Package 'netrics'

August 31, 2026

Title Many Marks, Measures, Memberships, and Motifs for Networks

Version 1.0.1

Description Many tools for calculating network, node, or tie
marks, measures, motifs and memberships of many different types of networks.
Marks identify structural positions, measures quantify network properties,
memberships classify nodes into groups, and motifs tabulate substructure participation.
All functions operate with all classes of network data covered in 'manynet',
and on directed, undirected, multiplex, multimodal, signed, and other networks.

URL https://stocnet.github.io/netrics/

BugReports https://github.com/stocnet/netrics/issues

License MIT + file LICENSE

Language en-GB

Encoding UTF-8

Depends R (>= 4.1.0), manynet (>= 2.2.3)

Imports dplyr, igraph (>= 2.1.0)

Suggests autograph, sna, testthat (>= 3.0.0)

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown, learnr

Config/testthat/parallel true

Config/testthat/edition 3

Config/testthat/start-first tutorials_netrics, measure_net,
member_nodes, measure_nodes

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID:
<https://orcid.org/0000-0002-8361-9647>)

Maintainer James Hollway <james.hollway@graduateinstitute.ch>

Repository CRAN

Date/Publication 2026-08-31 19:30:41 UTC

Contents

mark_core . . . 3
mark_degree . . . 5
mark_diff . . . 6
mark_dyads . . . 7
mark_nodes . . . 8
mark_select_node . . . 10
mark_select_tie . . . 11
mark_ties . . . 12
mark_triangles . . . 13
measure_assort_net . . . 15
measure_assort_node . . . 18
measure_breadth . . . 20
measure_brokerage . . . 21
measure_broker_node . . . 22
measure_broker_tie . . . 25
measure_centralisation_between . . . 26
measure_centralisation_close . . . 28
measure_centralisation_degree . . . 30
measure_centralisation_eigen . . . 31
measure_central_between . . . 33
measure_central_close . . . 36
measure_central_degree . . . 43
measure_central_eigen . . . 46
measure_central_tie_between . . . 52
measure_central_tie_close . . . 53
measure_central_tie_degree . . . 54
measure_central_tie_eigen . . . 56
measure_closure . . . 57
measure_closure_node . . . 59
measure_cohesion . . . 61
measure_core . . . 63
measure_diffusion_infection . . . 65
measure_diffusion_net . . . 67
measure_diffusion_node . . . 70
measure_diverse_net . . . 72
measure_diverse_node . . . 75
measure_features . . . 76
measure_fit . . . 80
measure_fragmentation . . . 84
measure_hierarchy . . . 86
measure_periods . . . 87
member_brokerage . . . 88
member_cliques . . . 89
member_community . . . 91
member_community_hier . . . 93
member_community_non . . . 96

member_components . . . 101
member_core . . . 102
member_diffusion . . . 104
member_equivalence . . . 105
method_cluster . . . 110
method_coreness . . . 111
method_kselect . . . 115
method_regularity . . . 117
method_split . . . 119
motif_brokerage_net . . . 120
motif_brokerage_node . . . 121
motif_clique . . . 122
motif_composition . . . 123
motif_exposure . . . 127
motif_hazard . . . 128
motif_hierarchy . . . 129
motif_homophily . . . 130
motif_net . . . 132
motif_node . . . 135
motif_path . . . 137
motif_periods . . . 139

Index **141**

`mark_core` *Marking nodes as core or periphery*

Description

`node_is_core()` identifies whether nodes belong to the core of the network, as opposed to the periphery.

Usage

```
node_is_core(
  .data,
  coreness = NULL,
  direction = c("all", "out", "in"),
  centrality = NULL
)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

coreness	Which method to use to calculate nodes' coreness. One of "correlation", "rich", "transition", or "hub"; see method_coreness for what each does. By default NULL, which uses "rich" for a weighted, directed, or two-mode network, since it is the only method that reads those properties directly, and "correlation" otherwise.
direction	One of "all" (the default), "out", or "in". For a directed network, "out" scores nodes on the ties they send and "in" on the ties they receive. Ignored for undirected and two-mode networks.
centrality	Deprecated; use coreness instead.

Value

A node_mark logical vector the length of the nodes in the network, giving either TRUE or FALSE for each node depending on whether the condition is matched.

Core-periphery

This function is used to identify which nodes should belong to the core, and which to the periphery. It seeks to minimize the following quantity:

$$Z(S_1) = \sum_{(i<j)\in S_1} \mathbf{I}_{\{A_{ij}=0\}} + \sum_{(i<j)\notin S_1} \mathbf{I}_{\{A_{ij}=1\}}$$

where nodes $\{i, j, ..., n\}$ are ordered in descending coreness, A is the adjacency matrix, and the indicator function is 1 if the predicate is true or 0 otherwise. Note that minimising this quantity maximises density in the core block and minimises density in the periphery block; it ignores ties between these blocks.

Which ordering the nodes are swept in depends on the method named by coreness, for which see method_coreness.

References

On core-periphery partitioning:

Borgatti, Stephen P., and Martin G. Everett. 2000. "Models of core/periphery structures". *Social Networks*, 21(4), 375-395. doi:10.1016/S03788733(99)000192

Lip, Sean Z. W. 2011. "A fast algorithm for the discrete core/periphery bipartitioning problem". doi:10.48550/arXiv.1102.5511

See Also

Other core-periphery: measure_core, member_core

Other marks: mark_degree, mark_diff, mark_dyads, mark_nodes, mark_select_node, mark_select_tie, mark_ties, mark_triangles

Other nodal: mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion,

member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_is_core(ison_adolescents)
ison_adolescents |>
   mutate(corep = node_is_core())
```

`mark_degree` *Marking nodes based on degree properties*

Description

These functions return logical vectors the length of the nodes in a network identifying which hold certain properties or positions in the network.

- `node_is_isolate()` marks nodes that are isolates, with neither incoming nor outgoing ties.
- `node_is_pendant()` marks nodes that are pendants, with exactly one incoming or outgoing tie.
- `node_is_universal()` identifies whether nodes are adjacent to all other nodes in the network.

Usage

```
node_is_isolate(.data)

node_is_pendant(.data)

node_is_universal(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `node_mark` logical vector the length of the nodes in the network, giving either TRUE or FALSE for each node depending on whether the condition is matched.

Universal/dominating node

A universal node is adjacent to all other nodes in the network. It is also sometimes called the dominating vertex because it represents a one-element dominating set. A network with a universal node is called a cone, and its universal node is called the apex of the cone. A classic example of a cone is a star graph, but friendship, wheel, and threshold graphs are also cones.

See Also

Other degree: measure_central_degree, measure_central_tie_degree, measure_centralisation_degree

Other marks: mark_core, mark_diff, mark_dyads, mark_nodes, mark_select_node, mark_select_tie, mark_ties, mark_triangles

Other nodal: mark_core, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_is_isolate(ison_brandes)
node_is_universal(create_star(11))
```

---

mark_diff *Marking nodes based on diffusion properties*

Description

These functions return logical vectors the length of the nodes in a network identifying which hold certain properties or positions in the network.

- `node_is_infected()` marks nodes that are infected by a particular time point.
- `node_is_exposed()` marks nodes that are exposed to a given (other) mark.
- `node_is_latent()` marks nodes that are latent at a particular time point.
- `node_is_recovered()` marks nodes that are recovered at a particular time point.

Usage

```
node_is_latent(.data, time = 0)

node_is_infected(.data, time = 0)

node_is_recovered(.data, time = 0)

node_is_exposed(.data, mark, time = 0)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`time`	A time step at which nodes are identified.
`mark`	vector denoting which nodes are infected

Value

A node_mark logical vector the length of the nodes in the network, giving either TRUE or FALSE for each node depending on whether the condition is matched.

Exposed

node_is_exposed() is similar to node_exposure(), but returns a mark (TRUE/FALSE) vector indicating which nodes are currently exposed to the diffusion content. This diffusion content can be expressed in the 'mark' argument. If no 'mark' argument is provided, and '.data' is a diff_model object, then the function will return nodes exposure to the seed nodes in that diffusion.

See Also

Other marks: mark_core, mark_degree, mark_dyads, mark_nodes, mark_select_node, mark_select_tie, mark_ties, mark_triangles

Other nodal: mark_core, mark_degree, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Other diffusion: measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, member_diffusion, motif_exposure, motif_hazard

Examples

```
# To mark nodes that are latent by a particular time point
node_is_latent(play_diffusion(create_tree(6), latency = 1), time = 1)
# To mark nodes that are infected by a particular time point
node_is_infected(play_diffusion(create_tree(6)), time = 1)
# To mark nodes that are recovered by a particular time point
node_is_recovered(play_diffusion(create_tree(6), recovery = 0.5), time = 3)
# To mark which nodes are currently exposed
(expos <- node_is_exposed(manynet::create_tree(14), mark = c(1,3)))
which(expos)
```

---

mark_dyads *Marking ties based on dyadic properties*

Description

These functions return logical vectors the length of the ties in a network identifying which are embedded within particular dyads.

- tie_is_multiple() marks ties that are multiples.
- tie_is_reciprocated() marks ties that are mutual/reciprocated.

They are most useful in highlighting parts of the network where relationships are denser.

Usage

```
tie_is_multiple(.data)

tie_is_reciprocated(.data)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().

Value

A tie_mark logical vector the length of the ties in the network, giving either TRUE or FALSE for each tie depending on whether the condition is matched.

See Also

Other marks: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, mark_select_tie, mark_ties, mark_triangles

Other tie: mark_select_tie, mark_ties, mark_triangles, measure_broker_tie, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen

Examples

```
tie_is_multiple(fict_marvel)
tie_is_reciprocated(ison_algebra)
```

---

mark_nodes *Marking nodes based on structural properties*

Description

These functions return logical vectors the length of the nodes in a network identifying which hold certain properties or positions in the network.

- node_is_independent() marks nodes that are members of the largest independent set, aka largest internally stable set.
- node_is_cutpoint() marks nodes that cut or act as articulation points in a network, increasing the number of connected components when removed.
- node_is_fold() marks nodes that are in a structural fold between two or more triangles that are only connected by that node.
- node_is_mentor() marks a proportion of high indegree nodes as 'mentors' (see details).
- node_is_neighbor() marks nodes that are neighbours of a given node.

Usage

```
node_is_independent(.data)

node_is_cutpoint(.data)

node_is_fold(.data)

node_is_mentor(.data, elites = 0.1)

node_is_neighbor(.data, node)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
elites	The proportion of nodes to be selected as mentors. By default this is set at 0.1. This means that the top 10% of nodes in terms of degree, or those equal to the highest rank degree in the network, whichever is the higher, will be used to select the mentors. Note that if nodes are equidistant from two mentors, they will choose one at random. If a node is without a path to a mentor, for example because they are an isolate, a tie to themselves (a loop) will be created instead. Note that this is a different default behaviour than that described in Valente and Davis (1999).
node	A node ID or name for which neighbors are to be marked.

Value

A node_mark logical vector the length of the nodes in the network, giving either TRUE or FALSE for each node depending on whether the condition is matched.

References

On independent sets:
Tsukiyama, Shuji, Mikio Ide, Hiromu Ariyoshi, and Isao Shirawaka. 1977. "A new algorithm for generating all the maximal independent sets". *SIAM Journal on Computing*, 6(3):505–517. doi:10.1137/0206036

On articulation or cut-points:
Tarjan, Robert E. and Uzi Vishkin. 1985. "An Efficient Parallel Biconnectivity Algorithm", *SIAM Journal on Computing* 14(4): 862-874. doi:10.1137/0214061

On structural folds:
Vedres, Balazs, and David Stark. 2010. "Structural folds: Generative disruption in overlapping groups", *American Journal of Sociology* 115(4): 1150-1190. doi:10.1086/649497

On mentoring:
Valente, Thomas, and Rebecca Davis. 1999. "Accelerating the Diffusion of Innovations Using Opinion Leaders", *Annals of the American Academy of Political and Social Science* 566: 56-67.

See Also

Other marks: mark_core, mark_degree, mark_diff, mark_dyads, mark_select_node, mark_select_tie, mark_ties, mark_triangles

Other nodal: mark_core, mark_degree, mark_diff, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_is_independent(ison_adolescents)
node_is_cutpoint(ison_brandes)
node_is_fold(create_explicit(A-B, B-C, A-C, C-D, C-E, D-E))
```

---

mark_select_node *Marking nodes based on measures*

Description

These functions return logical vectors the length of the nodes in a network identifying which hold certain properties or positions in the network.

- `node_is_random()` marks one or more nodes at random.
- `node_is_max()` and `node_is_min()` are more generally useful for converting the results from some node measure into a mark-class object. They can be particularly useful for highlighting which node or nodes are key because they minimise or, more often, maximise some measure.

Usage

```
node_is_random(.data, select = 1)

node_is_max(node_measure, ranks = 1)

node_is_min(node_measure, ranks = 1)

node_is_mean(node_measure, ranks = 1)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`select`	Number of elements to select (as TRUE).

node_measure	An object created by a node_ measure.
ranks	The number of ranks of max or min to return. For example, ranks = 3 will return TRUE for nodes with scores equal to any of the top (or, for node_is_min(), bottom) three scores. By default, ranks = 1.

Value

A node_mark logical vector the length of the nodes in the network, giving either TRUE or FALSE for each node depending on whether the condition is matched.

See Also

Other selection: mark_select_tie

Other marks: mark_core, mark_degree, mark_diff, mark_dyads, mark_nodes, mark_select_tie, mark_ties, mark_triangles

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_is_random(ison_brandes, 2)
node_is_max(node_by_degree(ison_brandes))
node_is_min(node_by_degree(ison_brandes))
node_is_mean(node_by_degree(ison_brandes))
```

---

mark_select_tie *Marking ties based on measures*

Description

These functions return logical vectors the length of the ties in a network:

- tie_is_random() marks one or more ties at random.
- tie_is_max() and tie_is_min() are more useful for converting the results from some tie measure into a mark-class object. They can be particularly useful for highlighting which tie or ties are key because they minimise or, more often, maximise some measure.

Usage

```
tie_is_random(.data, select = 1)

tie_is_max(tie_measure)

tie_is_min(tie_measure)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
select	Number of elements to select (as TRUE).
tie_measure	An object created by a tie_ measure.

Value

A tie_mark logical vector the length of the ties in the network, giving either TRUE or FALSE for each tie depending on whether the condition is matched.

See Also

Other marks: mark_core, mark_degree, mark_diff, mark_dyads, mark_nodes, mark_select_node, mark_ties, mark_triangles

Other tie: mark_dyads, mark_ties, mark_triangles, measure_broker_tie, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen

Other selection: mark_select_node

Examples

```
tie_is_max(tie_by_betweenness(ison_brandes))
tie_is_min(tie_by_betweenness(ison_brandes))
```

---

mark_ties *Marking ties based on structural properties*

Description

These functions return logical vectors the length of the ties in a network identifying which hold certain properties or positions in the network.

- tie_is_loop() marks ties that are loops.
- tie_is_feedback() marks ties that are feedback arcs causing the network to not be acyclic.
- tie_is_bridge() marks ties that cut or act as articulation points in a network.
- tie_is_path() marks ties on a path from one node to another.

They are most useful in highlighting parts of the network that are particularly well- or poorly-connected.

Usage

```
tie_is_loop(.data)

tie_is_feedback(.data)

tie_is_bridge(.data)

tie_is_path(.data, from, to, all_paths = FALSE)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
from	The index or name of the node from which the path should start.
to	The index or name of the node to which the path should be traced.
all_paths	Whether to return a list of paths or sample just one. By default FALSE, sampling just a single path.

Value

A tie_mark logical vector the length of the ties in the network, giving either TRUE or FALSE for each tie depending on whether the condition is matched.

See Also

Other marks: mark_core, mark_degree, mark_diff, mark_dyads, mark_nodes, mark_select_node, mark_select_tie, mark_triangles

Other tie: mark_dyads, mark_select_tie, mark_triangles, measure_broker_tie, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen

Examples

```
tie_is_loop(fict_marvel)
tie_is_feedback(ison_algebra)
tie_is_bridge(ison_brandes)
ison_adolescents |>
  mutate_ties(route = tie_is_path(from = "Jane", to = 7))
```

---

mark_triangles *Marking ties based on triangular properties*

Description

These functions return logical vectors the length of the ties in a network identifying which hold certain properties or positions in the network.

- `tie_is_triangular()` marks ties that are part of triangles.
- `tie_is_cyclical()` marks ties that are part of cycles.
- `tie_is_triplet()` marks ties that are part of transitive triplets.
- `tie_is_simmelian()` marks ties that are both in a triangle and fully reciprocated.
- `tie_is_imbalanced()` marks ties that are part of imbalanced triads.
- `tie_is_transitive()` marks ties that complete transitive closure.

They are most useful in highlighting parts of the network that are cohesively connected.

