# Package ‘clustAnalytics’

June 3, 2022

**Type** Package

**Title** Cluster Evaluation on Graphs

**Version** 0.4.1

**Date** 2022-05-30

**Author** Martí Renedo Mirambell

**Maintainer** Martí Renedo Mirambell <marti.renedo@gmail.com>

**Description** Evaluates the stability and significance of clusters on 'igraph' graphs.
Supports weighted and unweighted graphs. Implements the cluster evaluation methods defined by Arratia A, Renedo M (2021) <doi:10.7717/peerj-cs.600>. Also includes an implementation of the Reduced Mutual Information introduced by Newman et al. (2020).

**License** GPL (>= 3)

**Imports** Rcpp (>= 1.0.1), mcclust, mclust, truncnorm, boot, fossil,
dplyr, kdtools, Rdpack

**LinkingTo** Rcpp

**RdMacros** Rdpack

**RoxygenNote** 7.1.2

**Suggests** igraphdata, knitr, rmarkdown, testthat,

**VignetteBuilder** knitr

**Encoding** UTF-8

**Depends** R (>= 2.10), igraph

**LazyData** true

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2022-06-03 07:30:02 UTC

## R topics documented:

---

`average_degree` *Average Degree*

---

## Description

Average degree (weighted degree, if the graph is weighted) of a graph's communities.

## Usage

```
average_degree(g, com)
```

### Arguments

| | |
|---|---|
| `g` | Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights. |
| `com` | community membership integer vector. Each element corresponds to a vertex. |

### Value

Numeric vector with the average degree of each community.

### See Also

Other cluster scoring functions: `FOMD()`, `average_odf()`, `conductance()`, `coverage()`, `cut_ratio()`, `density_ratio()`, `edges_inside()`, `expansion()`, `internal_density()`, `max_odf()`, `normalized_cut()`, `weighted_clustering_coefficient()`, `weighted_transitivity()`

### Examples

```
data(karate, package="igraphdata")
average_degree(karate, membership(cluster_louvain(karate)))
```

---

## `average_odf` *Average Out Degree Fraction*

---

### Description

Computes the Average Out Degree Fraction (Average ODF) of a graph (which can be weighted) and its communities.

### Usage

```
average_odf(g, com)
```

### Arguments

| | |
|---|---|
| `g` | Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights (otherwise, all edges are assumed to be 1). |
| `com` | Community membership integer vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to. |

### Value

Numeric vector with the Average ODF of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
average_odf(karate, membership(cluster_louvain(karate)))
```

---

## barabasi_albert_blocks — *Generates a Barabási-Albert graph with community structure*

---

### Description

Generates a Barabási-Albert graph with community structure

### Usage

```
barabasi_albert_blocks(
  m,
  p,
  B,
  t_max,
  G0 = NULL,
  t0 = NULL,
  G0_labels = NULL,
  sample_with_replacement = FALSE,
  type = "Hajek"
)
```

### Arguments

| | |
|---|---|
| m | number of edges added at each step. |
| p | vector of label probabilities. If they don't sum 1, they will be scaled accordingly. |
| B | matrix indicating the affinity of vertices of each label. |
| t_max | maximum value of t (which corresponds to graph order) |
| G0 | initial graph |
| t0 | t value at which new vertex start to be attached. If G0 is provided, this argument is ignored and assumed to be gorder(G0)+1. If it isn't, a G0 graph will be generated with order t0-1. |
| G0_labels | labels of the initial graph. If NULL, they will all be set to 1. |
| sample_with_replacement | If TRUE, allows parallel edges. |
| type | Either "Hajek" or "block_first". |

### Value

The resulting graph, as an igraph object. The vertices have a "label" attribute.

### Examples

```
B <- matrix(c(1, 0.2, 0.2, 1), ncol=2)
G <- barabasi_albert_blocks(m=4, p=c(0.5, 0.5), B=B, t_max=100, type="Hajek",
                            sample_with_replacement = FALSE)
```

---

`boot_alg_list` *Performs nonparametric bootstrap to a graph and a list of clustering algorithms*

---

### Description

Performs nonparametric bootstrap on a graph's by resampling its vertices and clustering the results using a list of clustering algorithms.

### Usage

