k and I values.

Usage

```
cpThreshold(
  W,
  method = c("unweighted", "weighted", "weighted.CFinder"),
  k.range,
  I.range,
  threshold = c("largest.components.ratio", "chi", "entropy", "fuzzymod",
    "signedfuzzymod")
)
```

Arguments

W	A qgraph object or a symmetric matrix; see also qgraph
method	A string indicating the method to use ("unweighted", "weighted", or "weighted.CFinder"). See cpAlgorithm for more information
k.range	integer or vector of k value(s) for which threshold(s) are determined See cpAlgorithm for more information
I.range	integer or vector of I value(s) for which threshold(s) are determined See cpAlgorithm for more information
threshold	A string or vector indicating which threshold(s) to determine ("largest.components.ratio", "chi", "entropy", "fuzzymod", "signedfuzzymod"); see Details

Details

Optimizing k (clique size) and I (Intensity threshold) in clique percolation community detection is a difficult task. Farkas et al. (2007) recommend to look at the ratio of the largest to second largest community sizes (threshold = "largest.components.ratio") for very large networks or the variance of the community sizes when removing the community size of the largest community (threshold = "chi") for somewhat smaller networks. These thresholds were derived from percolation theory. If I for a certain k is too high, no community will be identified. If I is too low, a giant community with all nodes emerges. Just above this I, the distribution of community sizes often follows a power law, which constitutes a broad community sizes distribution. Farkas et al. (2007) point out, that for such I, the ratio of the largest to second largest community sizes is approximately 2, constituting one way to optimize I for each possible k. For somewhat smaller networks, the ratio can be rather unstable. Instead, Farkas et al. (2007, p.8) propose to look at the variance of the community sizes after removing the largest community. The idea is that when I is rather low, one giant community and multiple equally small ones occur. Then, the variance of the community sizes of the small communities (removing the giant community) is low. When I is high, only a few equally small communities will occur. Then, the variance of the community sizes (after removing the largest community) will also be low. In between, the variance will at some point be maximal, namely when the community size distribution is maximally broad (power law-distributed). Thus, the maximal variance could be used to optimize I for various k.

For very small networks, optimizing k and I based on the distribution of the community sizes will be impossible, as too few communities will occur. Another possible threshold for such networks is based on the entropy of the community sizes (threshold = "entropy"). Entropy can be interpreted as an indicator of how surprising the respective solution is. The formula used here is based on Shannon Information, namely

$$-\sum_{i=1}^{N} p_i * \log_2 p_i$$

with p_i being the probability that a node is part of community i. For instance, if there are two communities, one of size 5 and one of size 3, the result would be

$$-((5/8 * \log_2 5/8) + (3/8 * \log_2 3/8)) = 1.46$$

When calculating entropy, the isolated nodes identified by clique percolation are treated as a separate community. If there is only one community or only isolated nodes, entropy is zero, indicating that the surprisingness is low. As compared to the ratio and chi thresholds, entropy favors communities that are equal in size. Thus, it should not be used for larger networks for which a broader

community size distribution is preferred. Note that the entropy threshold has not been validated for clique percolation as of now. Initial simulation studies indicate that it consistently detects surprising community partitions in smaller networks especially if there are cliques of larger k.

Santiago et al. (2022) recently proposed in a simulation study that two alternative metrics, fuzzy modularity and signed fuzzy modularity, showed good performance in recovering the true community assignment in psychological networks with overlapping nodes and can also be used to optimize k (clique size) and I (Intensity threshold). See FuzzyMod and SignedFuzzyMod for more information.

Ratio thresholds can be determined only if there are at least two communities. Chi threshold can be determined only if there are at least three communities. If there are not enough communities for the respective threshold, their values are NA in the data frame. Entropy, fuzzy modularity, and signed fuzzy modularity can always be determined.

Value

A data frame with columns for k, I (if method = "weighted" or method = "weighted.CFinder"), number of communities, number of isolated nodes, and results of the specified threshold(s).