Usage

```
tie_is_triangular(.data)

tie_is_transitive(.data)

tie_is_triplet(.data)

tie_is_cyclical(.data)

tie_is_simmelian(.data)

tie_is_imbalanced(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `tie_mark` logical vector the length of the ties in the network, giving either TRUE or FALSE for each tie depending on whether the condition is matched.

See Also

Other marks: `mark_core`, `mark_degree`, `mark_diff`, `mark_dyads`, `mark_nodes`, `mark_select_node`, `mark_select_tie`, `mark_ties`

Other tie: `mark_dyads`, `mark_select_tie`, `mark_ties`, `measure_broker_tie`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`

Other cohesion: `measure_breadth`, `measure_cohesion`, `measure_fragmentation`, `motif_net`, `motif_node`

Examples

```
ison_monks |> to_uniplex("like") |>
  mutate_ties(tri = tie_is_triangular())
ison_adolescents |> to_directed() |>
  mutate_ties(trans = tie_is_transitive())
ison_adolescents |> to_directed() |>
  mutate_ties(trip = tie_is_triplet())
ison_adolescents |> to_directed() |>
  mutate_ties(cyc = tie_is_cyclical())
ison_monks |> to_uniplex("like") |>
  mutate_ties(simmel = tie_is_simmelian())
fict_marvel |> to_uniplex("relationship") |> tie_is_imbalanced()
```

---

measure_assort_net *Measures of network assortativity*

Description

These functions offer ways to measure the distribution or assortativity of ties in a network:

- `net_by_heterophily()` measures how embedded nodes in the network are within groups of nodes with the same attribute.
- `net_by_homophily()` measures how embedded nodes in the network are within groups of nodes with the same attribute, but with more options for method.
- `net_by_assortativity()` measures the degree assortativity in a network.
- `net_by_spatial()` measures the spatial association/autocorrelation (global Moran's I) in a network.

Note that for two-mode networks, homophily is calculated on the mode with the attribute of interest, and the other mode is ignored. If the attribute is present on both modes, homophily is calculated on the first mode by default, but a message is given and the user can choose to calculate homophily on the other mode instead by subsetting the attribute vector and converting the network to a one-mode network.

Usage

```
net_by_heterophily(.data, attribute)

net_by_homophily(
  .data,
  attribute,
  assortativity = c("ie", "ei", "yule", "geary")
)

net_by_assortativity(.data)

net_by_spatial(.data, attribute)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`attribute`	Name of a nodal attribute, mark, measure, or membership vector.
`assortativity`	Which method to use for `*_homophily()`. Either "ie" (negative E-I index), "ei" (E-I index), "yule" (Yule's Q), or "geary" (Geary's C). Default is "ie". The E-I index is the number of ties between (or *external*) nodes grouped in some mutually exclusive categories minus the number of ties within (or *internal*) these groups divided by the total number of ties. This value can range from 1 to -1, where 1 indicates ties only between categories/groups and -1 ties only within categories/groups. Yule's Q is a measure of association for two binary variables, calculated as: $$Q = \frac{ad - bc}{ad + bc}$$ where a is the number of ties between nodes in the same category, b is the number of ties between nodes in different categories, c is the number of non-ties between nodes in the same category, and d is the number of non-ties between nodes in different categories. This value can range from -1 to 1, where 1 indicates perfect association (all ties are between nodes in the same category), -1 indicates perfect disassociation (all ties are between nodes in different categories), and 0 indicates no association. Geary's C is a measure of spatial autocorrelation, calculated as: $$C = \frac{(n-1)\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(x_i - x_j)^2}{2W\sum_{i=1}^{n}(x_i - \bar{x})^2}$$ where n is the number of nodes, w_{ij} is the weight of the tie between nodes i and j, x_i is the attribute value of node i, $\bar{x}$ is the mean attribute value across all nodes, and W is the sum of all tie weights. This value can range from 0 to 2, where values less than 1 indicate positive autocorrelation (similar values are more likely to be connected), values greater than 1 indicate negative autocorrelation (dissimilar values are more likely to be connected), and a value of 1 indicates no autocorrelation. If an incompatible method is chosen for the attribute type, a suitable alternative will be used instead with a message.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Heterophily

Given a partition of a network into a number of mutually exclusive groups then The E-I index is the number of ties between (or *external*) nodes grouped in some mutually exclusive categories minus the number of ties within (or *internal*) these groups divided by the total number of ties. This value can range from 1 to -1, where 1 indicates ties only between categories/groups and -1 ties only within categories/groups.

Spatial autocorrelation

Moran's I is conventionally read on $[-1, 1]$, where positive values indicate that tied nodes hold similar values and negative values that they hold dissimilar ones. Its actual bounds, however, are set by the eigenvalues of the weight matrix, and on the unstandardised weights used here it can fall outside that interval. Its range is therefore declared open at both ends, and the conventional interval read as a guide rather than a guarantee.

References

On heterophily:

Krackhardt, David, and Robert N. Stern. 1988. Informal networks and organizational crises: an experimental simulation. *Social Psychology Quarterly* 51(2): 123-140. doi:10.2307/2786835

McPherson, Miller, Lynn Smith-Lovin, and James M. Cook. 2001. "Birds of a Feather: Homophily in Social Networks". *Annual Review of Sociology*, 27(1): 415-444. doi:10.1146/annurev.soc.27.1.415

On assortativity:

Newman, Mark E.J. 2002. "Assortative mixing in networks". *Physical Review Letters*, 89(20): 208701. doi:10.1103/physrevlett.89.208701

On spatial autocorrelation:

Moran, Patrick Alfred Pierce. 1950. "Notes on continuous stochastic phenomena". *Biometrika* 37(1): 17-23. doi:10.2307/2332142

See Also

Other diversity: measure_assort_node, measure_diverse_net, measure_diverse_node, motif_composition, motif_homophily

Other measures: measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
marvel_friends <- to_unsigned(to_uniplex(fict_marvel, "relationship"), "positive")
net_by_heterophily(marvel_friends, "Gender")
net_by_heterophily(marvel_friends, "Attractive")
net_by_homophily(marvel_friends, "Gender")
```

```
net_by_assortativity(ison_networkers)
net_by_spatial(ison_lawfirm, "age")
```

`measure_assort_node` *Measures of nodes assortativity*

Description

These functions offer ways to measure nodes' assortativity in a network:

- `node_by_heterophily()` measures each node's embeddedness within groups of nodes with the same attribute.
- `node_by_homophily()` measures each node's embeddedness within groups of nodes with the same attribute, using a variety of methods.

Usage

```
node_by_heterophily(.data, attribute)

node_by_homophily(
  .data,
  attribute,
  assortativity = c("ie", "ei", "yule", "geary")
)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`attribute`	Name of a nodal attribute, mark, measure, or membership vector.
`assortativity`	Which method to use for `*_homophily()`. Either "ie" (negative E-I index), "ei" (E-I index), "yule" (Yule's Q), or "geary" (Geary's C). Default is "ie". The E-I index is the number of ties between (or *external*) nodes grouped in some mutually exclusive categories minus the number of ties within (or *internal*) these groups divided by the total number of ties. This value can range from 1 to -1, where 1 indicates ties only between categories/groups and -1 ties only within categories/groups. Yule's Q is a measure of association for two binary variables, calculated as: $$Q = \frac{ad - bc}{ad + bc}$$ where a is the number of ties between nodes in the same category, b is the number of ties between nodes in different categories, c is the number of non-ties between nodes in the same category, and d is the number of non-ties between

nodes in different categories. This value can range from -1 to 1, where 1 indicates perfect association (all ties are between nodes in the same category), -1 indicates perfect disassociation (all ties are between nodes in different categories), and 0 indicates no association.

Geary's C is a measure of spatial autocorrelation, calculated as:

$$C = \frac{(n-1)\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(x_i - x_j)^2}{2W\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where n is the number of nodes, w_{ij} is the weight of the tie between nodes i and j, x_i is the attribute value of node i, $\bar{x}$ is the mean attribute value across all nodes, and W is the sum of all tie weights. This value can range from 0 to 2, where values less than 1 indicate positive autocorrelation (similar values are more likely to be connected), values greater than 1 indicate negative autocorrelation (dissimilar values are more likely to be connected), and a value of 1 indicates no autocorrelation. If an incompatible method is chosen for the attribute type, a suitable alternative will be used instead with a message.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

See Also

Other diversity: `measure_assort_net`, `measure_diverse_net`, `measure_diverse_node`, `motif_composition`, `motif_homophily`

Other measures: `measure_assort_net`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
marvel_friends <- to_unsigned(to_uniplex(fict_marvel, "relationship"), "positive")
node_by_heterophily(marvel_friends, "Gender")
node_by_heterophily(marvel_friends, "Attractive")
```

---

measure_breadth *Measures of network breadth*

Description

These functions return values or vectors relating to how broad a network is.

- `net_by_diameter()` measures the maximum path length in the network.
- `net_by_length()` measures the average path length in the network.

Usage

```
net_by_diameter(.data)

net_by_length(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Signed networks

Both measures count path lengths, and a negative tie is hostility rather than a channel along which cohesion travels. Where the network is signed, they therefore consider only the positive ties. Use `manynet::to_unsigned()` first to control this yourself.

Note that dropping the negative ties can disconnect the network, in which case the measure covers the reachable pairs only.

See Also

Other cohesion: mark_triangles, measure_cohesion, measure_fragmentation, motif_net, motif_node

Other measures: measure_assort_net, measure_assort_node, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
net_by_diameter(fict_marvel)
net_by_diameter(to_giant(fict_marvel))
net_by_length(fict_marvel)
net_by_length(to_giant(fict_marvel))
```

---

measure_brokerage *Measures of brokerage*

Description

These functions include ways to measure nodes' brokerage activity and exclusivity in a network:

- `node_by_brokering_activity()` measures nodes' brokerage activity.
- `node_by_brokering_exclusivity()` measures nodes' brokerage exclusivity.

Usage

```
node_by_brokering_activity(.data, membership)

node_by_brokering_exclusivity(.data, membership)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`membership`	A character string naming an existing node attribute in the network, or a categorical vector of the same length as the number of nodes in the network where each element indicates the group membership of the corresponding node. While this may often be a vector created using `node_in_*()` functions, it can be any character vector that assigns nodes to groups or categories.

Value

A node_measure numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

References

On brokerage activity and exclusivity:

Hamilton, Matthew, Jacob Hileman, and Orjan Bodin. 2020. "Evaluating heterogeneous brokerage: New conceptual and methodological approaches and their application to multi-level environmental governance networks" *Social Networks* 61: 1-10. doi:10.1016/j.socnet.2019.08.002

See Also

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Other brokerage: measure_broker_node, measure_broker_tie, member_brokerage, motif_brokerage_net, motif_brokerage_node

Examples

```
node_by_brokering_exclusivity(ison_networkers, "Discipline")
```

---

measure_broker_node *Measuring nodes brokerage*

Description

These function provide different measures of the degree to which nodes fill structural holes, as outlined in Burt (1992):

- `node_by_bridges()` measures the sum of bridges to which each node is adjacent.
- `node_by_redundancy()` measures the redundancy of each nodes' contacts.
- `node_by_effsize()` measures nodes' effective size.
- `node_by_efficiency()` measures nodes' efficiency.
- `node_by_constraint()` measures nodes' constraint scores for one-mode networks according to Burt (1992) and for two-mode networks according to Hollway et al (2020).
- `node_by_hierarchy()` measures nodes' exposure to hierarchy, where only one or two contacts are the source of closure.
- `node_by_neighbours_degree()` measures nodes' average nearest neighbors degree, or knn, a measure of the type of local environment a node finds itself in

Burt's theory holds that while those nodes embedded in dense clusters of close connections are likely exposed to the same or similar ideas and information, those who fill structural holes between two otherwise disconnected groups can gain some comparative advantage from that position.

Usage

```
node_by_bridges(.data)

node_by_redundancy(.data)

node_by_effsize(.data)

node_by_efficiency(.data)

node_by_constraint(.data)

node_by_hierarchy(.data)

node_by_neighbours_degree(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Details

A number of different ways of measuring these structural holes are available. Note that we use Borgatti's reformulation for unweighted networks in `node_redundancy()` and `node_effsize()`. Redundancy is thus $\frac{2t}{n}$, where t is the sum of ties and n the sum of nodes in each node's neighbourhood, and effective size is calculated as $n - \frac{2t}{n}$. Node efficiency is the node's effective size divided by its degree.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Constraint

Constraint has a natural floor at 0, for a node whose contacts are wholly unconnected to one another, but no clean ceiling: the standard result is that it can reach around 1.125 for one-mode networks, and the two-mode form is a different summation again. Its declared range is therefore left open above rather than asserting a bound the measure can exceed.

References

On structural holes:

Burt, Ronald S. 1992. *Structural Holes: The Social Structure of Competition*. Cambridge, MA: Harvard University Press.

Borgatti, Steven. 1997. "Structural Holes: Unpacking Burt's Redundancy Measures" *Connections* 20(1):35-38.

Burchard, Jake, and Benjamin Cornwell. 2018. "Structural Holes and Bridging in Two-Mode Networks." *Social Networks* 55:11–20. doi:10.1016/j.socnet.2018.04.001

Hollway, James, Jean-Frédéric Morin, and Joost Pauwelyn. 2020. "Structural conditions for novelty: The introduction of new environmental clauses to the trade regime complex." *International Environmental Agreements: Politics, Law and Economics* 20 (1): 61–83. doi:10.1007/s10784019-094645

On neighbours average degree:

Barrat, Alain, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. 2004. "The architecture of complex weighted networks", *Proc. Natl. Acad. Sci.* 101: 3747.

See Also

Other brokerage: `measure_broker_tie`, `measure_brokerage`, `member_brokerage`, `motif_brokerage_net`, `motif_brokerage_node`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`,

measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_bridges(ison_adolescents)
node_by_bridges(ison_southern_women)
node_by_redundancy(ison_adolescents)
node_by_redundancy(ison_southern_women)
node_by_effsize(ison_adolescents)
node_by_effsize(ison_southern_women)
node_by_efficiency(ison_adolescents)
node_by_efficiency(ison_southern_women)
node_by_constraint(ison_southern_women)
node_by_hierarchy(ison_adolescents)
node_by_hierarchy(ison_southern_women)
```

---

measure_broker_tie *Measuring ties brokerage*

Description

`tie_by_cohesion()` measures the ratio between common neighbors to ties' adjacent nodes and the total number of adjacent nodes, where high values indicate ties' embeddedness in dense local environments.

A tie whose two endpoints have no other neighbours has nothing to be embedded in, and so returns `NaN` rather than 0.

Usage

```
tie_by_cohesion(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `tie_measure` numeric vector the length of the ties in the network, providing the scores for each tie. If the network is labelled, then the scores will be labelled with the ties' adjacent nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

See Also

Other brokerage: measure_broker_node, measure_brokerage, member_brokerage, motif_brokerage_net, motif_brokerage_node

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other tie: mark_dyads, mark_select_tie, mark_ties, mark_triangles, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen

Examples

```
tie_by_cohesion(ison_adolescents)
```

---

measure_centralisation_between

Measuring networks betweenness-like centralisation

Description

- net_by_betweenness() measures the betweenness centralization for a network as a single score.
- mode_by_betweenness() measures betweenness centralization separately for each mode of a two-mode network, returning one score per mode (following Borgatti and Everett, 1997).

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. to_undirected() functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

For two-mode networks the two modes have different theoretical maxima, so net_by_betweenness() reports a single network-level score by applying Freeman's general centralization index over the normalized node betweenness scores, whereas mode_by_betweenness() reports the per-mode scores directly.

Usage

```
net_by_betweenness(.data, normalized = TRUE)

mode_by_betweenness(.data, normalized = TRUE, direction = c("all", "in"))
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
direction	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. By default "all".

Details

Betweenness centralisation has no directional variants: igraph::centr_betw() derives directedness from the network itself, so net_by_betweenness() takes no direction argument. For the per-mode scores, direction chooses the comparison set rather than a tie direction — "all" compares each mode's most central node against every node in the network, whereas "in" compares it only against the other nodes of its own mode. Since a two-mode incidence structure gives these no distinct "out" counterpart, mode_by_betweenness() accepts only "all" and "in".

Value

net_by_betweenness() returns a network_measure scalar; mode_by_betweenness() returns a mode_measure numeric vector of length two, giving one centralization score per mode.

References

Borgatti, Stephen P., and Martin G. Everett. 1997. "Network analysis of 2-mode data." *Social Networks* 19(3): 243-269. doi:10.1016/S03788733(96)003012

See Also

Other betweenness: measure_central_between, measure_central_tie_between

Other centrality: measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_centralisation_close, measure_centralisation_degree, measure_centralisation_eigen

Examples

```
net_by_betweenness(ison_southern_women)
mode_by_betweenness(ison_southern_women, direction = "in")
```

`measure_centralisation_close` *Measuring networks closeness-like centralisation*

Description

- `net_by_closeness()` measures a network's closeness centralization as a single score.
- `mode_by_closeness()` measures closeness centralization separately for each mode of a two-mode network, returning one score per mode (following Borgatti and Everett, 1997).
- `net_by_reach()` measures a network's reach centralization.
- `net_by_decay()` measures a network's decay centralization.
- `net_by_harmonic()` measures a network's harmonic centralization.
- `net_by_integration()` measures a network's integration centralization.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

For two-mode networks the two modes have different theoretical maxima, so `net_by_closeness()` reports a single network-level score by applying Freeman's general centralization index over the normalized node closeness scores, whereas `mode_by_closeness()` reports the per-mode scores directly.