```
boot_alg_list(
  alg_list = list(Louvain = cluster_louvain, `label prop` = cluster_label_prop,
    walktrap = cluster_walktrap),
  g,
  R = 999,
  return_data = FALSE,
  type = "global"
)
```

### Arguments

| | |
|---|---|
| `alg_list` | List of igraph clustering algorithms |
| `g` | `igraph` graph object |
| `R` | Number of bootstrap replicates. |
| `return_data` | Logical. If `TRUE`, returns a list of "`boot`" objects with the full results. Otherwise, returns a table with the mean results. |
| `type` | Can be "global" (Variation of Information, Reduced Mutual Information, and adjusted Rand Index) or "cluster-wise" (Jaccard distance) |

### Value

If `return_data` is set to `TRUE`, returns a list of objects of class "`boot`" (see `boot`). Otherwise, returns as table with the mean distances from the clusters in the original graph to the resampled ones, for each of the algorithms.

`clustAnalytics` *clustAnalytics*

## Description

This package evaluates the stability and significance of clusters in `igraph` graphs. Supports weighted and unweighted graphs.

## Details

Extensions to weighted graphs of multiple functions present in `igraph` are provided, such as scoring functions or edge rewiring methods.

## Author(s)

Martí Renedo Mirambell

`conductance` *Conductance*

## Description

Conductance of a graph's communities, which is given by

$$\frac{c_s}{2m_s + c_s}$$

, where $c_s$ is the weight of the edges connecting the community s to the rest of the graph, and m_s is the internal weight of the community.

## Usage

```
conductance(g, com)
```

## Arguments

| | |
|---|---|
| `g` | Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights. |
| `com` | community membership integer vector. Each element corresponds to a vertex. |

## Value

Numeric vector with the conductance of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
conductance(karate, membership(cluster_louvain(karate)))
```

---

## contingency_to_membership_vectors

*Computes possible membership vectors from contingency table*

---

### Description

Given a contingency table, obtains a possible pair of corresponding labelings. That is, element M[i,j] is the number of elements that belong to community i in the first labeling and j in the second.

### Usage

```
contingency_to_membership_vectors(M)
```

### Arguments

| | |
|---|---|
| M | the contingency table |

### Value

a list containing the two membership vectors

---

## count_contingency_tables_log

*Natural logarithm of the number of contingency tables*

---

### Description

Given a contingency table, returns the natural logarithm of the number of contingency tables that share the same column and row sums. This implementation combines a Markov Chain Monte Carlo approximation with an analytical formula. The input can be either M a contingency table, or two vectors of labels c1 and c2 (in this case, we are counting contingency tables with the same column an row sums as the one produced by c1 and c2)

### Usage

```
count_contingency_tables_log(c1, c2, M = NULL, monte_carlo_only = FALSE)
```

### Arguments

| | |
|---|---|
| c1, c2 | membership vectors |
| M | contingency table |
| monte_carlo_only | Uses only the Monte Carlo approximation |

---

## coverage *Coverage*

---

### Description

Computes the coverage (fraction of internal edges with respect to the total number of edges) of a graph and its communities

### Usage

```
coverage(g, com)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an `igraph` object). |
| com | Community membership integer vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to. |

### Value

Numeric value of the coverage of `g` and `com`.

### See Also

Other cluster scoring functions: `FOMD()`, `average_degree()`, `average_odf()`, `conductance()`, `cut_ratio()`, `density_ratio()`, `edges_inside()`, `expansion()`, `internal_density()`, `max_odf()`, `normalized_cut()`, `weighted_clustering_coefficient()`, `weighted_transitivity()`

### Examples

```
data(karate, package="igraphdata")
coverage(karate, membership(cluster_louvain(karate)))
```

## cut_ratio — *Cut Ratio*

### Description

The cut ratio of a graph's community is the total edge weight connecting the community to the rest of the graph divided by number of unordered pairs of vertices such that one belongs to the community and the other does not.

### Usage