Author(s)

Jens Lange, <lange.jens@outlook.com>

References

Farkas, I., Abel, D., Palla, G., & Vicsek, T. (2007). Weighted network modules. *New Journal of Physics, 9*, 180-180. http://doi.org/10.1088/1367-2630/9/6/180

Santiago, P. H. R., Soares, G. H., Quintero, A., & Jamieson, L. (2022). *The performance of the Clique Percolation to identify overlapping symptoms in psychological networks*. PsyArXiv. https://psyarxiv.com/fk963/

Examples

```
## Not run: ## Example for unweighted networks

# create qgraph object
W <- matrix(c(0,1,1,1,0,0,0,0,
              0,0,1,1,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,1,1,1,0,
              0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,1,0,
              0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0), nrow = 8, ncol = 8, byrow = TRUE)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# determine entropy and fuzzy modularity thresholds for k = 3 and k = 4
results <- cpThreshold(W = W, method = "unweighted", k.range = c(3,4), threshold = c("entropy",
"fuzzymod"))
```

```
## Example for weighted networks; three large communities with I = 0.3, 0.2, and 0.1, respectively

# create qgraph object
W <- matrix(c(0,0.10,0,0,0,0,0.10,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0.10,0,0,0,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0.10,0,0,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0.10,0,0,0.10,0.20,0,0,0,0,0.20,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0.10,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0.10,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0.10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0.20,0,0,0,0,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0.20,0,0,0,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0.20,0,0,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0.20,0,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0.20,0.20,0.30,0,0,0,0,0.30,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.20,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,0,0,0,0,0.30,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.30,
         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), nrow = 22, ncol = 22, byrow = TRUE)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W, layout = "spring", edge.labels = TRUE)

# determine ratio, chi, entropy, fuzzy modularity, and signed fuzzy modularity
# thresholds for k = 3 and I from 0.3 to 0.09
results <- cpThreshold(W = W, method = "weighted", k.range = 3,
                       I.range = c(seq(0.3, 0.09, by = -0.01)),
                       threshold = c("largest.components.ratio","chi","entropy",
                       "fuzzymod","signedfuzzymod"))

## Example with Obama data set (see ?Obama)

# get data
data(Obama)

# estimate network
net <- qgraph::EBICglasso(qgraph::cor_auto(Obama), n = nrow(Obama))

# determine entropy, fuzzy modularity, and signed fuzzy modularity thresholds
# for k from 3 to 4 and I from 0.1 to 0.5
threshold <- cpThreshold(net, method = "weighted",
                         k.range = 3:4,
                         I.range = seq(0.1, 0.5, 0.01),
                         threshold = c("entropy", "fuzzymod", "signedfuzzymod"))
## End(Not run)
```

`FuzzyMod` *Fuzzy Modularity of a community structure of a graph*

Description

Function calculates the fuzzy modularity of a (disjoint or non-disjoint division) of a graph into subgraphs.

Usage

```
FuzzyMod(graph, membership, abs = TRUE)
```

Arguments

`graph`	The input graph.
`membership`	Numeric vector or list indicating the membership structure.
`abs`	Should fuzzy modularity be calculated based on absolute values of network edges? Default is TRUE.

Details

The modularity of a graph with respect to some division is a measure of how good the division is. The traditional *modularity* Q was proposed by Newman and Girvan (2004):

$$Q = \frac{1}{2m} \sum_{c \epsilon_C} \sum_{u,v \epsilon_V} (A_{uv} - \frac{k_u k_v}{2m}) \delta_{cu} \delta_{cv}$$

where m is the total number of edges, C is the set of communities corresponding to a partition, V is the set of vertices (i.e. nodes) in the network, A_{uv} is the element of the A adjacency matrix in row i and column j, and k_u and k_v are the node degrees of nodes u and v, respectively. δ_{cu} indicates whether node u belongs to community c, which equals 1 if u and v belongs to community c and 0 otherwise. The product $\delta_{cu} * \delta_{cv}$ is a Kronecker delta function which equals 1 if u and v belongs to community c and 0 otherwise.

In the case of *weighted* networks, Fan, Li, Zhang, Wu, and Di (2007) proposed that to calculate *modularity* Q, m should be the total edge weights, and k_u and k_v should be the node strengths of nodes u and v, respectively.

One limitation of *modularity* Q proposed by Newman and Girvan (2004) was that modularity could not be calculated for non-disjoint community partitions (i.e. networks in which a node is assigned to more than one community). As such, Chen, Shang, Lv, and Fu (2010) proposed a generalisation in terms of fuzzy modularity:

$$Q = \frac{1}{2m} \sum_{c \epsilon_C} \sum_{u,v \epsilon_V} \alpha_{cu} \alpha_{cv} (A_{uv} - \frac{k_u k_v}{2m})$$

where α_{cu} is the *belonging coefficient*. The *belonging coefficient* reflects how much the node u belongs to community c. The belonging coefficient is calculated as:

$$\alpha_{cu} = \frac{k_{cu}}{\sum_{c \epsilon C} k_{cu}}$$

In case of a disjoint solution, the fuzzy modularity Q proposed by Chen, Shang, Lv, and Fu (2010) reduces to the modularity Q proposed by Newman and Girvan (2004).