Usage

```
net_by_closeness(.data, normalized = TRUE, direction = c("all", "out", "in"))

mode_by_closeness(.data, normalized = TRUE, direction = c("all", "out", "in"))

net_by_reach(.data, normalized = TRUE, cutoff = 2)

net_by_decay(.data, normalized = TRUE, decay = 0.5, direction = c("out", "in"))

net_by_integration(.data, normalized = TRUE, direction = c("in", "out"))

net_by_harmonic(.data, normalized = TRUE, cutoff = 2)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

`direction` Character string, “out” bases the measure on outgoing ties, “in” on incoming ties, and "all" on either/the sum of the two. By default "all".

`cutoff` Integer scalar, the maximum path length considered. Paths longer than this are ignored, which restricts the measure to a node’s local neighbourhood. Where a measure is defined over all paths by default, a negative value or `NULL` imposes no limit.

`decay` A proportion between 0 and 1 giving how much of a contribution survives each additional step of distance or walk length. Lower values discount more steeply, so that only nearby others count; higher values discount less, so that longer walks continue to contribute. The measures that take a `decay` differ in what they discount and in what value leaves the measure in its most familiar form, so each documents its own default.

Value

`net_by_*()` functions return a `network_measure` scalar; `mode_by_closeness()` returns a `mode_measure` numeric vector of length two, giving one centralization score per mode.

Decay and integration centralization

Unlike reach centrality, decay and integration scores are not bounded above by $N - 1$: integration scores scale with the network’s diameter. Freeman’s index therefore cannot use the same denominator as `net_by_reach()`, which would return negative values. Instead these apply the general centralization index over the *normalized* node scores, each of which lies in $[0, 1]$, so the numerator’s maximum is $N - 1$ and the result is guaranteed to lie in $[0, 1]$. This is the same approach `net_by_closeness()` takes for two-mode networks.

References

Borgatti, Stephen P., and Martin G. Everett. 1997. "Network analysis of 2-mode data." *Social Networks* 19(3): 243-269. doi:10.1016/S03788733(96)003012

See Also

Other closeness: `measure_central_close`, `measure_central_tie_close`

Other centrality: `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_central_tie_eigen`, `measure_centralisation_between`, `measure_centralisation_degree`, `measure_centralisation_eigen`

Examples

```
net_by_closeness(ison_southern_women, direction = "in")
mode_by_closeness(ison_southern_women, direction = "in")
net_by_decay(ison_adolescents)
net_by_integration(ison_adolescents)
```

`measure_centralisation_degree` *Measuring networks degree-like centralisation*

Description

- `net_by_degree()` measures a network's degree centralization as a single score; there are several related shortcut functions:
  - `net_by_indegree()` returns the `direction = 'in'` results.
  - `net_by_outdegree()` returns the `direction = 'out'` results.
- `mode_by_degree()` measures degree centralization separately for each mode of a two-mode network, returning one score per mode (following Borgatti and Everett, 1997); it has the same shortcuts:
  - `mode_by_indegree()` returns the `direction = 'in'` results.
  - `mode_by_outdegree()` returns the `direction = 'out'` results.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

For two-mode networks, the two modes have different theoretical maxima, so `net_by_degree()` reports a single network-level score by applying Freeman's general centralization index over the mode-normalized node degrees, whereas `mode_by_degree()` reports the per-mode centralization scores directly. For the per-mode scores, "all" uses as numerator the sum of differences between the maximum centrality score for the mode against all other centrality scores in the network, whereas "in" uses as numerator the sum of differences between the maximum centrality score for the mode against only the centrality scores of the other nodes in that mode.

Usage

```
net_by_degree(.data, normalized = TRUE, direction = c("all", "out", "in"))

mode_by_degree(.data, normalized = TRUE, direction = c("all", "out", "in"))

net_by_outdegree(.data, normalized = TRUE)

net_by_indegree(.data, normalized = TRUE)

mode_by_outdegree(.data, normalized = TRUE)

mode_by_indegree(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
direction	Character string, “out” bases the measure on outgoing ties, “in” on incoming ties, and "all" on either/the sum of the two. By default "all".

Value

`net_by_*()` functions return a `network_measure` scalar; `mode_by_*()` functions return a `mode_measure` numeric vector of length two, giving one centralization score per mode.

References

On centralisation:

Freeman, Linton C. 1978. "Centrality in social networks: Conceptual clarification". *Social Networks* 1(3): 215-239. doi:10.1016/03788733(78)900217

On two-mode centralisation:

Borgatti, Stephen P., and Martin G. Everett. 1997. "Network analysis of 2-mode data." *Social Networks* 19(3): 243-269. doi:10.1016/S03788733(96)003012

See Also

Other degree: `mark_degree`, `measure_central_degree`, `measure_central_tie_degree`

Other centrality: `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_central_tie_eigen`, `measure_centralisation_between`, `measure_centralisation_close`, `measure_centralisation_eigen`

Examples

```
net_by_degree(ison_southern_women, direction = "in")
mode_by_degree(ison_southern_women, direction = "in")
```

---

measure_centralisation_eigen

Measuring networks eigenvector-like centralisation

Description

- `net_by_eigenvector()` measures the eigenvector centralization for a network as a single score.
- `mode_by_eigenvector()` measures eigenvector centralization separately for each mode of a two-mode network (via projection to each mode), returning one score per mode (following Borgatti and Everett, 1997).

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the

salient properties using e.g. to_undirected() functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

For two-mode networks the two modes have different theoretical maxima, so net_by_eigenvector() reports a single network-level score by applying Freeman's general centralization index over the normalized node eigenvector scores, whereas mode_by_eigenvector() reports the per-mode scores directly.

Usage

```
net_by_eigenvector(.data, normalized = TRUE)

mode_by_eigenvector(.data, normalized = TRUE)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

net_by_eigenvector() returns a network_measure scalar; mode_by_eigenvector() returns a mode_measure numeric vector of length two, giving one centralization score per mode.

References

Borgatti, Stephen P., and Martin G. Everett. 1997. "Network analysis of 2-mode data." *Social Networks* 19(3): 243-269. doi:10.1016/S03788733(96)003012

See Also

Other eigenvector: measure_central_eigen, measure_central_tie_eigen

Other centrality: measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_centralisation_between, measure_centralisation_close, measure_centralisation_degree

Examples

```
net_by_eigenvector(ison_southern_women)
mode_by_eigenvector(ison_southern_women)
```

`measure_central_between`

Measuring nodes betweenness-like centrality

Description

These functions calculate common betweenness-related centrality measures for one- and two-mode networks:

- `node_by_betweenness()` measures the betweenness centralities of nodes in a network.
- `node_by_induced()` measures the induced betweenness centralities of nodes in a network.
- `node_by_flow()` measures the flow betweenness centralities of nodes in a network, which uses an electrical current model for information spreading in contrast to the shortest paths model used by normal betweenness centrality.
- `node_by_stress()` measures the stress centrality of nodes in a network.

These four differ in what they count: `node_by_betweenness()` sums the *proportion* of shortest paths between each pair that run through a node, so every pair of nodes contributes at most one unit however many shortest paths connect it; `node_by_stress()` instead sums the raw *count* of those paths, so pairs joined by many equally short routes count for more; `node_by_flow()` abandons shortest paths altogether for maximum flow, crediting nodes that carry traffic along longer routes as well; and `node_by_induced()` asks a different question again — not how much passes through a node, but how much total betweenness the network would lose if it were removed. For ties rather than nodes, see `tie_by_betweenness()`.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalised or scaled measures where available, reported when the measure is printed.

Usage

```
node_by_betweenness(.data, normalized = TRUE, cutoff = NULL)

node_by_induced(.data, normalized = TRUE, cutoff = NULL)

node_by_flow(.data, normalized = TRUE)

node_by_stress(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

`cutoff` The maximum path length to consider when calculating betweenness. If negative or NULL (the default), there's no limit to the path lengths considered.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Betweenness centrality

Betweenness centrality is based on the number of shortest paths between other nodes that a node lies upon:

$$C_B(i) = \sum_{j,k:j\neq k, j\neq i, k\neq i} \frac{g_{jik}}{g_{jk}}$$

Setting `cutoff` counts only those shortest paths no longer than k, which elsewhere goes by *distance-bounded betweenness* (Brandes, 2008) or *range-limited betweenness* (Ercsey-Ravasz et al., 2012). Normalization still applies, so a bounded score remains comparable across networks.

Induced centrality

Induced centrality concerns the change in total betweenness centrality between networks with and without a given node:

$$C_I(i) = C_B(G) - C_B(G\ i)$$

This "remove the node and re-measure" logic is the general *delta centrality* framework of Latora and Marchiori (2007); `node_by_induced()` is its betweenness instance, and `node_by_vitality()` its closeness instance.

Flow betweenness centrality

Flow betweenness centrality concerns the total maximum flow, f, between other nodes j, k in a network G that a given node mediates:

$$C_F(i) = \sum_{j,k:j\neq k, j\neq i, k\neq i} f(j,k,G) - f(j,k,G\ i)$$

When normalized (by default) this sum of differences is divided by the sum of flows $f(i,j,G)$.

Stress centrality

Stress centrality is the number of all shortest paths or geodesics, g, between other nodes that a given node mediates:

$$C_S(i) = \sum_{j,k:j\neq k, j\neq i, k\neq i} g_{jik}$$

High stress nodes lie on a large number of shortest paths between other nodes, and thus associated with bridging or spanning boundaries.

References

On betweenness centrality:

Freeman, Linton. 1977. "A set of measures of centrality based on betweenness". *Sociometry*, 40(1): 35–41. doi:10.2307/3033543

On bounding path length:

Brandes, Ulrik. 2008. "On variants of shortest-path betweenness centrality and their generic computation". *Social Networks* 30(2): 136-145. doi:10.1016/j.socnet.2007.11.001

Ercsey-Ravasz, Maria, Ryan N. Lichtenwalter, Nitesh V. Chawla, and Zoltan Toroczkai. 2012. "Range-limited centrality measures in complex networks". *Physical Review E* 85(6): 066103. doi:10.1103/PhysRevE.85.066103

On induced centrality:

Everett, Martin and Steve Borgatti. 2010. "Induced, endogenous and exogenous centrality" *Social Networks*, 32: 339-344. doi:10.1016/j.socnet.2010.06.004

On delta centrality:

Latora, Vito, and Massimo Marchiori. 2007. "A measure of centrality based on network efficiency". *New Journal of Physics* 9(6): 188. doi:10.1088/13672630/9/6/188

On flow centrality:

Freeman, Linton C., Stephen P. Borgatti, and Douglas R. White. 1991. "Centrality in Valued Graphs: A Measure of Betweenness Based on Network Flow". *Social Networks*, 13(2), 141-154. doi:10.1016/03788733(91)90017N

Koschutzki, D., K.A. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski. 2005. "Centrality Indices". In U. Brandes and T. Erlebach (eds.), *Network Analysis: Methodological Foundations*. Berlin: Springer.

On stress centrality:

Shimbel, A. 1953. "Structural Parameters of Communication Networks". *Bulletin of Mathematical Biophysics*, 15:501-507. doi:10.1007/BF02476438

See Also

Other betweenness: `measure_central_tie_between`, `measure_centralisation_between`

Other centrality: `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_centralisation_between`, `measure_centralisation_close`, `measure_centralisation_degree`, `measure_centralisation_eigen`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_betweenness(ison_southern_women)
node_by_induced(ison_adolescents)
```

`measure_central_close` *Measuring nodes closeness-like centrality*

Description

These functions calculate common closeness-related centrality measures that rely on path-length for one- and two-mode networks:

- `node_by_closeness()` measures the closeness centrality of nodes in a network.
- `node_by_harmonic()` measures nodes' harmonic centrality or valued centrality, which is thought to behave better than reach centrality for disconnected networks.
- `node_by_reach()` measures nodes' reach centrality, or how many nodes they can reach within *k* steps.
- `node_by_decay()` measures nodes' decay centrality, a distance-weighted generalisation of reach centrality.
- `node_by_integration()` measures nodes' integration or radiality, which weights alters by how close they are rather than counting them; `node_by_radiality()` returns the `direction = 'out'` results. Note that on a connected network integration ranks nodes identically to closeness centrality, of which it is an affine transformation; it differs only in how it treats unreachable nodes.
- `node_by_information()` measures nodes' information centrality or current-flow closeness centrality.
- `node_by_eccentricity()` measures nodes' eccentricity or maximum distance from another node in the network.
- `node_by_distance()` measures nodes' geodesic distance from or to a given node.
- `node_by_vitality()` measures a network's closeness vitality centrality, or the change in closeness centrality between networks with and without a given node.
- `node_by_randomwalk()` measures nodes' random walk closeness centrality, or the inverse of the average time a random walk takes to reach them.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalised or scaled measures where available, reported when the measure is printed. Most of these measures are *normalised* against a theoretical maximum, so that scores can be compared across networks; `node_by_randomwalk()` and `node_by_distance()` have no such maximum and are instead *scaled* against the largest value observed in this network.

Usage

```
node_by_closeness(
  .data,
  normalized = TRUE,
  direction = c("out", "in", "all"),
  cutoff = NULL
)

node_by_harmonic(
  .data,
  normalized = TRUE,
  cutoff = -1,
  decay = NULL,
  direction = c("out", "in")
)

node_by_reach(.data, normalized = TRUE, cutoff = 2)

node_by_decay(
  .data,
  normalized = TRUE,
  decay = 0.5,
  direction = c("out", "in")
)

node_by_integration(.data, normalized = TRUE, direction = c("in", "out"))

node_by_radiality(.data, normalized = TRUE)

node_by_information(.data, normalized = TRUE)

node_by_eccentricity(.data, normalized = TRUE)

node_by_distance(.data, from, to, normalized = TRUE)

node_by_vitality(.data, normalized = TRUE)

node_by_randomwalk(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
`direction`	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. By default "all".
`cutoff`	Integer scalar, the maximum path length considered. Paths longer than this are ignored, which restricts the measure to a node's local neighbourhood. Where a measure is defined over all paths by default, a negative value or `NULL` imposes no limit.
`decay`	A proportion between 0 and 1 giving how much of a contribution survives each additional step of distance or walk length. Lower values discount more steeply, so that only nearby others count; higher values discount less, so that longer walks continue to contribute. The measures that take a `decay` differ in what they discount and in what value leaves the measure in its most familiar form, so each documents its own default.
`from`, `to`	Index or name of a node to calculate distances from or to.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Closeness centrality

Closeness centrality is also known as status centrality, barycenter centrality, or the Sabidussi index. It is defined as the reciprocal of the farness or distance, d, from a node to all other nodes in the network:

$$C_C(i) = \frac{1}{\sum_j d(i,j)}$$

When (more commonly) normalised, the numerator is instead $N - 1$.

Harmonic centrality

Harmonic centrality or valued centrality reverses the sum and reciprocal operations compared to closeness centrality:

$$C_H(i) = \sum_{i,i \neq j} \frac{1}{d(i,j)}$$

where $\frac{1}{d(i,j)} = 0$ where there is no path between i and j. Normalization is by $N - 1$. Since the harmonic mean performs better than the arithmetic mean on unconnected networks, i.e. networks with infinite distances, harmonic centrality is to be preferred in these cases.

Harmonic centrality sums a decreasing function of each distance, $\sum_j f(d(i,j))$, and setting `decay` simply swaps in a different such function, δ^d, giving decay centrality (see below). Note that `node_by_closeness()` cannot be reached this way: it sums the distances and inverts once, $1/\sum_j d(i,j)$, which is a different order of aggregation that no choice of decay function reproduces.

Reach centrality

In some cases, longer path lengths are irrelevant and 'closeness' should be defined as how many others are in a local neighbourhood. How many steps out this neighbourhood should be defined as is given by the 'cutoff' parameter. This is usually termed k or m in equations, which is why this is sometimes called (m- or) k-step reach centrality:

$$C_R(i) = \sum_j d(i,j) \leq k$$

The maximum reach score is $N-1$, achieved when the node can reach all other nodes in the network in k steps or less, but the normalised version, $\frac{C_R}{N-1}$, is more common. Note that if $k = 1$ (i.e. cutoff = 1), then this returns the node's degree. At higher cutoff reach centrality returns the size of the node's component. Counting the others reachable by a geodesic of length at most k is also known as *geodesic k-path centrality* (Borgatti and Everett, 2006); note that it is not the same as the k-path indices that count paths rather than nodes.

Decay centrality

Here `decay` defaults to 0.5, so that each additional step halves a node's contribution. As it approaches 0 this approaches degree centrality, and as it approaches 1 the size of the node's component.

Where reach centrality counts how many others are within a fixed number of steps, decay centrality weights every reachable other by how far away they are, so that nearer nodes count for more:

$$C_D(i) = \sum_{j, j \neq i} \delta^{d(i,j)-1}$$

where δ is the decay parameter and unreachable nodes contribute nothing. This avoids having to choose a single cutoff, since the contribution of distant nodes tapers off smoothly rather than being truncated. Normalization is by $N - 1$, the score achieved when a node is adjacent to all others.

Integration and radiality

Integration centrality, also known as radiality, inverts the usual farness logic: instead of summing distances, it sums how much *closer* than the network's diameter each other node is:

$$C_I(i) = \sum_{j, j \neq i} (\Delta - d(i,j) + 1)$$

where Δ is the maximum finite distance in the network. Nodes that are near to many others therefore score highly, while unreachable pairs contribute nothing. Normalization is by $(N-1)\Delta$.

Valente and Foreman distinguish the two directions: *integration* is calculated on incoming ties, capturing how well a node is reached by others, whereas *radiality* is calculated on outgoing ties, capturing how well a node reaches others. Use `direction` to choose; in undirected networks they coincide.

Information centrality

Information centrality, also known as current-flow centrality, is a hybrid measure relating to both path-length and walk-based measures. The information centrality of a node is the harmonic average of the "bandwidth" or inverse path-length for all paths originating from the node.