```
cut_ratio(g, com)
```

### Arguments

| | |
|---|---|
| `g` | Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights. |
| `com` | community membership integer vector. Each element corresponds to a vertex. |

### Value

Numeric vector with the cut ratio of each community.

### See Also

Other cluster scoring functions: `FOMD()`, `average_degree()`, `average_odf()`, `conductance()`, `coverage()`, `density_ratio()`, `edges_inside()`, `expansion()`, `internal_density()`, `max_odf()`, `normalized_cut()`, `weighted_clustering_coefficient()`, `weighted_transitivity()`

### Examples

```
data(karate, package="igraphdata")
cut_ratio(karate, membership(cluster_louvain(karate)))
```

## c_rs_table — *Contingency table from membership vectors*

### Description

Given two membership vectors, returns the corresponding contingency table. we assume the labels are >=1 and numbered consecutively. If not consecutive (some labels are unused) this implementation still works, but will be less efficient.

### Usage

```
c_rs_table(c1, c2)
```

### Arguments

| | |
|---|---|
| c1, c2 | membership vectors (integer values containing the index of each community) |

---

`density_ratio` *Density Ratio*

---

### Description

Density ratio of a graph's communities.

### Usage

```
density_ratio(g, com, type = "local")
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights. |
| com | community membership integer vector. Each element corresponds to a vertex. |
| type | can either be "local" or "global" |

### Value

Numeric vector with the internal density of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
density_ratio(karate, membership(cluster_louvain(karate)))
```

## edges_inside *Edges Inside*

### Description

Number of edges inside a graph's communities, or their accumulated weight if the graph's edges are weighted.

### Usage

```
edges_inside(g, com)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights. |
| com | community membership integer vector. Each element corresponds to a vertex. |

### Value

Numeric vector with the internal edge weight of each community

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
edges_inside(karate, membership(cluster_louvain(karate)))
```

## estimate_H_fraction *Estimates |H_i|/|H_i+1|*

### Description

Estimates the fraction of elements of H_i that are also in H_i+1 (where i=(k,l))

### Usage

```
estimate_H_fraction(M, k, l, error = 0.1)
```

### Arguments

| | |
|---|---|
| `M` | matrix |
| `k, l` | Coordinates of the first element that is not invariant |
| `error` | error for the convergence of the method |

### Value

value of H_i/H_i+1

---

## `estimate_H_fractions` *Estimates |H_i|/|H_i+1| for the first r rows*

---

### Description

The product of all these ratios is is the total number of contingency tables (of the same margins as M) divided by the number that match M until the r-th row (included, 0-indexed).

### Usage

```
estimate_H_fractions(M, r, error = 0.1)
```

### Arguments

| | |
|---|---|
| `M` | contingency table |
| `r` | row index |
| `error` | error for the convergence of the method |

### Value

NumericVector containing all the ratios

---

## `estimate_H_fraction_r_rows` *Estimates |H_0|/|H_r*|*

---

### Description

This is the total number of contingency tables (of the same margins as M) divided by the number that match M until the r-th row (included, 0-indexed). Note that if r==0, this is always 1 by definition.

### Usage

```
estimate_H_fraction_r_rows(M, r, error = 0.1)
```

### Arguments

| | |
|---|---|
| `M` | contingency table |
| `r` | row index |
| `error` | error for the convergence of the method |

---

## `evaluate_significance` *Evaluates significance of cluster algorithm results on a graph*

---

### Description

Given a graph and a list of clustering algorithms, computes several scoring functions on the clusters found by each of the algorithms.

### Usage

