Package ‘CliquePercolation’

October 28, 2019

Version 0.2.0

Date 2019-10-14

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

Depends R (>= 3.6.0)

Imports colorspace, graphics, igraph, magrittr, Matrix, methods,
Polychrome, qgraph, stats, utils

RoxygenNote 6.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Jens Lange [aut, cre],
Janis Zickfeld [ctb]

Repository CRAN

Date/Publication 2019-10-28 13:30:08 UTC

R topics documented:

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; 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 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 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 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,
              0,0,0,0,0,1,1,0,
              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)

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

## 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)
```

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 = (count_set + 1), ...)
```

Arguments

`W`	A qgraph object; 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
`...`	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 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. Changing the number of pure communities therefore changes 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. 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).

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

```
#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)


#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)
```

`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

```
# 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")
```

cpCommunitySizeDistribution

Plotting Clique Percolation Community Size Distribution

Description

Function for plotting the frequency distribution of community sizes from clique percolation community detection.

Usage

```
cpCommunitySizeDistribution(list.of.communities, color.line = "#bc0031")
```

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"`

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.

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`).

Author(s)

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

Examples

```
# 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")
```

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)
```

Arguments

`W`	A qgraph object; 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

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`

Author(s)

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

Examples

```
# 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

set.seed(4186)
results <- cpPermuteEntropy(W = W, cpThreshold.object = cpThreshold.object,
                            n = 100, interval = 0.95)

# check results
results$Confidence.Interval
results$Extracted.Rows
```

---

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) 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"))
```

Arguments

`W`	A qgraph object; 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","e` 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`.

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 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

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)
W <- Matrix::forceSymmetric(W)
W <- qgraph::qgraph(W)

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

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

# determine ratio, chi, and entropy 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"))
```

Index