As described in the `{sna}` package, information centrality works on an undirected but potentially weighted network excluding isolates (which take scores of zero). It is defined as:

$$C_I = \frac{1}{T + \frac{\sum T - 2 \sum C_1}{|N|}}$$

where $C = B^{-}1$ with B is a pseudo-adjacency matrix replacing the diagonal of $1 - A$ with $1 + k$, and T is the trace of C and S_R an arbitrary row sum (all rows in C have the same sum).

Nodes with higher information centrality have a large number of short paths to many others in the network, and are thus considered to have greater control of the flow of information.

Information centrality is the closeness-like member of the current-flow family; its betweenness-like counterpart is random walk (or current-flow) betweenness centrality, which netrics does not yet offer.

Eccentricity centrality

Eccentricity centrality, graph centrality, or the Koenig number, is the (if normalized, inverse of) the distance to the furthest node:

$$C_E(i) = \frac{1}{max_{j \in N} d(i, j)}$$

where the distance from i to j is ∞ if unconnected. As such it is only well defined for connected networks.

Geodesic distance

Unlike the other functions documented here, `node_by_distance()` is not a centrality index but a distance query: it reports each node's geodesic distance from (or to) one named node, rather than summarising its position with respect to the network as a whole. It is grouped here because the closeness-like centralities are all built from the same geodesic distances.

Closeness vitality centrality

The closeness vitality of a node is the change in the sum of all distances in a network, also known as the Wiener Index, when that node is removed. Since the Wiener Index of a disconnected network is infinite, the unnormalised closeness vitality of a cut node — one whose removal would disconnect the network — is negative infinity. This is a property of the definition rather than a failure of it: it picks out exactly the cut nodes. Because that is awkward to work with, the normalised version

rescales the finite scores onto $[0, 1]$ and gives cut nodes a score of 0, the endpoint that negative infinity occupies. Formally:

$$C_V(i) = \sum_{j,k} d(j,k) - \sum_{j,k} d(j,k,G\ i)$$

where $d(j,k,G\ i)$ is the distance between nodes j and k in the network with node i removed. This is the closeness instance of the *delta centrality* framework; for its betweenness instance see `node_by_induced()`.

Random walk closeness centrality

Random walk closeness centrality is based on the average length of random walks starting at all other nodes to reach a given node. It is defined as the inverse of the average hitting time to a node. This means that higher values are given to nodes that can be reached more quickly on average by random walks starting at other nodes. Formally:

$$C_{RW}(i) = \frac{1}{\frac{1}{N-1}\sum_{j \neq i} H_{ji}}$$

where H_{ji} is the hitting time from node j to node i.

References

On closeness centrality:

Sabidussi, Gert. 1966. "The centrality index of a graph". *Psychometrika*, 31(4): 581–603. doi:10.1007/BF02289527

Bavelas, Alex. 1950. "Communication Patterns in Task-Oriented Groups". *The Journal of the Acoustical Society of America*, 22(6): 725–730. doi:10.1121/1.1906679

Harary, Frank. 1959. "Status and Contrastatus". *Sociometry*, 22(1): 23–43. doi:10.2307/2785610

On harmonic centrality:

Marchiori, Massimo, and Vito Latora. 2000. "Harmony in the small-world". *Physica A* 285: 539-546. doi:10.1016/S03784371(00)003113

Dekker, Anthony. 2005. "Conceptual distance in social network analysis". *Journal of Social Structure* 6(3).

Boldi, Paolo, and Sebastiano Vigna. 2014. "Axioms for Centrality". *Internet Mathematics* 10(3-4): 222-262. doi:10.1080/15427951.2013.865686

On reach centrality:

Borgatti, Stephen P., Martin G. Everett, and J.C. Johnson. 2013. *Analyzing social networks*. London: SAGE Publications Limited.

Borgatti, Stephen P., and Martin G. Everett. 2006. "A graph-theoretic perspective on centrality". *Social Networks* 28(4): 466-484. doi:10.1016/j.socnet.2005.11.005

On decay centrality:

Jackson, Matthew O. 2008. *Social and Economic Networks*. Princeton: Princeton University Press.

On integration and radiality:

Valente, Thomas W., and Robert K. Foreman. 1998. "Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network". *Social Networks* 20(1): 89-105. doi:10.1016/S03788733(97)000075

On information centrality:

Stephenson, Karen, and Marvin Zelen. 1989. "Rethinking centrality: Methods and examples". *Social Networks* 11(1):1-37. doi:10.1016/03788733(89)900166

Brandes, Ulrik, and Daniel Fleischer. 2005. "Centrality Measures Based on Current Flow". *Proc. 22nd Symp. Theoretical Aspects of Computer Science* LNCS 3404: 533-544. doi:10.1007/9783-540318569_44

On eccentricity centrality:

Hage, Per, and Frank Harary. 1995. "Eccentricity and centrality in networks". *Social Networks*, 17(1): 57-63. doi:10.1016/03788733(94)002489

On closeness vitality centrality:

Koschuetzki, Dirk, Katharina Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. 2005. "Centrality Indices", in Brandes, Ulrik, and Thomas Erlebach (eds.). *Network Analysis: Methodological Foundations*. Springer: Berlin, pp. 16-61.

On random walk closeness centrality:

Noh, J.D. and R. Rieger. 2004. "Random Walks on Complex Networks". *Physical Review Letters*, 92(11): 118701. doi:10.1103/PhysRevLett.92.118701

See Also

Other closeness: `measure_central_tie_close`, `measure_centralisation_close`

Other centrality: `measure_central_between`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_centralisation_between`, `measure_centralisation_close`, `measure_centralisation_degree`, `measure_centralisation_eigen`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
node_by_closeness(ison_southern_women)
node_by_reach(ison_adolescents)
node_by_decay(ison_adolescents)
node_by_integration(ison_adolescents)
node_by_radiality(ison_adolescents)
```

`measure_central_degree` *Measuring nodes degree-like centrality*

Description

These functions calculate common degree-related centrality measures for one- and two-mode networks:

- `node_by_degree()` measures the degree centrality of nodes in an unweighted network, or weighted degree/strength of nodes in a weighted network; there are several related shortcut functions:
  - `node_by_deg()` returns the unnormalised results.
  - `node_by_indegree()` returns the `direction = 'in'` results.
  - `node_by_outdegree()` returns the `direction = 'out'` results.
- `node_by_multidegree()` measures the ratio between types of ties in a multiplex network.
- `node_by_leverage()` measures the leverage centrality of nodes in a network.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `manynet::to_undirected()` functions. All centrality and centralization measures return normalised or scaled measures where available, reported when the measure is printed. Note that a weighted network has no theoretical maximum degree, so `node_by_degree()` there returns *scaled* rather than normalised scores, which rank nodes within this network but are not comparable with those of another.

`node_by_multidegree()` is the one measure here that is not reached by dispatch: a multiplex network does not itself say *which* two types of tie to contrast, so `tie1` and `tie2` must be named.

Usage

```
node_by_degree(
  .data,
  normalized = TRUE,
  alpha = 0,
  direction = c("all", "out", "in")
)

node_by_deg(.data, alpha = 0, direction = c("all", "out", "in"))
```

```
node_by_outdegree(.data, normalized = TRUE, alpha = 0)

node_by_indegree(.data, normalized = TRUE, alpha = 0)

node_by_multidegree(.data, tie1, tie2)

node_by_leverage(.data)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
alpha	Numeric scalar, the positive tuning parameter introduced in Opsahl et al (2010) for trading off between degree and strength centrality measures. By default, alpha = 0, which ignores tie weights and the measure is solely based upon degree (the number of ties). alpha = 1 ignores the number of ties and provides the sum of the tie weights as strength centrality. Values between 0 and 1 reflect different trade-offs in the relative contributions of degree and strength to the final outcome, with 0.5 as the middle ground. Values above 1 penalise for the number of ties. Of two nodes with the same sum of tie weights, the node with fewer ties will obtain the higher score. This argument is ignored except in the case of a weighted network.
direction	Character string, “out” bases the measure on outgoing ties, “in” on incoming ties, and "all" on either/the sum of the two. By default "all".
tie1	Character string indicating the first uniplex network.
tie2	Character string indicating the second uniplex network.

Value

A node_measure numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes’ names.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

Multiplex networks

node_by_degree() counts every tie a node holds, whatever its layer, so a node tied twice to the same alter on two layers scores 2. To score one layer at a time, take it first with manynet::to_uniplex(), or use node_by_multidegree() to contrast two. Note that to_uniplex() drops the nodes that hold none of the retained ties, so scores from two layers are of different lengths.

Degree centrality

The degree of a node is the number of connections it has. It is also sometimes called the valency of a node, $d(v)$. The maximum degree in a network is often denoted $\Delta(G)$ and the minimum degree in a network $\delta(G)$. The total degree of a network is the sum of all degrees, $\sum_v d(v)$. The degree sequence is the set of all nodes' degrees, ordered from largest to smallest. Directed networks discriminate between outdegree (degree of outgoing ties) and indegree (degree of incoming ties).

Strength centrality

Given a weighted network, `node_by_degree()` sums tie weights rather than counting ties, which is also known as *strength centrality* or *weighted degree centrality*. The `alpha` argument tunes between the two, following Opsahl et al. (2010), and the measure reports itself as "strength centrality" whenever `alpha` is not zero.

Leverage centrality

Leverage centrality concerns the degree of a node compared with that of its neighbours, J:

$$C_L(i) = \frac{1}{d(i)} \sum_{j \in J(i)} \frac{d(i) - d(j)}{d(i) + d(j)}$$

References

On degree centrality:

Freeman, Linton C. 1978. "Centrality in social networks: Conceptual clarification". *Social Networks* 1(3): 215-239. doi:10.1016/03788733(78)900217

On multimodal centrality:

Faust, Katherine. 1997. "Centrality in affiliation networks." *Social Networks* 19(2): 157-191. doi:10.1016/S03788733(96)003000

Borgatti, Stephen P., and Martin G. Everett. 1997. "Network analysis of 2-mode data." *Social Networks* 19(3): 243-270. doi:10.1016/S03788733(96)003012

Borgatti, Stephen P., and Daniel S. Halgin. 2011. "Analyzing affiliation networks." In *The SAGE Handbook of Social Network Analysis*, edited by John Scott and Peter J. Carrington, 417–33. London, UK: Sage. doi:10.4135/9781446294413.n28

On strength centrality:

Opsahl, Tore, Filip Agneessens, and John Skvoretz. 2010. "Node centrality in weighted networks: Generalizing degree and shortest paths." *Social Networks* 32, 245-251. doi:10.1016/j.socnet.2010.03.006

On leverage centrality:

Joyce, Karen E., Paul J. Laurienti, Jonathan H. Burdette, and Satoru Hayasaka. 2010. "A New Measure of Centrality for Brain Networks". *PLoS ONE* 5(8): e12200. doi:10.1371/journal.pone.0012200

See Also

Other degree: mark_degree, measure_central_tie_degree, measure_centralisation_degree

Other centrality: measure_central_between, measure_central_close, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_centralisation_between, measure_centralisation_close, measure_centralisation_degree, measure_centralisation_eigen

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degr measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_degree(ison_southern_women)
```

`measure_central_eigen` *Measuring nodes eigenvector-like centrality*

Description

These functions calculate common eigenvector-related centrality measures, or walk-based eigen-measures, for one- and two-mode networks:

- `node_by_eigenvector()` measures the eigenvector centrality of nodes in a network.
- `node_by_power()` measures the Bonacich, beta, or power centrality of nodes in a network.
- `node_by_alpha()` measures the alpha or Katz centrality of nodes in a network.
- `node_by_pagerank()` measures the pagerank centrality of nodes in a network.
- `node_by_hub()` measures how well nodes in a network serve as hubs pointing to many authorities.
- `node_by_authority()` measures how well nodes in a network serve as authorities from many hubs.
- `node_by_subgraph()` measures nodes' participation in all closed walks in the network, weighting shorter walks more heavily.
- `node_by_posneg()` measures the PN (positive-negative) centrality of a signed network.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalised or scaled measures where available, reported when the measure is printed.

Walk-based measures are mostly unbounded, so few of them can be *normalised* against a theoretical maximum in the way that degree, closeness and betweenness can. Most are instead *scaled* against the observed maximum, which ranks nodes within one network but does not give scores that are comparable between networks.

Usage

```
node_by_eigenvector(.data, normalized = TRUE, scaled = TRUE, scale = NULL)

node_by_power(
  .data,
  normalized = TRUE,
  scaled = FALSE,
  scale = NULL,
  exponent = 1
)

node_by_alpha(.data, decay = 0.85, alpha = NULL)

node_by_pagerank(.data, decay = 0.85)

node_by_authority(.data, scaled = TRUE)

node_by_hub(.data, scaled = TRUE)

node_by_subgraph(
  .data,
  decay = 1,
  walks = c("all", "odd", "even"),
  method = NULL
)

node_by_posneg(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
`scaled`	Logical scalar, whether to divide the results by the maximum observed in this network, so that the highest-scoring node takes the value one. Note that, unlike

	normalisation against a theoretical maximum, scaled scores are not comparable across different networks.
`scale`	Deprecated; use `scaled` instead.
`exponent`	Decay rate or attentuation factor for the Bonacich power centrality score. Can be positive or negative.
`decay`	A proportion between 0 and 1 giving how much of a contribution survives each additional step of distance or walk length. Lower values discount more steeply, so that only nearby others count; higher values discount less, so that longer walks continue to contribute. The measures that take a `decay` differ in what they discount and in what value leaves the measure in its most familiar form, so each documents its own default.
`alpha`	Deprecated; use `decay` instead.
`walks`	Character string indicating which closed walks to count. By default `"all"`, which is subgraph centrality as usually defined. `"odd"` counts only walks of odd length and `"even"` only those of even length; the two sum to `"all"`. Odd closed walks cannot occur within a bipartite structure, so a node scoring near zero on `"odd"` sits in a locally two-mode-like neighbourhood. See `net_by_bipartivity()` for the network-level counterpart.
`method`	Deprecated. The former spelling of `walks`. Still accepted, but warns; please use `walks` instead.

Details

We use `{igraph}` routines behind the scenes here for consistency and because they are often faster. For example, `igraph::eigencentrality()` is approximately 25% faster than `sna::evcent()`.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Eigenvector centrality

Eigenvector centrality operates as a measure of a node's influence in a network. The idea is that being connected to well-connected others results in a higher score. Each node's eigenvector centrality can be defined as:

$$x_i = \frac{1}{\lambda} \sum_{j \in N} a_{i,j} x_j$$

where $a_{i,j} = 1$ if i is linked to j and 0 otherwise, and λ is a constant representing the principal eigenvalue. Rather than performing this iteration, most routines solve the eigenvector equation $Ax = \lambda x$. Note that since `{igraph}` v2.1.1, the values will always be rescaled so that the maximum is 1. This is not a limitation so much as a property of the measure: an eigenvector is defined only up to a scalar multiple, so its scores carry no absolute units to preserve.

Power or beta (or Bonacich) centrality

Power centrality includes an exponent that weights contributions to a node's centrality based on how far away those other nodes are.

$$c_b(i) = \sum A(i,j)(\alpha = \beta c(j))$$

Where β is positive, this means being connected to central people increases centrality. Where β is negative, this means being connected to central people decreases centrality (and being connected to more peripheral actors increases centrality). When $\beta = 0$, this is the outdegree. α is calculated to make sure the root mean square equals the network size.

Alpha centrality

Alpha centrality is also known as Katz centrality, Katz-Bonacich centrality, or Katz status. The measure is named for the α of Bonacich and Lloyd, which trades off the importance of external influence against the importance of connection: when $\alpha = 0$ only the external influence matters, and as α grows only the connectivity matters and we reduce to eigenvector centrality. Since α is a per-step discount, netrics takes it as `decay`, the name it uses for that parameter throughout; by default 0.85. It operates better than eigenvector centrality for directed networks because eigenvector centrality will return 0s for all nodes not in the main strongly-connected component. Each node's alpha centrality can be defined as:

$$x_i = \frac{1}{\lambda} \sum_{j \in N} a_{i,j} x_j + e_i$$

where $a_{i,j} = 1$ if i is linked to j and 0 otherwise, λ is a constant representing the principal eigenvalue, and e_i is some external influence used to ensure that even nodes beyond the main strongly connected component begin with some basic influence. Note that many equations replace $\frac{1}{\lambda}$ with α, hence the name.

For example, if $\alpha = 0.5$, then each direct connection (or alter) would be worth $(0.5)^1 = 0.5$, each secondary connection (or tertius) would be worth $(0.5)^2 = 0.25$, each tertiary connection would be worth $(0.5)^3 = 0.125$, and so on.

Rather than performing this iteration though, most routines solve the equation $x = (I - \frac{1}{\lambda} A^T)^{-1} e$.

Pagerank centrality

Pagerank centrality, or the PageRank citation ranking, is the stationary distribution of a random walk that at each step either follows an outgoing tie or teleports to a node chosen at random. Scores are therefore already shares that sum to one. `decay` is the probability of following a tie rather than teleporting, elsewhere called the damping factor; by default 0.85. As it approaches 0 the walk teleports at every step and all nodes score alike; as it approaches 1 the walk never teleports.

Hub and authority centrality

Hub and authority centrality are the two halves of Kleinberg's HITS (Hyperlink-Induced Topic Search) algorithm, and are computed together: good authorities are pointed to by good hubs, and good hubs point to good authorities. `node_by_hub()` and `node_by_authority()` return one each. In an undirected network the two coincide.