```
evaluate_significance(
  g,
  alg_list = list(Louvain = cluster_louvain, `label prop` = cluster_label_prop,
    walktrap = cluster_walktrap),
  no_clustering_coef = TRUE,
  ground_truth = FALSE,
  gt_clustering = NULL,
  w_max = NULL
)
```

### Arguments

| | |
|---|---|
| `g` | Graph to be analyzed (as an `igraph` object) |
| `alg_list` | List of clustering algorithms, which take an `igraph` graph as input and return an object of the `communities` class. |
| `no_clustering_coef` | Logical. If `TRUE`, skips the computation of the clustering coefficient, which is the most computationally costly of the scoring functions. |
| `ground_truth` | Logical. If set to `TRUE`, computes the scoring functions for a ground truth clustering, which has to be provided as `gt_clustering` |
| `gt_clustering` | Vector of integers that correspond to labels of the ground truth clustering. Only used if `ground_truth` is set to `TRUE`. |
| `w_max` | Numeric. Upper bound for edge weights. Should be generally left as default (`NULL`). |

### Value

A data frame with the values of scoring functions (see `scoring_functions`) of the clusters obtained by applying the clustering algorithms to the graph.

## Examples

```
data(karate, package="igraphdata")
evaluate_significance(karate)
```

---

`evaluate_significance_r`

*Evaluates the significance of a graph's clusters*

---

## Description

Computes community scoring functions to the communities obtained by applying the given clustering algorithms to a graph. These are compared to the same scores for randomized versions of the graph obtained by a switching algorithm that rewires edges.

## Usage

```
evaluate_significance_r(
  g,
  alg_list = list(Louvain = cluster_louvain, `label prop` = cluster_label_prop,
    walktrap = cluster_walktrap),
  no_clustering_coef = FALSE,
  ground_truth = FALSE,
  gt_clustering = NULL,
  table_style = "default",
  ignore_degenerate_cl = TRUE,
  Q = 100,
  lower_bound = 0,
  weight_sel = "const_var",
  n_reps = 5,
  w_max = NULL
)
```

## Arguments

`g` Graph to be analyzed (as an `igraph` object)

`alg_list` List of clustering algorithms, which take an `igraph` graph as input and return an object of the `communities` class.

`no_clustering_coef` Logical. If `TRUE`, skips the computation of the clustering coefficient, which is the most computationally costly of the scoring functions.

`ground_truth` Logical. If set to `TRUE`, computes the scoring functions for a ground truth clustering, which has to be provided as `gt_clustering`

`gt_clustering` Vector of integers that correspond to labels of the ground truth clustering. Only used if `ground_truth` is set to `TRUE`.

| | |
|---|---|
| table_style | By default returns a table with three columns per algorithm: the original one, the mean of the corresponding rewired scores (suffix "_r") and it's percentile rank within the distribution of rewired scores (suffix "_percentile"). If table_style == "string", instead returns a table with a column per algorithm where each element is of the form "original\|rewired(percentile)" |
| ignore_degenerate_cl | Logical. If TRUE, when computing the means of the scoring functions, samples with only one cluster will be ignored. See rewireCpp. |
| Q | Numeric. Parameter that controls the number of iterations of the switching algorithm, which will be Q times the order of the graph. |
| lower_bound | Numeric. Lower bound to the edge weights. The randomization process will avoid steps that would make edge weights fall outside this bound. It should generally be left as 0 to avoid negative weights. |
| weight_sel | Can be either `const_var` or `max_weight`. |
| n_reps | Number of samples of the rewired graph. |
| w_max | Numeric. Upper bound for edge weights. The randomization algorithm will avoid steps that would make edge weights fall outside this bound. Should be generally left as default (`NULL`), unless the network has by nature or by construction a known upper bound. |

## Value

A matrix with the results of each scoring function and algorithm. See `table_style` for details.

---

`expansion` *Expansion*

---

## Description

Given a graph (possibly weighted) split into communities, the expansion of a community is the sum of all edge weights connecting it to the rest of the graph divided by the number of vertices in the community

## Usage

```
expansion(g, com)
```

## Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights. |
| com | community membership integer vector. Each element corresponds to a vertex. |

## Value

Numeric vector with the expansion of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
expansion(karate, membership(cluster_louvain(karate)))
```

---

## FOMD — *FOMD (Fraction Over Median Degree)*

---

### Description

Given a weighted graph and a partition into communities, returns the fraction of nodes of each community whose internal degree (i.e. the degree accounting only intra-community edges) is greater than the median degree of the whole graph.