Value

A numeric scalar, the fuzzy modularity score of the given configuration.

Author(s)

Pedro Henrique Ribeiro Santiago, <phrs16@gmail.edu.au> [ctb]

Gustavo Hermes Soares, [rev]

Adrian Quintero, [rev]

Lisa Jamieson, [rev]

References

Newman, M. E., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical review E*, 69(2), 026113.

Fan, Y., Li, M., Zhang, P., Wu, J., & Di, Z. (2007). Accuracy and precision of methods for community identification in weighted networks. *Physica A: Statistical Mechanics and its Applications*, 377(1), 363-372.

Chen, D., Shang, M., Lv, Z., & Fu, Y. (2010). Detecting overlapping communities of weighted networks via a local algorithm. *Physica A: Statistical Mechanics and its Applications*, 389(19), 4177-4187.

Examples

```
g <- igraph::disjoint_union(igraph::make_full_graph(5),igraph::make_full_graph(4))
g <- igraph::add_edges(g, c(2,6, 2,7, 2,8, 2,9))
wc <- list(c(1,2,3,4,5),c(2,6,7,8,9))
FuzzyMod(graph=g, membership=wc, abs=TRUE)
```

---

immuno *Data: Immunoglobulin interaction network*

Description

Unweighted, undirected network of interactions in the immunoglobulin network. The 1,316 nodes represent amino-acids and two nodes are connected by an edge if the shortest distance of their C_alpha atoms is smaller than $\Theta = 8$ Angstrom.

Usage

```
data(immuno)
```

Format

An object of class "qgraph" with 1,316 nodes and 6,300 edges.

Source

https://CRAN.R-project.org/package=igraphdata

References

Gfeller, D. (2007). *Simplifying complex networks: From a clustering to a coarse graining strategy*. EPFL. http://library.epfl.ch/theses/?nr=3888

Examples

```
data(immuno)
```

Obama *Data: Evaluative Reactions Toward Barack Obama (2012)*

Description

A data set containing evaluative reactions toward Barack Obama from the American National Election Studies in 2012. The study included 5,914 participants, representative of the adult U.S. American population (note missing values). The participants rated Obama on 10 evaluative reactions.

Usage

```
data(Obama)
```

Format

An object of class "data.frame" with 5,914 observations and 10 variables.

Mor "Is moral"

Led "Would provide strong leadership"

Car "Really cares about people like you"

Kno "Is knowledgeable"

Int "Is intelligent"

Hns "Is honest"

Ang "Angry"

Hop "Hopeful"

Afr "Afraid of him"

Prd "Proud"

Source

https://electionstudies.org/

References

Dalege, J., Borsboom, D., Van Harreveld, F., Van der Maas, H. L. J. (2017). Network analysis on attitudes: A brief tutorial. *Social Psychological and Personality Science, 8*, 528-537. https://doi.org/10.1177/1948550617709827

Examples

```
data(Obama)
```

`print.cpAlgorithm` *print.cpAlgorithm*

Description

Print method for objects of class `cpAlgorithm`.

Usage

```
## S3 method for class 'cpAlgorithm'
print(x, ...)
```

Arguments

`x`	An object of class `cpAlgorithm`; see also cpAlgorithm
`...`	currently ignored

Author(s)

Jens Lange, <lange.jens@outlook.com>

print.cpPermuteEntropy *print.cpPermuteEntropy*

Description

Print method for objects of class cpPermuteEntropy.

Usage

```
## S3 method for class 'cpPermuteEntropy'
print(x, ...)
```

Arguments

x	An object of class cpPermuteEntropy; see also cpPermuteEntropy
...	currently ignored

Author(s)

Jens Lange, <lange.jens@outlook.com>

SignedFuzzyMod *Signed Fuzzy Modularity of a community structure of a graph*

Description

Function calculates the fuzzy modularity of a (disjoint or non-disjoint division) of a graph into subgraphs for signed weighted networks.

Usage

```
SignedFuzzyMod(netinput, membassigned)
```

Arguments

netinput	The input graph.
membassigned	Numeric vector or list indicating the membership structure.