Subgraph centrality

Subgraph centrality measures the participation of a node in all subgraphs in the network, giving higher weight to smaller subgraphs. It is defined as:

$$C_S(i) = \sum_{k=0}^{\infty} \frac{\delta^k (A^k)_{ii}}{k!}$$

where $(A^k)_{ii}$ is the ith diagonal element of the kth power of the adjacency matrix A, representing the number of closed walks of length k starting and ending at node i. Weighting by $\frac{1}{k!}$ ensures that shorter walks contribute more to the centrality score than longer walks. The `decay` parameter δ tunes that further, discounting each step by a further factor: at the default of 1 the measure takes its usual form, and lower values concentrate it on ever shorter walks.

Subgraph centrality is a good choice of measure when the focus is on local connectivity and clustering around a node, as it captures the extent to which a node is embedded in tightly-knit groups within the network. Note though that because of the way spectral decomposition is used to calculate this measure, this is not a good measure for very large graphs.

Summing these scores over all nodes gives the network's *Estrada index*, so a node's subgraph centrality is its contribution to that index.

PN (positive-negative) centrality

PN centrality extends walk-based centrality to signed networks. Negative ties are weighted twice as heavily as positive ties, $P - 2N$, and the measure is then obtained in closed form by matrix inversion, so that — like alpha centrality, of which it is the signed analogue — it counts walks of all lengths with a length discount rather than counting only direct ties. Scores centre on 1: nodes above 1 are advantaged by their pattern of positive and negative ties, and those below 1 disadvantaged.

References

On eigenvector centrality:

Bonacich, Phillip. 1972. "Factoring and Weighting Approaches to Status Scores and Clique Identification." *The Journal of Mathematical Sociology* 2(1): 113–120. doi:10.1080/0022250X.1972.9989806

Bonacich, Phillip. 1991. "Simultaneous Group and Individual Centralities." *Social Networks* 13(2):155–68. doi:10.1016/03788733(91)90018O

On power centrality:

Bonacich, Phillip. 1987. "Power and Centrality: A Family of Measures." *The American Journal of Sociology*, 92(5): 1170–82. doi:10.1086/228631.

On alpha centrality:

Katz, Leo 1953. "A new status index derived from sociometric analysis". *Psychometrika*. 18(1): 39–43.

Bonacich, P. and Lloyd, P. 2001. "Eigenvector-like measures of centrality for asymmetric relations" *Social Networks*. 23(3):191-201.

On pagerank centrality:

Brin, Sergey and Page, Larry. 1998. "The anatomy of a large-scale hypertextual web search engine". *Proceedings of the 7th World-Wide Web Conference*. Brisbane, Australia.

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. "The PageRank Citation Ranking: Bringing Order to the Web". *Stanford InfoLab Technical Report* 1999-66.

On hub and authority centrality:

Kleinberg, Jon. 1999. "Authoritative sources in a hyperlinked environment". *Journal of the ACM* 46(5): 604–632. doi:10.1145/324133.324140

On subgraph centrality:

Estrada, Ernesto and Rodríguez-Velázquez, Juan A. 2005. "Subgraph centrality in complex networks". *Physical Review E* 71(5): 056103. doi:10.1103/PhysRevE.71.056103

On odd and even closed walks:

Estrada, Ernesto and Rodríguez-Velázquez, Juan A. 2005. "Spectral measures of bipartivity in complex networks". *Physical Review E* 72(4): 046105. doi:10.1103/PhysRevE.72.046105

On signed centrality:

Everett, Martin G., and Stephen P. Borgatti. 2014. "Networks Containing Negative Ties." *Social Networks* 38:111–20. doi:10.1016/j.socnet.2014.03.005

See Also

Other eigenvector: measure_central_tie_eigen, measure_centralisation_eigen

Other centrality: measure_central_between, measure_central_close, measure_central_degree, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_centralisation_between, measure_centralisation_close, measure_centralisation_degree, measure_centralisation_eigen

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_tie_between, measure_central_tie_close, measure_central_tie_deg measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_eigenvector(ison_southern_women)
node_by_power(ison_southern_women, exponent = 0.5)
```

`measure_central_tie_between` *Measuring ties betweenness-like centrality*

Description

`tie_by_betweenness()` measures the number of shortest paths going through a tie.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

Usage

```
tie_by_betweenness(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A `tie_measure` numeric vector the length of the ties in the network, providing the scores for each tie. If the network is labelled, then the scores will be labelled with the ties' adjacent nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Edge betweenness centrality

The betweenness centrality of a tie, also known as *edge betweenness*, counts the shortest paths between other nodes that run along it. It is best known as the quantity iteratively recomputed by the Girvan-Newman community detection algorithm, where the ties with the highest betweenness are removed first; see `node_in_betweenness()`.

References

On edge betweenness centrality:

Girvan, Michelle, and Mark E.J. Newman. 2002. "Community structure in social and biological networks". *Proceedings of the National Academy of Sciences* 99(12): 7821-7826. doi:10.1073/pnas.122653799

Brandes, Ulrik. 2001. "A faster algorithm for betweenness centrality". *Journal of Mathematical Sociology* 25(2): 163-177. doi:10.1080/0022250X.2001.9990249

See Also

Other betweenness: measure_central_between, measure_centralisation_between

Other centrality: measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_centralisation_between, measure_centralisation_close, measure_centralisation_degree, measure_centralisation_eigen

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other tie: mark_dyads, mark_select_tie, mark_ties, mark_triangles, measure_broker_tie, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen

Examples

```
(tb <- tie_by_betweenness(ison_adolescents))
ison_adolescents |> mutate_ties(weight = tb)
```

---

measure_central_tie_close

Measuring ties closeness-like centrality

Description

`tie_by_closeness()` measures the closeness of each tie to other ties in the network.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

Usage

```
tie_by_closeness(.data, normalized = TRUE)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A tie_measure numeric vector the length of the ties in the network, providing the scores for each tie. If the network is labelled, then the scores will be labelled with the ties' adjacent nodes' names.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

See Also

Other closeness: measure_central_close, measure_centralisation_close

Other centrality: measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_degree, measure_central_tie_eig measure_centralisation_between, measure_centralisation_close, measure_centralisation_degree, measure_centralisation_eigen

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other tie: mark_dyads, mark_select_tie, mark_ties, mark_triangles, measure_broker_tie, measure_central_tie_between, measure_central_tie_degree, measure_central_tie_eigen

Examples

```
(ec <- tie_by_closeness(ison_adolescents))
ison_adolescents |> mutate_ties(weight = ec)
```

---

measure_central_tie_degree

Measuring ties degree-like centrality

Description

`tie_by_degree()` measures the degree centrality of ties in a network

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

Usage

```
tie_by_degree(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A `tie_measure` numeric vector the length of the ties in the network, providing the scores for each tie. If the network is labelled, then the scores will be labelled with the ties' adjacent nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

See Also

Other degree: `mark_degree`, `measure_central_degree`, `measure_centralisation_degree`

Other centrality: `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_eige` `measure_centralisation_between`, `measure_centralisation_close`, `measure_centralisation_degree`, `measure_centralisation_eigen`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other tie: `mark_dyads`, `mark_select_tie`, `mark_ties`, `mark_triangles`, `measure_broker_tie`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_eigen`

Examples

```
tie_by_degree(ison_adolescents)
```

`measure_central_tie_eigen` *Measuring ties eigenvector-like centrality*

Description

`tie_by_eigenvector()` measures the eigenvector centrality of ties in a network.

All measures attempt to use as much information as they are offered, including whether the networks are directed, weighted, or multimodal. If this would produce unintended results, first transform the salient properties using e.g. `to_undirected()` functions. All centrality and centralization measures return normalized measures by default, including for two-mode networks.

Usage

```
tie_by_eigenvector(.data, normalized = TRUE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A `tie_measure` numeric vector the length of the ties in the network, providing the scores for each tie. If the network is labelled, then the scores will be labelled with the ties' adjacent nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

See Also

Other eigenvector: `measure_central_eigen`, `measure_centralisation_eigen`

Other centrality: `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degr` `measure_centralisation_between`, `measure_centralisation_close`, `measure_centralisation_degree`, `measure_centralisation_eigen`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`,

measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other tie: mark_dyads, mark_select_tie, mark_ties, mark_triangles, measure_broker_tie, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree

Examples

```
tie_by_eigenvector(ison_adolescents)
```

---

measure_closure *Measuring network closure*

Description

These functions offer methods for summarising the closure in configurations in one-, two-, and three-mode networks:

- `net_by_reciprocity()` measures reciprocity in a (usually directed) network.
- `net_by_transitivity()` measures transitivity in a network.
- `net_by_cyclicality()` measures cyclicality in a (necessarily directed) network.
- `net_by_equivalency()` measures equivalence or reinforcement in a (usually two-mode) network.
- `net_by_congruency()` measures congruency across two two-mode networks.

Usage

```
net_by_reciprocity(.data, variant = c("default", "ratio"), method = NULL)

net_by_transitivity(.data)

net_by_cyclicality(.data)

net_by_equivalency(.data)

net_by_congruency(.data, object2)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`variant`	Character string naming which variant of the measure to compute, where more than one definition of the same quantity is in use. The variant chosen is reported when the result is printed.

`method`	Deprecated. The former spelling of `variant`. Still accepted, but warns; please use `variant` instead.
`object2`	Optionally, a second (two-mode) matrix, igraph, or tidygraph

Details

For one-mode networks, shallow wrappers of igraph versions exist via `net_reciprocity` and `net_transitivity`.

For two-mode networks, `net_equivalency` calculates the proportion of three-paths in the network that are closed by fourth tie to establish a "shared four-cycle" structure.

For three-mode networks, `net_congruency` calculates the proportion of three-paths spanning two two-mode networks that are closed by a fourth tie to establish a "congruent four-cycle" structure.

`net_by_reciprocity()` takes a `variant`: either `"default"`, the share of ties that are reciprocated, or `"ratio"`, the share of dyads that are mutual rather than asymmetric. See `?igraph::reciprocity`.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Cyclicality

Where transitivity asks how often a two-path $i \to j \to k$ is closed by a tie $i \to k$, cyclicality asks how often it is closed in the other direction, by $k \to i$:

$$C = \frac{|\{i \to j \to k \to i\}|}{|\{i \to j \to k\}|}$$

The two capture different social logics. Transitivity is the signature of hierarchy and of "a friend of a friend is a friend", while cyclicality is the signature of generalised exchange, where resources circulate around a loop rather than flowing consistently in one direction.

A two-mode network contains no cycle of odd length, so it scores 0 here, just as it does for transitivity. Use `net_by_equivalency()` for closure in a two-mode network, which counts four-cycles instead.

In an undirected network every two-path closed in one direction is also closed in the other, so cyclicality and transitivity coincide.

Equivalency

The `net_by_equivalency()` function calculates the Robins and Alexander (2004) clustering coefficient for two-mode networks. The coefficient is a proportion of three-paths, and so is defined on binary data; weighted networks are dichotomised before it is calculated.

References

On cyclicality and generalised exchange:

Bearman, Peter. 1997. "Generalized Exchange". *American Journal of Sociology* 102(5): 1383-1415. doi:10.1086/231087

On equivalency or four-cycles:

Robins, Garry L, and Malcolm Alexander. 2004. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory* 10(1): 69–94. doi:10.1023/B:CMOT.0000032580.12184.c0.

On congruency:

Knoke, David, Mario Diani, James Hollway, and Dimitris C Christopoulos. 2021. *Multimodal Political Networks*. Cambridge University Press. Cambridge University Press. doi:10.1017/9781108985000

See Also

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
net_by_reciprocity(ison_southern_women)
net_by_transitivity(ison_adolescents)
net_by_cyclicality(ison_networkers)
net_by_equivalency(ison_southern_women)
```

`measure_closure_node` *Measuring node closure*

Description

These functions offer methods for summarising the closure in configurations in one- and two-mode networks:

- `node_by_reciprocity()` measures nodes' reciprocity.
- `node_by_transitivity()` measures nodes' transitivity.
- `node_by_equivalency()` measures nodes' equivalence or reinforcement in a (usually two-mode) network.

Usage

```
node_by_reciprocity(.data)

node_by_transitivity(.data)

node_by_equivalency(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Details

For one-mode networks, shallow wrappers of igraph versions exist via `node_by_reciprocity` and `node_by_transitivity`.

For two-mode networks, `node_by_equivalency` calculates the proportion of three-paths in the network that are closed by fourth tie to establish a "shared four-cycle" structure.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Node reciprocity

A node's reciprocity is the proportion of its ties that are returned. Where a network is undirected, including where it is two-mode, there is no direction for a tie to be returned along, so every node scores 1. This is what `net_by_reciprocity()` reports for such a network too.

Node transitivity

A node's transitivity is the proportion of its neighbours that are themselves connected, which is also known as the *local clustering coefficient* of the node.

References

On the local clustering coefficient:

Watts, Duncan J., and Steven H. Strogatz. 1998. "Collective dynamics of 'small-world' networks". *Nature* 393(6684): 440-442. doi:10.1038/30918

Holland, Paul W., and Samuel Leinhardt. 1971. "Transitivity in structural models of small groups". *Comparative Group Studies* 2(2): 107-124. doi:10.1177/104649647100200201

See Also

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_reciprocity(ison_networkers)
node_by_transitivity(ison_adolescents)
```

---

measure_cohesion *Measures of network cohesion*

Description

These functions return values or vectors relating to how cohesive a network is:

- net_by_density() measures the ratio of ties to the number of possible ties.
- net_by_compactness() measures the average closeness of all pairs of nodes in the network.
- net_by_components() measures the number of components in the network, either strongly or weakly connected.
- net_by_independence() measures the independence number, or size of the largest independent set in the network.

Usage

```
net_by_density(.data)

net_by_compactness(.data)

net_by_components(.data, connectivity = c("strong", "weak"))

net_by_independence(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`connectivity`	Character string, "weak" treats a directed network's components as if the network were undirected, and "strong" requires ties in both directions between members. This is ignored for undirected networks, where the two notions coincide. Note that the default differs by function: functions that assert or count connectedness default to "strong", while functions that scope or split a network into components default to "weak".

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Signed networks

`net_by_compactness()` measures distance, and a negative tie is hostility rather than a channel along which cohesion travels. Where the network is signed, it therefore considers only the positive ties. Use `manynet::to_unsigned()` first to control this yourself. The other measures in this topic do not depend on distance, and so use every tie whatever its sign.

Multilevel networks

A multilevel network reports itself as two-mode, but holds ties within a mode as well as between them, so it cannot be projected onto one mode. `net_by_independence()` therefore measures a multilevel network whole, which is the quantity wanted in any case. The projection remains for genuine two-mode networks, where no two nodes of one mode are ever tied and the unprojected answer would be trivially the larger mode.

Compactness

Compactness is the average of the reciprocal distances between all pairs of nodes:

$$C = \frac{\sum_{i \neq j} \frac{1}{d(i,j)}}{N(N-1)}$$

where unreachable pairs contribute 0. Its complement, $1 - C$, is sometimes called breadth.

Compactness is more discriminating than `net_by_connectedness()`, which counts only whether pairs are reachable at all. Two networks in which every node can reach every other are equally connected, but the one in which they do so in fewer steps is more compact. A complete network scores 1, and an empty network 0. It is the network-level counterpart of `node_by_harmonic()`, such that `net_by_compactness(ison_adolescents) == mean(node_by_harmonic(ison_adolescents, normalized = TRUE, cutoff = -1))`.

Note that this quantity is known in the physics literature as the *global efficiency* of a network (Latora and Marchiori 2001). It is named compactness here for the social network analytic tradition, partly to avoid confusion with the unrelated `net_by_efficiency()` (Krackhardt) and `node_by_efficiency()` (Burt).

References

On compactness:

Borgatti, Stephen P., Martin G. Everett, Jeffrey C. Johnson, and Filip Agneessens. 2022. *Analyzing Social Networks Using R*, chapter 10. London: SAGE.

Latora, Vito, and Massimo Marchiori. 2001. "Efficient Behavior of Small-World Networks". *Physical Review Letters* 87(19): 198701. doi:10.1103/PhysRevLett.87.198701

See Also

Other cohesion: `mark_triangles`, `measure_breadth`, `measure_fragmentation`, `motif_net`, `motif_node`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Examples

```
net_by_density(ison_adolescents)
net_by_density(ison_southern_women)
net_by_compactness(ison_adolescents)
net_by_compactness(ison_southern_women)
net_by_components(fict_thrones)
net_by_components(fict_thrones, connectivity = "weak")
net_by_independence(ison_adolescents)
net_by_independence(fict_actually)
```

---

measure_core *Measuring nodes' coreness*

Description

These functions identify nodes belonging to (some level of) the core of a network:

- `node_by_core()` returns a continuous measure of how closely each node resembles a typical core node.
- `node_by_kcoreness()` assigns nodes to their level of k-coreness.