### Usage

```
FOMD(g, com, edgelist = NULL)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights. |
| com | Community membership integer vector. Each element corresponds to a vertex. |
| edgelist | alternatively, the edgelist of the graph, as a matrix where the first two columns to the vertices and the third is the weight of each edge. |

### Value

Numeric vector with the FOMD of each community.

### See Also

Other cluster scoring functions: average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
FOMD(karate, membership(cluster_louvain(karate)))
```

## g_forex — *Forex correlation network*

### Description

Network built from correlations between time series of exchange rate returns. It was built from the 13 most traded currencies and with data of January 2009. It is a complete graph of 78 vertices (corresponding to pairs of currencies) and has edge weights bounded between 0 and 1.

### Usage

```
g_forex
```

### Format

An igraph object with 78 vertices and 3003 weighted edges

## H_fractions_rows — *Estimates |H_i|/|H_(i+1)| for the first n_rows rows*

### Description

Estimates |H_i|/|H_(i+1)| for the first n_rows rows

### Usage

```
H_fractions_rows(M, n_rows, error = 0.01)
```

### Arguments

| | |
|---|---|
| `M` | contingency table |
| `n_rows` | number of rows |
| `error` | for the convergence of the method |

### Value

vector with all the |H_i|/|H_(i+1)| fractions

## igraph_to_edgelist *Returns edgelist with weights from a weighted igraph graph*

### Description

This function is just used internally for testing the package

### Usage

```
igraph_to_edgelist(g, sort = TRUE)
```

### Arguments

| | |
|---|---|
| g | igraph graph with weighted edges |
| sort | sorts the edge list lexicographically before returning |

### Value

A matrix where the first two columns indicate the incident vertices, and the third is the weight of the corresponding edge.

## internal_density *Internal Density*

### Description

Internal density of a graph's communities. That is, the sum of weights of their edges divided by the number of unordered pairs of vertices (which is the number of potential edges).

### Usage

```
internal_density(g, com)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights. |
| com | community membership integer vector. Each element corresponds to a vertex. |

### Value

Numeric vector with the internal density of each community.

### See Also

Other cluster scoring functions: `FOMD()`, `average_degree()`, `average_odf()`, `conductance()`, `coverage()`, `cut_ratio()`, `density_ratio()`, `edges_inside()`, `expansion()`, `max_odf()`, `normalized_cut()`, `weighted_clustering_coefficient()`, `weighted_transitivity()`

### Examples

```
data(karate, package="igraphdata")
internal_density(karate, membership(cluster_louvain(karate)))
```

---

## `log_omega_estimation` *Approximation of log(omega(a,b))*

---

### Description

Where omega(a,b) is the number of contingency tables with a, b as row and column sums. This approximation is only good for dense tables.

### Usage

```
log_omega_estimation(c1, c2, base = exp(1))
```

### Arguments

| | |
|---|---|
| `c1, c2` | membership vectors |
| `base` | base of the logarithm (e by default) |

---

## `max_odf` *Max Out Degree Fraction*

---

### Description

Computes the Maximum Out Degree Fraction (Max ODF) of a graph (which can be weighted) and its communities.

Computes the Flake Out Degree Fraction (Max ODF) of a graph (which can be weighted) and its communities.

### Usage

```
max_odf(g, com)

max_odf(g, com)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights (otherwise, all edges are assumed to be 1). |
| com | Community membership integer vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to. |

### Value

Numeric vector with the Max ODF of each community.

Numeric vector with the Max ODF of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), normalized_cut(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
max_odf(karate, membership(cluster_louvain(karate)))
data(karate, package="igraphdata")
max_odf(karate, membership(cluster_louvain(karate)))
```

---

## normalized_cut — *Normalized cut*

---

### Description

Normalized cut of a graph's communities, which is given by

$$\frac{c_s}{2m_s + c_s} + \frac{c_s}{2(m - m_s) + c_s}$$

, where $c_s$ is the weight of the edges connecting the community s to the rest of the graph, $m_s$ is the internal weight of the community, and $m$ is the total weight of the network.

### Usage

```
normalized_cut(g, com)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object). If the edges have a "weight" attribute, those will be used as weights. |
| com | community membership integer vector. Each element corresponds to a vertex. |

### Value

Numeric vector with the normalized cut of each community.