Details

For *signed* weighted networks (i.e. networks with positive and negative edges), the calculation of the modularity Q is problematic. Gomez, Jensen, and Arenas (2009) explain that, when calculating modularity Q for unweighted (Newman & Girvan, 2004) or weighted networks (Fan, Li, Zhang, Wu, & Di, 2007), the term $\frac{k_u}{2m}$ indicates the probability of node u making connections with other nodes in the network, if connections between nodes were random. Gomez, Jensen, and Arenas (2009) discuss how, when networks are signed, the positive and negative edges cancel each other out and the term $\frac{k_u}{2m}$ loses its probabilistic meaning. To deal with this limitation, Gomez, Jensen, and Arenas (2009) proposed modularity Q for signed weighted networks, generalised to fuzzy modularity Q for signed weighted networks:

$$Q = (\frac{2w^+}{2w^+ + 2w^-})(\frac{1}{2m^+})\sum_{c\epsilon_C}\sum_{u,v\epsilon_V}\alpha^+_{cu}\alpha^+_{cv}(A^+_{uv} - \frac{k^+_u k^+_v}{2m}) - (\frac{2w^-}{2w^+ + 2w^-})(\frac{1}{2m^-})\sum_{c\epsilon_C}\sum_{u,v\epsilon_V}\alpha^-_{cu}\alpha^-_{cv}(A^-_{uv} - \frac{k^-_u k^-_v}{2m})$$

where the sign + indicates positive edge weights and the sign - indicates negative edge weights, respectively.

Value

A numeric scalar, the fuzzy modularity score for signed weighted networks of the given configuration.

Author(s)

Pedro Henrique Ribeiro Santiago, <phrs16@gmail.com> [ctb]

Gustavo Hermes Soares, [rev]

Adrian Quintero, [rev]

Lisa Jamieson, [rev]

References

Gomez, S., Jensen, P., & Arenas, A. (2009). Analysis of community structure in networks of correlated data. *Physical review E*, 80(1), 016114.

See Also

FuzzyMod

Examples

```
`%du%` <- igraph::`%du%`
g <- igraph::make_full_graph(6) %du% igraph::make_full_graph(6)
g <- igraph::add_edges(g, c(1,7, 2,8))
edges <- rep(1,32)
edges[31] <- -1
igraph::E(g)$weight <- edges
plot(g, edge.label=round(igraph::E(g)$weight, 3))
```

```
wc <- list(c(1,2,3,4,5,6),c(7,8,9,10,11,12))
SignedFuzzyMod(netinput=g, membassigned=wc)
```

---

summary.cpAlgorithm *summary.cpAlgorithm*

Description

Summary method for objects of class cpAlgorithm.

Usage

```
## S3 method for class 'cpAlgorithm'
summary(
  object,
 details = c("communities.labels", "shared.nodes.labels", "isolated.nodes.labels"),
  ...
)
```

Arguments

object	An object of class cpAlgorithm; see also cpAlgorithm
details	A string or vector indicating about which part of the results more information is requested; default is c("communities.labels","shared.nodes.labels","isolated.nodes.labels' see Details
...	currently ignored

Details

The function extracts information from an object produced by cpAlgorithm. To do so, the user has to specify in details which information is requested. It is possible to extract information about the communities with either numbers (communities.numbers) or labels (communities.labels) as identifiers of the nodes. Moreover, it is possible to extract information about shared nodes with either numbers (shared.nodes.numbers) or labels (shared.nodes.labels) as identifiers of the nodes. Finally, it is possible to extract information about isolated nodes with either numbers (isolated.nodes.numbers) or labels (isolated.nodes.labels) as identifiers of the nodes. Any combination of these options can be specified in details.

Value

Prints information depending on details.

Author(s)

Jens Lange, <lange.jens@outlook.com>

Examples

```
## Example for unweighted networks

# create qgraph object
W <- matrix(c(0,1,1,1,0,0,0,0,
              0,0,1,1,0,0,0,0,
              0,0,0,0,0,0,0,0,
              0,0,0,0,1,1,1,0,
              0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,1,0,
              0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0), nrow = 8, ncol = 8, byrow = TRUE)
colnames(W) <- letters[1:8]
rownames(W) <- letters[1:8]
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

# run clique percolation for unweighted networks
results <- cpAlgorithm(W = W, k = 3, method = "unweighted")

# print results overview
results

# extract details about the communities
summary(results, details = "communities.labels")

# extract information about shared and isolated nodes
summary(results, details = c("shared.nodes.numbers", "isolated.nodes.labels"))
```

Index