Usage

```
node_by_kcoreness(.data)

node_by_core(.data, coreness = NULL, direction = c("all", "out", "in"))
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`coreness`	Which method to use to calculate nodes' coreness. One of "correlation", "rich", "transition", or "hub"; see method_coreness for what each does. By default NULL, which uses "rich" for a weighted, directed, or two-mode network, since it is the only method that reads those properties directly, and "correlation" otherwise.
`direction`	One of "all" (the default), "out", or "in". For a directed network, "out" scores nodes on the ties they send and "in" on the ties they receive. Ignored for undirected and two-mode networks.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

k-coreness

k-coreness captures the maximal subgraphs in which each vertex has at least degree k, where k is also the order of the subgraph. As described in `igraph::coreness`, a node's coreness is k if it belongs to the k-core but not to the (k+1)-core.

Coreness

Where `node_is_core()` forces a yes or no answer, `node_by_core()` grades how core-like each node is on a scale from 0 to 1. The two agree on which method to use and read the same `coreness` and `direction` arguments, so the mark is always the cut of the measure returned here.

Each method uses as much of the network as it can. The rich-core and hub methods read tie weights and tie direction directly. The correlation and transition methods compare the network against a symmetric ideal, so they symmetrise a directed network and say that they have done so. To keep a method from using a property, transform the network first with e.g. `manynet::to_undirected()` or `manynet::to_unweighted()`.

This function was called `node_by_coreness()` prior to version 1.0.0. It is now named for the property, as `node_is_core()` and `node_in_core()` are, which also frees "coreness" from meaning two different things: the continuous score here, and the peeling depth of `node_by_kcoreness()`.

References

On k-coreness:

Seidman, Stephen B. 1983. "Network structure and minimum degree". *Social Networks*, 5(3), 269-287. doi:10.1016/03788733(83)90028X

Batagelj, Vladimir, and Matjaz Zaversnik. 2003. "An O(m) algorithm for cores decomposition of networks". *arXiv preprint* cs/0310049. doi:10.48550/arXiv.cs/0310049

See Also

Other core-periphery: `mark_core`, `member_core`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_features`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
node_by_kcoreness(ison_adolescents)
node_by_core(ison_adolescents)
node_by_core(ison_networkers, direction = "out")
```

`measure_diffusion_infection`

Measures of network infection

Description

These functions allow measurement of various features of a diffusion process at the network level:

- `net_by_infection_complete()` measures the number of time steps until (the first instance of) complete infection. For diffusions that are not observed to complete, this function returns the value of `Inf` (infinity). This makes sure that at least ordinality is respected.
- `net_by_infection_total()` measures the proportion or total number of nodes that are infected/activated at some time by the end of the diffusion process. This includes nodes that subsequently recover. Where reinfection is possible, the proportion may be higher than 1.
- `net_by_infection_peak()` measures the number of time steps until the highest infection rate is observed.

Usage

```
net_by_infection_complete(.data)

net_by_infection_total(.data, normalized = TRUE)

net_by_infection_peak(.data)
```

Arguments

.data	Network data with nodal changes, as created by play_diffusion(), or a valid network diffusion model, as created by as_diffusion().
normalized	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A network_measure numeric score.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

See Also

Other diffusion: mark_diff, measure_diffusion_net, measure_diffusion_node, member_diffusion, motif_exposure, motif_hazard

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
smeg <- generate_smallworld(15, 0.025)
smeg_diff <- play_diffusion(smeg)
net_by_infection_complete(smeg_diff)
net_by_infection_total(smeg_diff)
net_by_infection_peak(smeg_diff)
```

`measure_diffusion_net` *Measures of network diffusion*

Description

These functions allow measurement of various features of a diffusion process at the network level:

- `net_by_transmissibility()` measures the average transmissibility observed in a diffusion simulation, or the number of new infections over the number of susceptible nodes.
- `net_by_recovery()` measures the average number of time steps nodes remain infected once they become infected.
- `net_by_reproduction()` measures the observed reproductive number in a diffusion simulation as the network's transmissibility over the network's average infection length.
- `net_by_immunity()` measures the proportion of nodes that would need to be protected through vaccination, isolation, or recovery for herd immunity to be reached.

Usage

```
net_by_transmissibility(.data)

net_by_recovery(.data, censor = TRUE)

net_by_reproduction(.data)

net_by_immunity(.data, normalized = TRUE)
```

Arguments

`.data`	Network data with nodal changes, as created by `play_diffusion()`, or a valid network diffusion model, as created by `as_diffusion()`.
`censor`	Where some nodes have not yet recovered by the end of the simulation, right censored values can be replaced by the number of steps. By default TRUE. Note that this will likely still underestimate recovery.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Transmissibility

`net_transmissibility()` measures how many directly susceptible nodes each infected node will infect in each time period, on average. That is:

$$T = \frac{1}{n} \sum_{j=1}^{n} \frac{i_j}{s_j}$$

where i is the number of new infections in each time period, $j \in n$, and s is the number of nodes that could have been infected in that time period (note that $s \neq S$, or the number of nodes that are susceptible in the population). T can be interpreted as the proportion of susceptible nodes that are infected at each time period.

Recovery time

`net_recovery()` measures the average number of time steps that nodes in a network remain infected. Note that in a diffusion model without recovery, average infection length will be infinite. This will also be the case where there is right censoring. The longer nodes remain infected, the longer they can infect others.

Reproduction number

`net_reproduction()` measures a given diffusion's reproductive number. Here it is calculated as:

$$R = \min\left(\frac{T}{1/L}, \bar{k}\right)$$

where T is the observed transmissibility in a diffusion and L is the observed recovery length in a diffusion. Since L can be infinite where there is no recovery or there is right censoring, and since network structure places an upper limit on how many nodes each node may further infect (their degree), this function returns the minimum of R_0 and the network's average degree.

Interpretation of the reproduction number is oriented around R = 1. Where $R > 1$, the 'disease' will 'infect' more and more nodes in the network. Where $R < 1$, the 'disease' will not sustain itself and eventually die out. Where $R = 1$, the 'disease' will continue as endemic, if conditions allow.

Herd immunity

`net_immunity()` estimates the proportion of a network that need to be protected from infection for herd immunity to be achieved. This is known as the Herd Immunity Threshold or HIT:

$$1 - \frac{1}{R}$$

where R is the reproduction number from `net_reproduction()`. The HIT indicates the threshold at which the reduction of susceptible members of the network means that infections will no longer keep increasing. Note that there may still be more infections after this threshold has been reached, but there should be fewer and fewer. These excess infections are called the *overshoot*. This function does *not* take into account the structure of the network, instead using the average degree.

Interpretation is quite straightforward. A HIT or immunity score of 0.75 would mean that 75% of the nodes in the network would need to be vaccinated or otherwise protected to achieve herd

immunity. To identify how many nodes this would be, multiply this proportion with the number of nodes in the network.

Where $R < 1$ the diffusion is already sub-critical and dies out of its own accord, so no one needs protecting and the threshold is reported as 0. The formula would otherwise return a negative proportion, which has no interpretation.

References

On epidemiological models:

Kermack, William O., and Anderson Gray McKendrick. 1927. "A contribution to the mathematical theory of epidemics". *Proc. R. Soc. London A* 115: 700-721. doi:10.1098/rspa.1927.0118

On the basic reproduction number:

Diekmann, Odo, Hans J.A.P. Heesterbeek, and Hans J.A.J. Metz. 1990. "On the definition and the computation of the basic reproduction ratio R0 in models for infectious diseases in heterogeneous populations". *Journal of Mathematical Biology*, 28(4): 365–82. doi:10.1007/BF00178324

Kenah, Eben, and James M. Robins. 2007. "Second look at the spread of epidemics on networks". *Physical Review E*, 76(3 Pt 2): 036113. doi:10.1103/PhysRevE.76.036113

On herd immunity:

Garnett, G.P. 2005. "Role of herd immunity in determining the effect of vaccines against sexually transmitted disease". *The Journal of Infectious Diseases*, 191(1): S97-106. doi:10.1086/425271

See Also

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other diffusion: mark_diff, measure_diffusion_infection, measure_diffusion_node, member_diffusion, motif_exposure, motif_hazard

Examples

```
smeg <- generate_smallworld(15, 0.025)
smeg_diff <- play_diffusion(smeg, recovery = 0.2)
plot(smeg_diff)
# To calculate the average transmissibility for a given diffusion model
net_by_transmissibility(smeg_diff)
# To calculate the average infection length for a given diffusion model
net_by_recovery(smeg_diff)
# To calculate the reproduction number for a given diffusion model
net_by_reproduction(smeg_diff)
# Calculating the proportion required to achieve herd immunity
net_by_immunity(smeg_diff)
# To find the number of nodes to be vaccinated
net_by_immunity(smeg_diff, normalized = FALSE)
```

`measure_diffusion_node`

Measures of nodes in a diffusion

Description

These functions allow measurement of various features of a diffusion process:

- `node_by_adopt_time()`: Measures the number of time steps until nodes adopt/become infected
- `node_by_adopt_threshold()`: Measures nodes' thresholds from the amount of exposure they had when they became infected
- `node_by_adopt_recovery()`: Measures the average length nodes that become infected remain infected in a compartmental model with recovery
- `node_by_adopt_exposure()`: Measures how many exposures nodes have to a given mark

Usage

```
node_by_adopt_time(.data)

node_by_adopt_threshold(.data, normalized = TRUE, lag = 1)

node_by_adopt_recovery(.data)

node_by_adopt_exposure(.data, mark, time = 0)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`normalized`	Logical scalar, whether scores are normalized. Different denominators may be used depending on the measure, whether the object is one-mode or two-mode, and other arguments. By default TRUE.
`lag`	The number of time steps back upon which the thresholds are inferred.
`mark`	A valid 'node_mark' object or logical vector (TRUE/FALSE) of length equal to the number of nodes in the network.
`time`	A time point until which infections/adoptions should be identified. By default `time = 0`.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Adoption time

`node_by_adopt_time()` measures the time units it took until each node became infected. Note that an adoption time of 0 indicates that this was a seed node.

Thresholds

`node_by_adopt_threshold()` infers nodes' thresholds based on how much exposure they had when they were infected. This inference is of course imperfect, especially where there is a sudden increase in exposure, but it can be used heuristically. In a threshold model, nodes activate when $\sum_{j:\text{active}} w_{ji} \geq \theta_i$, where w is some (potentially weighted) matrix, j are some already activated nodes, and $theta$ is some pre-defined threshold value. Where a fractional threshold is used, the equation is $\frac{\sum_{j:\text{active}} w_{ji}}{\sum_j w_{ji}} \geq \theta_i$. That is, $theta$ is now a proportion, and works regardless of whether w is weighted or not.

Recovery

`node_by_adopt_recovery()` measures the average length of time that nodes that become infected remain infected in a compartmental model with recovery. Infections that are not concluded by the end of the study period are calculated as infinite.

Exposure

`node_exposure()` calculates the number of infected/adopting nodes to which each susceptible node is exposed. It usually expects network data and an index or mark (TRUE/FALSE) vector of those nodes which are currently infected, but if a diff_model is supplied instead it will return nodes exposure at $t = 0$.

References

On diffusion measures:

Valente, Tom W. 1995. *Network models of the diffusion of innovations* (2nd ed.). Cresskill N.J.: Hampton Press.

See Also

Other diffusion: `mark_diff`, `measure_diffusion_infection`, `measure_diffusion_net`, `member_diffusion`, `motif_exposure`, `motif_hazard`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`,

measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
smeg <- generate_smallworld(15, 0.025)
smeg_diff <- play_diffusion(smeg, recovery = 0.2)
plot(smeg_diff)
# To measure when nodes adopted a diffusion/were infected
(times <- node_by_adopt_time(smeg_diff))
# To infer nodes' thresholds
node_by_adopt_threshold(smeg_diff)
# To measure how long each node remains infected for
node_by_adopt_recovery(smeg_diff)
# To measure how much exposure nodes have to a given mark
node_by_adopt_exposure(smeg, mark = c(1,3))
node_by_adopt_exposure(smeg_diff)
```

---

measure_diverse_net *Measures of network diversity*

Description

These functions offer ways to measure the heterogeneity of an attribute across a network, within groups of a network, or the distribution of ties across this attribute:

- `net_by_richness()` measures the number of unique categories in a network attribute.
- `net_by_diversity()` measures the heterogeneity of ties across a network.

Usage

```
net_by_richness(.data, attribute)

net_by_diversity(
  .data,
  attribute,
  diversity = c("blau", "teachman", "variation", "gini")
)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`attribute`	Name of a nodal attribute, mark, measure, or membership vector.
`diversity`	Which method to use for `*_diversity()`. Either "blau" (Blau's index) or "teachman" (Teachman's index) for categorical attributes, or "variation" (coefficient of variation) or "gini" (Gini coefficient) for numeric attributes. Default is "blau". If an incompatible method is chosen for the attribute type, a suitable alternative will be used instead with a message.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Richness

Richness is a simple count of the number of different categories present for a given attribute.

Diversity

Blau's index (1977) uses a formula known also in other disciplines by other names (Gini-Simpson Index, Gini impurity, Gini's diversity index, Gibbs-Martin index, and probability of interspecific encounter (PIE)):

$$1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the proportion of group members in ith category and k is the number of categories for an attribute of interest. This index can be interpreted as the probability that two members randomly selected from a group would be from different categories. This index finds its minimum value (0) when there is no variety, i.e. when all individuals are classified in the same category. The maximum value depends on the number of categories and whether nodes can be evenly distributed across categories.

Teachman's index (1980) is based on information theory and is calculated as:

$$-\sum_{i=1}^{k} p_i \log(p_i)$$

where p_i is the proportion of group members in ith category and k is the number of categories for an attribute of interest. This index finds its minimum value (0) when there is no variety, i.e. when all individuals are classified in the same category. The maximum value depends on the number of categories and whether nodes can be evenly distributed across categories. It thus shares similar properties to Blau's index, but includes also a notion of richness that tends to give more weight to rare categories and thus tends to highlight imbalances more.

The coefficient of variation (CV) is a standardised measure of dispersion of a probability distribution or frequency distribution. It is defined as the ratio of the standard deviation σ to the mean μ:

$$CV = \frac{\sigma}{\mu}$$

It is often expressed as a percentage. The CV is useful because the standard deviation of data must always be understood in the context of the mean of the data. The CV is particularly useful when comparing the degree of variation from one data series to another, even if the means are drastically different from each other.

The Gini coefficient is a measure of statistical dispersion that is intended to represent the income or wealth distribution of a nation's residents, and is commonly used as a measure of inequality. It is defined as a ratio with values between 0 and 1, where 0 corresponds with perfect equality (everyone has the same income) and 1 corresponds with perfect inequality (one person has all the income, and everyone else has zero income). The Gini coefficient can be calculated from the Lorenz curve, which plots the proportion of the total income of the population that is cumulatively earned by the bottom x% of the population. The Gini coefficient is defined as the area between the line of equality and the Lorenz curve, divided by the total area under the line of equality.

References

On richness:

Magurran, Anne E. 1988. *Ecological Diversity and Its Measurement*. Princeton: Princeton University Press. doi:10.1007/9789401573580

On diversity:

Blau, Peter M. 1977. *Inequality and heterogeneity*. New York: Free Press.

Teachman, Jay D. 1980. Analysis of population diversity: Measures of qualitative variation. *Sociological Methods & Research*, 8:341-362. doi:10.1177/004912418000800305

Page, Scott E. 2010. *Diversity and Complexity*. Princeton: Princeton University Press. doi:10.1515/9781400835140

See Also

Other diversity: measure_assort_net, measure_assort_node, measure_diverse_node, motif_composition, motif_homophily

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
net_by_richness(ison_networkers)
marvel_friends <- to_unsigned(to_uniplex(fict_marvel, "relationship"), "positive")
net_by_diversity(marvel_friends, "Gender")
net_by_diversity(marvel_friends, "Appearances")
```

`measure_diverse_node` *Measures of nodes diversity*

Description

These functions offer ways to measure the heterogeneity of an attribute across a network, within groups of a network, or the distribution of ties across this attribute:

- `node_by_richness()` measures the number of unique categories of an attribute to which each node is connected.
- `node_by_diversity()` measures the heterogeneity of each node's local neighbourhood.

Usage

```
node_by_richness(.data, attribute)

node_by_diversity(
  .data,
  attribute,
  diversity = c("blau", "teachman", "variation", "gini")
)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`attribute`	Name of a nodal attribute, mark, measure, or membership vector.
`diversity`	Which method to use for `*_diversity()`. Either "blau" (Blau's index) or "teachman" (Teachman's index) for categorical attributes, or "variation" (coefficient of variation) or "gini" (Gini coefficient) for numeric attributes. Default is "blau". If an incompatible method is chosen for the attribute type, a suitable alternative will be used instead with a message.

Value

A `node_measure` numeric vector the length of the nodes in the network, providing the scores for each node. If the network is labelled, then the scores will be labelled with the nodes' names.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

See Also

Other diversity: measure_assort_net, measure_assort_node, measure_diverse_net, motif_composition, motif_homophily

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_features, measure_fit, measure_fragmentation, measure_hierarchy, measure_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_by_richness(ison_networkers, "Discipline")
marvel_friends <- to_unsigned(to_uniplex(fict_marvel, "relationship"), "positive")
node_by_diversity(marvel_friends, "Gender")
node_by_diversity(marvel_friends, "Attractive")
```

`measure_features` *Measuring network topological features*

Description

These functions measure topological features that are intrinsic to a network, in the sense that they require nothing of the user beyond the network itself:

- `net_by_richclub()` measures the rich-club coefficient of a network.
- `net_by_smallworld()` measures the small-world coefficient for one- or two-mode networks. Small-world networks can be highly clustered and yet have short path lengths.
- `net_by_scalefree()` measures the exponent of a fitted power-law distribution. An exponent between 2 and 3 usually indicates a power-law distribution.
- `net_by_balance()` measures the structural balance index on the proportion of balanced triangles, ranging between `0` if all triangles are imbalanced and `1` if all triangles are balanced.
- `net_by_bipartivity()` measures how close a network is to being bipartite, that is, to dividing into two sets with ties only between them.

Usage

```
net_by_richclub(.data)

net_by_smallworld(
  .data,
  variant = c("omega", "sigma", "SWI"),
  times = 100,
  method = NULL
)

net_by_scalefree(.data)

net_by_bipartivity(.data)

net_by_balance(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`variant`	Character string naming which variant of the measure to compute, where more than one definition of the same quantity is in use. The variant chosen is reported when the result is printed.
`times`	Integer of number of simulations.
`method`	Deprecated. The former spelling of `variant`. Still accepted, but warns; please use `variant` instead.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Small-world variants

For `net_by_smallworld()` there are three small-world measures implemented:

- "sigma" is the original equation from Watts and Strogatz (1998),

$$\frac{\frac{C}{C_r}}{\frac{L}{L_r}}$$

, where C and L are the observed clustering coefficient and path length, respectively, and C_r and L_r are the averages obtained from random networks of the same dimensions and density. A $\sigma > 1$ is considered to be small-world, but this measure is highly sensitive to network size.