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), weighted_clustering_coefficient(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
normalized_cut(karate, membership(cluster_louvain(karate)))
```

---

## out_degree_fractions — *Maximum, Average, and Flake Out Degree Fractions of a Graph Partition*

---

### Description

Given a weighted graph and a partition into communities, returns the maximum, average and flake out degree fractions of each community.

### Usage

```
out_degree_fractions(g, com, edgelist)
```

### Arguments

| | |
|---|---|
| g | Graph to be analyzed (as an igraph object) |
| com | Community membership vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to. |
| edgelist | alternatively, the edgelist of the graph |

### Value

A numeric matrix where each row corresponds to a community, and the columns contain the max, average and flake ODFs respectively.

`reduced_mutual_information` *Reduced Mutual Information*

## Description

Computes the Newman's Reduced Mutual Information (RMI) as defined in (Newman et al. 2020).

## Usage

```
reduced_mutual_information(
  c1,
  c2,
  base = exp(1),
  normalized = FALSE,
  method = "approximation2",
  warning = TRUE
)
```

## Arguments

| | |
|---|---|
| `c1, c2` | membership vectors |
| `base` | base of the logarithms used in the calculations. Changing it only scales the final value. By default set to e=exp(1). |
| `normalized` | If true, computes the normalized version of the corrected mutual information. |
| `method` | Can be "hybrid" (default, combines Monte Carlo with analytical formula), "monte_carlo", approximation1" (appropriate for partitions into many very small clusters), or "approximation2" (for partitions into few larger clusters). |
| `warning` | set to false to ignore the warning. |

## Details

The implementation is based on equations 23 (25 for the normalized case) and 29 in (Newman et al. 2020). The evaluations of the $\Gamma$ functions can get too large and cause overflow issues in the intermediate steps, so the following term of equation 29:

$$\frac{1}{2}\log\frac{\Gamma(\mu R)\Gamma(\nu S)}{(\Gamma(\nu)\Gamma(R))^S(\Gamma(\mu)\Gamma(S))^R}$$

is rewritten as

$$\frac{1}{2}(\log\Gamma(\mu R) + \log\Gamma(\nu S) - S\log(\Gamma(\nu) - S\log(\Gamma(R) - R\log\Gamma(\mu) - R\log\Gamma(R))$$

, and then the function lgamma is used instead of gamma.

## Value

The value of Newman's RMI (a scalar).

## References

Newman MEJ, Cantwell GT, Young J (2020). "Improved mutual information measure for clustering, classification, and community detection." *Phys. Rev. E*, **101**(4), 042304. doi: 10.1103/PhysRevE.101.042304.

---

`relabel` *Relabels membership vector*

---

## Description

Takes a vector of vertex ids indicating community membership, and relabels the communities to have consecutive values from 1 to the number of communities.

## Usage

```
relabel(c)
```

## Arguments

| | |
|---|---|
| `c` | numeric vector of vertex ids, not necessarily consecutive |

## Value

A numeric vector of consecutive vertex ids starting from one

---

`rewireCpp` *Randomizes a weighted graph while keeping the degree distribution constant.*

---

## Description

Converts the graph to a weighted edge list in NumericMatrix, which is compatible with Rcpp. The Rcpp function "randomize" is called, and then the resulting edge list is converted back into an igraph object.

## Usage

```
rewireCpp(
  g,
  Q = 100,
  weight_sel = "max_weight",
  lower_bound = 0,
  upper_bound = NULL
)
```

## Arguments

- `g` — `igraph` graph, which can be weighted.
- `Q` — Numeric. Parameter that controls the number of iterations, which will be Q times the order of the graph.
- `weight_sel` — can be either "const_var" or "max_weight".
- `lower_bound, upper_bound` — Bounds to the edge weights. The randomization process will avoid steps that would make edge weights fall outside these bounds. Set to NULL for no bound. By default, 0 and NULL respectively.

## Value

The rewired graph.

---

`scoring_functions` *Scoring Functions of a Graph Partition*

---

## Description

Computes the scoring functions of a graph and its clusters.

## Usage