- "omega" (the default) is an update from Telesford et al. (2011),

$$\frac{L_r}{L} - \frac{C}{C_l}$$

, where C_l is the clustering coefficient for a lattice graph with the same dimensions. ω ranges between -1 and 1, where values close to 0 are as close to a small-world as possible; negative values indicate a lattice-like network, and positive values a more random one.

- "SWI" is an alternative proposed by Neal (2017),

$$\frac{L - L_l}{L_r - L_l} \times \frac{C - C_r}{C_l - C_r}$$

, where L_l is the average path length for a lattice graph with the same dimensions. SWI ranges between 0 and 1, where 1 is as close to a small-world as possible, though there may not be a network for which $SWI = 1$.

Bipartivity

A network is bipartite when its nodes divide into two sets with ties only running between them and never within, which is exactly the condition that it contains no closed walk of odd length. Bipartivity therefore measures how close a network comes to that condition, as the share of its closed walks that are of even length:

$$b(G) = \frac{\sum_i C_{even}(i)}{\sum_i C_{all}(i)}$$

A genuinely two-mode network scores exactly 1, and the more odd-length structure a network carries — triangles above all — the further it falls below 1. Note that this asks whether a network *could* be split in two, not whether it has been: it is defined on a one-mode network, whereas `manynet::is_twomode()` reports whether nodes are already partitioned into two modes. The node-level counterpart is `node_by_subgraph()` with `walks = "odd"` or `"even"`.

Source

`{signnet}` by David Schoch

References

On the rich-club coefficient:

Zhou, Shi, and Raul J. Mondragon. 2004. "The Rich-Club Phenomenon in the Internet Topology". *IEEE Communications Letters*, 8(3): 180-182. doi:10.1109/lcomm.2004.823426

On small-worldliness:

Watts, Duncan J., and Steven H. Strogatz. 1998. “Collective Dynamics of ‘Small-World’ Networks”. *Nature* 393(6684):440–42. doi:10.1038/30918

Telesford QK, Joyce KE, Hayasaka S, Burdette JH, Laurienti PJ. 2011. "The ubiquity of small-world networks". *Brain Connectivity* 1(5): 367–75. doi:10.1089/brain.2011.0038

Neal, Zachary P. 2017. "How small is it? Comparing indices of small worldliness". *Network Science*. 5 (1): 30–44. doi:10.1017/nws.2017.5

On scale-free networks:

Barabasi, Albert-Laszlo, and Reka Albert. 1999. "Emergence of scaling in random networks", *Science*, 286(5439): 509-512. doi:10.1126/science.286.5439.509

Clauset, Aaron, Cosma Rohilla Shalizi, and Mark E.J. Newman. 2009. "Power-law distributions in empirical data", *SIAM Review*, 51(4): 661-703. doi:10.1137/070710111

Stumpf, Michael P.H., and Mason Porter. 2012. "Critical truths about power laws", *Science*, 335(6069): 665-666. doi:10.1126/science.1216142

Holme, Petter. 2019. "Rare and everywhere: Perspectives on scale-free networks", *Nature Communications*, 10(1): 1016. doi:10.1038/s41467019090388

On bipartivity:

Estrada, Ernesto, and Juan A. Rodríguez-Velázquez. 2005. "Spectral measures of bipartivity in complex networks". *Physical Review E* 72(4): 046105. doi:10.1103/PhysRevE.72.046105

On balance theory:

Heider, Fritz. 1946. "Attitudes and cognitive organization". *The Journal of Psychology*, 21: 107-112. doi:10.1080/00223980.1946.9917275

Cartwright, D., and Frank Harary. 1956. "Structural balance: A generalization of Heider's theory". *Psychological Review*, 63(5): 277-293. doi:10.1037/h0046049

See Also

`net_by_transitivity()` and `net_by_equivalency()` for how clustering is calculated

Other features: `measure_fit`

Other measures: `measure_assort_net`, `measure_assort_node`, `measure_breadth`, `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_central_tie_between`, `measure_central_tie_close`, `measure_central_tie_degree`, `measure_central_tie_eigen`, `measure_closure`, `measure_closure_node`, `measure_cohesion`, `measure_core`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `measure_diverse_net`, `measure_diverse_node`, `measure_fit`, `measure_fragmentation`, `measure_hierarchy`, `measure_periods`

Examples

```
net_by_richclub(ison_adolescents)
net_by_smallworld(ison_brandes)
net_by_smallworld(ison_southern_women)
net_by_scalefree(ison_adolescents)
net_by_scalefree(generate_scalefree(50, 1.5))
net_by_scalefree(create_lattice(100))
# A two-mode network is bipartite by construction
net_by_bipartivity(ison_southern_women)
net_by_bipartivity(ison_adolescents)
net_by_balance(to_uniplex(fict_marvel, "relationship"))
```

`measure_fit` *Measuring how well a structure fits a network*

Description

These functions measure how well some proposed structure describes a network. Unlike the intrinsic properties in measure_features, each takes a structure from the user — a core-periphery mark, or a partition of the nodes — and returns how closely the observed network corresponds to it:

- `net_by_core()` measures the correlation between a network and a core-periphery model with the same dimensions.
- `net_by_factions()` measures the correlation between a network and a component model with the same dimensions.
- `net_by_modularity()` measures the modularity of a network based on nodes' membership in defined clusters.
- `net_by_inconsistency()` measures how far a partition's blocks depart from ideal block types.

These are the natural companions to the `node_in_*()` functions, which propose a structure; these say how good that proposal is. Where a partition is expected but none is given, the network is partitioned into two using `node_in_partition()`.

Note that they are not on a common scale, and do not all run in the same direction, so they are not interchangeable:

measure	compares the network against	range	better
`net_by_core()`	a core-periphery model	-1 to 1	higher
`net_by_factions()`	a components model	-1 to 1	higher
`net_by_modularity()`	the partition's communities	-0.5 to 1 (at the default resolution)	higher
`net_by_inconsistency()`	ideal block types	0 upwards	**lower**

Compare partitions using one measure at a time.

Usage

```
net_by_core(
  .data,
  mark = NULL,
  variant = c("correlation", "ident", "ndiff", "diff"),
  coreness = NULL,
  direction = c("all", "out", "in"),
  method = NULL
)

net_by_factions(.data, membership = NULL)
```

```
net_by_modularity(.data, membership = NULL, resolution = 1)

net_by_inconsistency(.data, membership = NULL, blocks = c("nul", "com"))
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
mark	A logical vector indicating which nodes belong to the core.
variant	Character string naming which variant of the measure to compute, where more than one definition of the same quantity is in use. The variant chosen is reported when the result is printed.
coreness	Which method to use to calculate nodes' coreness. One of "correlation", "rich", "transition", or "hub"; see method_coreness for what each does. By default NULL, which uses "rich" for a weighted, directed, or two-mode network, since it is the only method that reads those properties directly, and "correlation" otherwise.
direction	One of "all" (the default), "out", or "in". For a directed network, "out" scores nodes on the ties they send and "in" on the ties they receive. Ignored for undirected and two-mode networks.
method	Deprecated. The former spelling of variant. Still accepted, but warns; please use variant instead.
membership	A character string naming an existing node attribute in the network, or a categorical vector of the same length as the number of nodes in the network where each element indicates the group membership of the corresponding node. While this may often be a vector created using node_in_*() functions, it can be any character vector that assigns nodes to groups or categories.
resolution	A proportion indicating the resolution scale. By default 1, which returns the original definition of modularity. The higher this parameter, the more smaller communities will be privileged. The lower this parameter, the fewer larger communities are likely to be found.
blocks	A character vector of permitted ideal block types, or a list-matrix giving the permitted types for each block position. By default c("nul", "com"), which is structural blockmodelling. See the section below.

Value

A network_measure numeric score.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

Core-periphery fit variants

For `net_by_core()`, which of the following to use to calculate the fit of the core assignment to a core-periphery model. "correlation" calculates the correlation between the empirical network and an ideal typical network, and "ident" calculates the Euclidean distances between the same. "ndiff", however, calculates how distinct the core and periphery groups are based on the difference in coreness scores between the least core-like member of the core and the most core-like member of the periphery. "diff" is similar to "ndiff", but multiplies the raw "ndiff" score by the square root of the size of the core, thus penalising large cores.

Core-Periphery

`net_by_core()` calculates the Pearson correlation between the given network, where the nodes in the core are assigned by some given mark, and an ideal typical core-periphery network with the same number of nodes in the core and the periphery.

Where `mark` is not given, it is calculated with `node_is_core()`, to which the `coreness` and `direction` arguments are passed. For a directed network the fit itself is measured on the symmetrised network, since the ideal it is compared against is symmetric.

Modularity

Modularity measures the difference between the number of ties within each community from the number of ties expected within each community in a random graph with the same degrees. At the default `resolution` it ranges between -0.5 and +1; a higher resolution can push it further below that floor. Modularity scores approaching +1 mean that ties only appear within communities, while negative scores mean that ties appear between communities more often than chance would predict. A score of 0 would mean that ties are half within and half between communities, as one would expect in a random graph.

Modularity faces a difficult problem known as the resolution limit (Fortunato and Barthélemy 2007). This problem appears when optimising modularity, particularly with large networks or depending on the degree of interconnectedness, can miss small clusters that 'hide' inside larger clusters. In the extreme case, this can be where they are only connected to the rest of the network through a single tie. To help manage this problem, a `resolution` parameter is added. Please see the argument definition for more details.

Blockmodelling

A blockmodel proposes that a partition reduces a network to a small number of positions, so that every block — the ties running from one position to another — is of some simple ideal type. `net_by_inconsistency()` measures how far the network departs from that proposal, by counting the ties that would have to be added or removed to make every block ideal, normalized by the number of cells. **Lower is better**: 0 means the partition fits perfectly.

This is a *distance* from an ideal image rather than a measure of fit — hence the name, and hence its running the opposite way to the rest of this page. Three consequences are worth knowing:

- **Its complement is not a proportion.** The criterion mixes units: `nul` and `com` count cells, while `reg` counts empty rows and columns, and all are divided by the cell count. So do not read $1 - x$ as the share of the network that the blockmodel gets right.

- **It is not bounded above by 1.** That holds only for cell-counting vocabularies such as `c("nul", "com")`. With `reg` permitted it can exceed 1 — on `ison_adolescents`, `blocks = "reg"` over singleton positions reaches about 1.57.
- **The vocabularies behave very differently at fine partitions.** Giving every node its own position scores 0 under `c("nul", "com")`, since each block is then a single cell and trivially ideal, but scores its *worst* under `"reg"`, since each block then has an empty row and column.

For a correlation-scaled, higher-is-better reading of the common structural case, see `net_by_factions()`. The two are related but not equivalent: `net_by_factions()` fixes the image — complete on the diagonal, null off it — whereas `net_by_inconsistency(blocks = c("nul", "com"))` lets each block take whichever of the two ideals fits it better, and so is more permissive.

The ideal types are:

`nul` a null block, containing no ties.

`com` a complete block, containing every possible tie.

`reg` a regular block, in which every row and every column has at least one tie, though not necessarily all of them.

`rdo`, `cdo` a row- or column-dominant block, containing at least one complete row or column.

`dnc` "do not care": a block left unconstrained.

`blocks` is a *vocabulary* rather than an assignment: each block is scored at the lowest inconsistency of any permitted type, and the results summed. Any subset may be given, and the two conventional choices are `c("nul", "com")` for structural equivalence and `c("nul", "reg")` for regular equivalence.

Note that permitting more types can only lower the criterion, since each block gains more ways to be satisfied. The size of the vocabulary is therefore itself a modelling choice, and criterion values are comparable across partitions only when the same vocabulary is used for each.

For fully generalized blockmodelling, pass a `g` by `g` list-matrix naming the types permitted at each position separately, e.g. `reg` on the diagonal and `nul` off it for a "cohesive positions" model.

References

On core-periphery:

Borgatti, Stephen P., and Martin G. Everett. 2000. "Models of Core/Periphery Structures." *Social Networks* 21(4):375–95. doi:10.1016/S03788733(99)000192

On modularity:

Newman, Mark E.J. 2006. "Modularity and community structure in networks", *Proceedings of the National Academy of Sciences* 103(23): 8577-8696. doi:10.1073/pnas.0601602103

Murata, Tsuyoshi. 2010. "Modularity for Bipartite Networks". In: Memon, N., Xu, J., Hicks, D., Chen, H. (eds) *Data Mining for Social Network Data. Annals of Information Systems*, Vol 12. Springer, Boston, MA. doi:10.1007/9781441962874_7

On generalized blockmodelling:

Doreian, Patrick, Vladimir Batagelj, and Anuska Ferligoj. 2005. *Generalized Blockmodeling*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511584176

See Also

Other features: measure_features

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fragmentation, measure_hierarchy, measure_periods

Examples

```
net_by_core(ison_adolescents)
net_by_core(ison_southern_women)
  net_by_factions(ison_southern_women)
net_by_modularity(ison_adolescents,
  node_in_partition(ison_adolescents))
net_by_modularity(ison_southern_women,
  node_in_partition(ison_southern_women))
net_by_inconsistency(ison_hightech, node_in_regular(ison_hightech))
# a regular-equivalence vocabulary instead of a structural one
net_by_inconsistency(ison_hightech, node_in_structural(ison_hightech),
       blocks = c("nul", "reg"))
```

---

measure_fragmentation *Measures of network fragmentation*

Description

These functions return values relating to how connected a network is and the number of nodes or edges to remove that would increase fragmentation.

- net_by_cohesion() measures the minimum number of nodes to remove from the network needed to increase the number of components.
- net_by_toughness() measures the number of nodes that would need to be removed from a network to increase its number of components.
- net_by_adhesion() measures the minimum number of ties to remove from the network needed to increase the number of components.
- net_by_strength() measures the number of ties that would need to be removed from a network to increase its number of components.

Usage

```
net_by_cohesion(.data)

net_by_adhesion(.data)
```

```
net_by_strength(.data)

net_by_toughness(.data)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().

Value

A network_measure numeric score.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

References

On cohesion:

White, Douglas R and Frank Harary. 2001. "The Cohesiveness of Blocks In Social Networks: Node Connectivity and Conditional Density." *Sociological Methodology* 31(1): 305-59. doi:10.1111/00811750.00098

See Also

Other cohesion: mark_triangles, measure_breadth, measure_cohesion, motif_net, motif_node

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_hierarchy, measure_periods

Examples

```
net_by_cohesion(fict_marvel)
net_by_cohesion(to_giant(fict_marvel))
net_by_adhesion(fict_marvel)
net_by_adhesion(to_giant(fict_marvel))
net_by_strength(ison_adolescents)
net_by_toughness(ison_adolescents)
```

`measure_hierarchy` *Measures of hierarchy*

Description

These functions, together with `net_reciprocity()`, are used jointly to measure how hierarchical a network is:

- `net_by_connectedness()` measures the proportion of dyads in the network that are reachable to one another, or the degree to which network is a single component.
- `net_by_efficiency()` measures the Krackhardt efficiency score.
- `net_by_upperbound()` measures the Krackhardt (least) upper bound score.

Usage

```
net_by_connectedness(.data)

net_by_efficiency(.data)

net_by_upperbound(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `network_measure` numeric score.

The object also carries the `measure` it computed, the `range` its values can fall within, and whether and how those values were `normalized`. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the `variant` it used. All can be retrieved with `attr()`.

Efficiency

A perfect hierarchy is a tree: every node but the root has exactly one superior, and there are no ties to spare. Krackhardt's efficiency asks how close a network comes to that, by counting the ties it carries in excess of the minimum needed to hold its components together, as a proportion of the most excess ties it could possibly carry:

$$E = 1 - \frac{|E| - \sum_i (N_i - 1)}{\sum_i (M_i - (N_i - 1))}$$

where N_i is the size of weak component i and M_i the number of ties possible within it. A tree or forest scores 1, and a complete network 0.

References

On hierarchy:

Krackhardt, David. 1994. Graph theoretical dimensions of informal organizations. In Carley and Prietula (eds) *Computational Organizational Theory*, Hillsdale, NJ: Lawrence Erlbaum Associates. Pp. 89-111.

Everett, Martin, and David Krackhardt. 2012. "A second look at Krackhardt's graph theoretical dimensions of informal organizations." *Social Networks*, 34: 159-163. doi:10.1016/j.socnet.2011.10.006

See Also

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_periods

Other hierarchy: motif_hierarchy

Examples

```
net_by_connectedness(ison_networkers)
1 - net_by_reciprocity(ison_networkers)
net_by_efficiency(ison_networkers)
net_by_upperbound(ison_networkers)
```

---

measure_periods *Measures of network change*

Description

`net_by_waves()` measures the number of waves in longitudinal network data.

Usage

```
net_by_waves(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A network_measure numeric score.