```
scoring_functions(
  g,
  com,
  no_clustering_coef = TRUE,
  type = "local",
  weighted = TRUE,
  w_max = NULL
)
```

## Arguments

- `g` — Graph to be analyzed (as an `igraph` object). If the edges have a "weight" attribute, those will be used as weights (otherwise, all edges are assumed to be 1).
- `com` — Community membership integer vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to.
- `no_clustering_coef` — Logical. If TRUE, skips the computation of the clustering coefficient (which can be slow on large graphs).
- `type` — can be "local" for a cluster by cluster analysis, or "global" for a global analysis of the whole graph partition.
- `weighted` — Is the graph weighted? If it is, doesn't compute TPR score.
- `w_max` — Numeric. Upper bound for edge weights. Should be generally left as default (NULL). Only affects the computation of the clustering coefficient.

### Value

If `type=="local"`, returns a dataframe with a row for each community, and a column for each score. If `type=="global"`, returns a single row with the weighted average scores.

### Examples

```
data(karate, package="igraphdata")
scoring_functions(karate, membership(cluster_louvain(karate)))
```

---

## sort_matrix *Sort matrix*

---

### Description

Given a matrix, rearranges rows and columns so that row sums and col sums end up in ascending order.

### Usage

```
sort_matrix(M)
```

### Arguments

| | |
|---|---|
| M | matrix |

### Value

rearranged matrix

---

## triangle_participation_ratio_communities *Triangle Participation Ratio (community-wise)*

---

### Description

Computes the triangle participation ratio (proportion of vertices that belong to a triangle). The computation is done to the subgraphs induced by each of the communities in the given partition.

### Usage

```
triangle_participation_ratio_communities(g, com)
```

### Arguments

| | |
|---|---|
| g | The input graph (as an igraph object). Edge weights and directions are ignored. |
| com | Community membership vector. Each element corresponds to a vertex of the graph, and contains the index of the community it belongs to. |

### Value

A vector containing the triangle participation ratio of each community.

---

## walk_step *Performs a step of the Markov Chain Monte Carlo method*

---

### Description

Modifies the matrix while keeping the column and row sums constant, as well as leaving the positions strictly preceding (k,l) in lexicographical order invariant.

### Usage

```
walk_step(M, k, l)
```

### Arguments

| | |
|---|---|
| `M` | matrix |
| `k, l` | Coordinates of the first element that is not invariant |

### Value

boolean indicating whether the step left the matrix invariant

---

## weighted_clustering_coefficient *Weighted clustering coefficient of a weighted graph.*

---

### Description

Weighted clustering Computed using the definition given by McAssey, M. P. and Bijma, F. in "A clustering coefficient for complete weighted networks" (2015).

### Usage

```
weighted_clustering_coefficient(g, upper_bound = NULL)
```

### Arguments

| | |
|---|---|
| `g` | igraph graph |
| `upper_bound` | upper bound to the edge weights used to compute the integral |

### Value

The weighted clustering coefficient of the graph (a scalar).

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_transitivity()

### Examples

```
data(karate, package="igraphdata")
weighted_clustering_coefficient(karate)
```

---

## weighted_transitivity *Weighed transitivity of a weighted graph.*

---

### Description

Computed using the definition given by McAssey, M. P. and Bijma, F. in "A clustering coefficient for complete weighted networks" (2015).

### Usage

```
weighted_transitivity(g, upper_bound = NULL)
```

### Arguments

| | |
|---|---|
| g | igraph graph |
| upper_bound | upper bound to the edge weights used to compute the integral |

### Value

The weighted transitivity of the graph (a scalar).

### See Also

Other cluster scoring functions: FOMD(), average_degree(), average_odf(), conductance(), coverage(), cut_ratio(), density_ratio(), edges_inside(), expansion(), internal_density(), max_odf(), normalized_cut(), weighted_clustering_coefficient()

### Examples

```
data(karate, package="igraphdata")
weighted_transitivity(karate)
```

# Index