The object also carries the measure it computed, the range its values can fall within, and whether and how those values were normalized. These are shown as a one-line header when the object is printed. Where a measure offers a choice between several ways of counting the same thing, it also carries the variant it used. All can be retrieved with attr().

See Also

Other change: motif_periods

Other measures: measure_assort_net, measure_assort_node, measure_breadth, measure_broker_node, measure_broker_tie, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_central_tie_between, measure_central_tie_close, measure_central_tie_degree, measure_central_tie_eigen, measure_closure, measure_closure_node, measure_cohesion, measure_core, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, measure_diverse_net, measure_diverse_node, measure_features, measure_fit, measure_fragmentation, measure_hierarchy

Examples

```
net_by_waves(ison_monks)
```

---

member_brokerage *Memberships in brokerage positions*

Description

node_in_brokering() returns nodes membership as a powerhouse, connector, linchpin, or sideliner according to Hamilton et al. (2020).

Usage

```
node_in_brokering(.data, membership)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
membership	A character string naming an existing node attribute in the network, or a categorical vector of the same length as the number of nodes in the network where each element indicates the group membership of the corresponding node. While this may often be a vector created using node_in_*() functions, it can be any character vector that assigns nodes to groups or categories.

Value

A node_member character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

References

On brokerage activity and exclusivity:

Hamilton, Matthew, Jacob Hileman, and Orjan Bodin. 2020. "Evaluating heterogeneous brokerage: New conceptual and methodological approaches and their application to multi-level environmental governance networks" *Social Networks* 61: 1-10. doi:10.1016/j.socnet.2019.08.002

See Also

Other brokerage: measure_broker_node, measure_broker_tie, measure_brokerage, motif_brokerage_net, motif_brokerage_node

Other memberships: member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_in_brokering(ison_networkers, "Discipline")
```

---

member_cliques *Memberships in maximally diverse cliques*

Description

These functions create a vector of nodes' memberships in cliques:

- node_in_roulette() assigns nodes to maximally diverse groups.

Usage

```
node_in_roulette(.data, groups, group_size, times = NULL, num_groups = NULL)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`groups`	An integer indicating the number of groups desired.
`group_size`	An integer indicating the desired size of most of the groups. Note that if the number of nodes is not divisible into groups of equal size, there may be some larger or smaller groups.
`times`	Integer scalar, how many times the algorithm repeats its work. Where the algorithm is stochastic, this is how many times it runs, and the best or the most frequent result is kept. Where the algorithm searches, this is how many steps the search takes. More repetitions give a more reliable result and take longer, so each function documents its own default.
`num_groups`	Deprecated. The former spelling of `groups`. Still accepted, but warns; please use `groups` instead.

Details

`times` defaults to the number of nodes multiplied by the number of groups. This heuristic may be insufficient for small networks and numbers of groups, and burdensome for large ones, but can be overwritten. At every 10th iteration, a stronger perturbation of a number of successive changes, approximately the number of nodes divided by the number of groups, takes place whether or not it improves the objective function.

Value

A `node_member` character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

Maximally diverse grouping problem

This well known computational problem is a NP-hard problem with a number of relevant applications, including the formation of groups of students that have encountered each other least or least recently. Essentially, the aim is to return a membership of nodes in cliques that minimises the sum of their previous (weighted) ties:

$$\sum_{g=1}^{m}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} x_{ij}y_{ig}y_{jg}$$

where $y_{ig} = 1$ if node i is in group g, and 0 otherwise.

x_{ij} is the existing network data. If this is an empty network, the function will just return cliques. To run this repeatedly, one can join a clique network of the membership result with the original network, using this as the network data for the next round.

A form of the Lai and Hao (2016) iterated maxima search (IMS) is used here. This performs well for small and moderately sized networks. It includes both weak and strong perturbations to an

initial solution to ensure that a robust solution from the broader state space is identified. The user is referred to Lai and Hao (2016) and Lai et al (2021) for more details.

References

On the maximally diverse grouping problem:

Lai, Xiangjing, and Jin-Kao Hao. 2016. "Iterated Maxima Search for the Maximally Diverse Grouping Problem." *European Journal of Operational Research* 254(3):780–800. doi:10.1016/j.ejor.2016.05.018.

Lai, Xiangjing, Jin-Kao Hao, Zhang-Hua Fu, and Dong Yue. 2021. "Neighborhood Decomposition Based Variable Neighborhood Search and Tabu Search for Maximally Diverse Grouping." *European Journal of Operational Research* 289(3):1067–86. doi:10.1016/j.ejor.2020.07.048.

See Also

Other memberships: `member_brokerage`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
node_in_roulette(ison_adolescents, groups = 3)
```

---

member_community *Memberships in communities*

Description

`node_in_community()` returns a single community partition of a network, drawing on all the community detection algorithms available for that type of network.

By default it *selects* a partition. Where feasible (a small enough network), the optimal problem solving technique is used to ensure the maximal modularity partition. For larger networks, it identifies the applicable algorithms, runs each of them, and returns the partition with the largest modularity score.

Where `consensus = TRUE` it *combines* the partitions instead. Each applicable algorithm is run, the stochastic ones repeatedly, and the algorithms are then rerun on how often each pair of nodes is placed together until they agree. This costs considerably more time than selection, but does not rest the answer on a single run of a single algorithm.

Usage

```
node_in_community(
  .data,
  k = NULL,
  max_k = 8L,
  consensus = FALSE,
  times = 20,
  Kmax = NULL
)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
k	Integer indicating the target number of communities to return. By default NULL, in which case the algorithm returns the number of communities that it finds itself. Alternatively, a character string naming a selection method: "silhouette" selects the number that maximises the mean silhouette width over geodesic distances, "elbow" selects the number at the elbow of the coverage curve, and "strict" returns the partition in which no tie crosses a group, i.e. the components. Prefer "silhouette"; the elbow method is unreliable where the coverage curve has no clear elbow. If the algorithm cannot return exactly the number of communities requested, a warning is given and the nearest number is returned.
max_k	Integer indicating the maximum number of communities to evaluate for "silhouette" and "elbow". By default 8. Otherwise ignored. Note that for node_in_louvain() and node_in_leiden() each candidate requires its own search over the resolution parameter, so a large max_k is costly on large networks.
consensus	Logical, whether to combine the partitions of all the applicable algorithms instead of selecting the one with the highest modularity. By default FALSE, since combining them costs more time. This argument is ignored on a network small enough for node_in_optimal(), which already returns the maximum modularity partition.
times	Integer scalar, how many times the algorithm repeats its work. Where the algorithm is stochastic, this is how many times it runs, and the best or the most frequent result is kept. Where the algorithm searches, this is how many steps the search takes. More repetitions give a more reliable result and take longer, so each function documents its own default.
Kmax	Deprecated. The former spelling of max_k. Still accepted, but warns; please use max_k instead.

Details

times applies only when consensus = TRUE, and is 20 by default. Deterministic algorithms are run once however it is set.

Value

A node_member character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

References

On consensus community detection:

Lancichinetti, Andrea, and Santo Fortunato. 2012. "Consensus clustering in complex networks". *Scientific Reports* 2: 336. doi:10.1038/srep00336

Tagarelli, Andrea, Alessia Amelio, and Francesco Gullo. 2017. "Ensemble-based Community Detection in Multilayer Networks". *Data Mining and Knowledge Discovery* 31: 1506-1543. doi:10.1007/s1061801705288

See Also

Other community: member_community_hier, member_community_non

Other memberships: member_brokerage, member_cliques, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_in_community(ison_adolescents)
```

---

member_community_hier *Memberships in hierarchical communities*

Description

These functions offer algorithms for hierarchically clustering networks into communities. Since all of the following are hierarchical, their dendrograms can be plotted:

- node_in_betweenness() is a hierarchical, decomposition algorithm where edges are removed in decreasing order of the number of shortest paths passing through the edge.
- node_in_greedy() is a hierarchical, agglomerative algorithm, that tries to optimize modularity in a greedy manner.
- node_in_eigen() is a top-down, hierarchical algorithm.
- node_in_walktrap() is a hierarchical, agglomerative algorithm based on random walks.

The different algorithms offer various advantages in terms of computation time, availability on different types of networks, ability to maximise modularity, and their logic or domain of inspiration.

Usage

```
node_in_betweenness(.data, k = NULL, max_k = 8L, Kmax = NULL)

node_in_greedy(.data, k = NULL, max_k = 8L, Kmax = NULL)

node_in_eigen(.data, k = NULL, max_k = 8L, Kmax = NULL)

node_in_walktrap(.data, k = NULL, max_k = 8L, steps = 4, Kmax = NULL)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
k	Integer indicating the target number of communities to return. By default NULL, in which case the algorithm returns the number of communities that it finds itself. Alternatively, a character string naming a selection method: "silhouette" selects the number that maximises the mean silhouette width over geodesic distances, "elbow" selects the number at the elbow of the coverage curve, and "strict" returns the partition in which no tie crosses a group, i.e. the components. Prefer "silhouette"; the elbow method is unreliable where the coverage curve has no clear elbow. If the algorithm cannot return exactly the number of communities requested, a warning is given and the nearest number is returned.
max_k	Integer indicating the maximum number of communities to evaluate for "silhouette" and "elbow". By default 8. Otherwise ignored. Note that for node_in_louvain() and node_in_leiden() each candidate requires its own search over the resolution parameter, so a large max_k is costly on large networks.
Kmax	Deprecated. The former spelling of max_k. Still accepted, but warns; please use max_k instead.
steps	Integer indicating the length of the random walks. By default steps = 4, as in {igraph}. Longer walks reach further and tend to return fewer, larger communities.

Value

A node_member character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

Edge-betweenness

This is motivated by the idea that edges connecting different groups are more likely to lie on multiple shortest paths when they are the only option to go from one group to another. This method yields good results but is very slow because of the computational complexity of edge-betweenness calculations and the betweenness scores have to be re-calculated after every edge removal. Networks of ~700 nodes and ~3500 ties are around the upper size limit that are feasible with this approach.

Fast-greedy

Initially, each node is assigned a separate community. Communities are then merged iteratively such that each merge yields the largest increase in the current value of modularity, until no further increases to the modularity are possible. The method is fast and recommended as a first approximation because it has no parameters to tune. However, it is known to suffer from a resolution limit.

Leading eigenvector

In each step, the network is bifurcated such that modularity increases most. The splits are determined according to the leading eigenvector of the modularity matrix. A stopping condition prevents tightly connected groups from being split further. Note that due to the eigenvector calculations involved, this algorithm will perform poorly on degenerate networks, but will likely obtain a higher modularity than fast-greedy (at some cost of speed).

Walktrap

The general idea is that random walks on a network are more likely to stay within the same community because few edges lead outside a community. By repeating random walks of 4 steps many times, information about the hierarchical merging of communities is collected.

References

On edge-betweenness community detection:

Newman, Mark, and Michelle Girvan. 2004. "Finding and evaluating community structure in networks." *Physical Review E* 69: 026113. doi:10.1103/PhysRevE.69.026113

On fast-greedy community detection:

Clauset, Aaron, Mark E.J. Newman, and Cristopher Moore. 2004. "Finding community structure in very large networks." *Physical Review E*, 70: 066111. doi:10.1103/PhysRevE.70.066111

On leading eigenvector community detection:

Newman, Mark E.J. 2006. "Finding community structure using the eigenvectors of matrices" *Physical Review E* 74:036104. doi:10.1103/PhysRevE.74.036104

On walktrap community detection:

Pons, Pascal, and Matthieu Latapy. 2005. "Computing communities in large networks using random walks". 1-20. doi:10.48550/arXiv.physics/0512106

See Also

Other memberships: `member_brokerage`, `member_cliques`, `member_community`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`,

motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Other community: member_community, member_community_non

Examples

```
node_in_betweenness(ison_adolescents)
node_in_greedy(ison_adolescents)
node_in_eigen(ison_adolescents)
node_in_walktrap(ison_adolescents)
```

---

member_community_non *Memberships in non-hierarchical communities*

Description

These functions offer algorithms for partitioning networks into sets of communities:

- node_in_optimal() is a problem-solving algorithm that seeks to maximise modularity over all possible partitions.
- node_in_partition() is a greedy, iterative, deterministic partitioning algorithm that results in two equally-sized communities.
- node_in_infomap() is an algorithm based on the information in random walks.
- node_in_spinglass() is a greedy, iterative, probabilistic algorithm, based on analogy to model from statistical physics.
- node_in_fluid() is a propogation-based partitioning algorithm, based on analogy to model from fluid dynamics.
- node_in_louvain() is an agglomerative multilevel algorithm that seeks to maximise modularity over all possible partitions.
- node_in_leiden() is an agglomerative multilevel algorithm that seeks to maximise the Constant Potts Model over all possible partitions.
- node_in_labels() is a fast, propagation-based algorithm in which nodes iteratively adopt whichever community label is most common among their neighbours.

The different algorithms offer various advantages in terms of computation time, availability on different types of networks, ability to maximise modularity, and their logic or domain of inspiration.

Usage

```
node_in_optimal(.data)

node_in_partition(.data, k = 2L, max_k = 8L, Kmax = NULL)

node_in_infomap(.data, times = 50)

node_in_spinglass(.data, max_k = 200, resolution = 1)
```

```
node_in_fluid(.data, k = NULL, max_k = 8L, Kmax = NULL)

node_in_louvain(.data, k = NULL, max_k = 8L, resolution = 1, Kmax = NULL)

node_in_leiden(.data, k = NULL, max_k = 8L, resolution = 1, Kmax = NULL)

node_in_labels(.data, k = NULL, max_k = 8L, Kmax = NULL)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`k`	Integer indicating the target number of communities to return. By default `NULL`, in which case the algorithm returns the number of communities that it finds itself. Alternatively, a character string naming a selection method: `"silhouette"` selects the number that maximises the mean silhouette width over geodesic distances, `"elbow"` selects the number at the elbow of the coverage curve, and `"strict"` returns the partition in which no tie crosses a group, i.e. the components. Prefer `"silhouette"`; the elbow method is unreliable where the coverage curve has no clear elbow. If the algorithm cannot return exactly the number of communities requested, a warning is given and the nearest number is returned.
`max_k`	Integer indicating the maximum number of communities to evaluate for `"silhouette"` and `"elbow"`. By default `8`. Otherwise ignored. Note that for `node_in_louvain()` and `node_in_leiden()` each candidate requires its own search over the resolution parameter, so a large `max_k` is costly on large networks.
`Kmax`	Deprecated. The former spelling of `max_k`. Still accepted, but warns; please use `max_k` instead.
`times`	Integer scalar, how many times the algorithm repeats its work. Where the algorithm is stochastic, this is how many times it runs, and the best or the most frequent result is kept. Where the algorithm searches, this is how many steps the search takes. More repetitions give a more reliable result and take longer, so each function documents its own default.
`resolution`	The Reichardt-Bornholdt "gamma" resolution parameter for modularity. By default 1, making existing and non-existing ties equally important. Smaller values make existing ties more important, and larger values make missing ties more important.

Value

A `node_member` character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

Optimal

The general idea is to calculate the modularity of all possible partitions, and choose the community structure that maximises this modularity measure. Note that this is an NP-complete problem with exponential time complexity. The guidance in the igraph package is networks of <50-200 nodes is probably fine.

Partition

The general idea is to assign nodes to two groups, and then iteratively swap pairs of nodes (one from each group) that give a positive sum of net tie costs, where the net tie cost of a node is the difference between the sum of the weights of ties to nodes in the other group (external costs) and the sum of the weights of ties to nodes in the same group (internal costs). Where `k` is greater than two, the same swap pass is run for every pair of groups, and the rounds repeat until no swap improves the partition. This is a deterministic algorithm that will always return the same partition for a given network, but it is not guaranteed to maximise modularity. Note that this algorithm is only applicable to undirected, unipartite networks, and returns `k` communities of equal size (or as close to equal as possible).

Infomap

Motivated by information theoretic principles, this algorithm tries to build a grouping that provides the shortest description length for a random walk, where the description length is measured by the expected number of bits per node required to encode the path.

Spin-glass

Here `max_k` is the number of spins, an upper limit on the communities found rather than a bound on a search, so some can end up empty.

This is motivated by analogy to the Potts model in statistical physics. Each node can be in one of k "spin states", and ties (particle interactions) provide information about which pairs of nodes want similar or different spin states. The final community definitions are represented by the nodes' spin states after a number of updates. A different implementation than the default is used in the case of signed networks, such that nodes connected by negative ties will be more likely found in separate communities.

Fluid

The general idea is to observe how a discrete number of fluids interact, expand and contract, in a non-homogenous environment, i.e. the network structure. Unlike the `{igraph}` implementation that this function wraps, this function iterates over all possible numbers of communities and returns the membership associated with the highest modularity.

Louvain

The general idea is to take a hierarchical approach to optimising the modularity criterion. Nodes begin in their own communities and are re-assigned in a local, greedy way: each node is moved to the

community where it achieves the highest contribution to modularity. When no further modularity-increasing reassignments are possible, the resulting communities are considered nodes (like a reduced graph), and the process continues. Where `k` is given, the resolution parameter is searched for the value that returns that number of communities, and `resolution` is ignored.

Leiden

The general idea is to optimise the Constant Potts Model, which does not suffer from the resolution limit, instead of modularity. As outlined in the `{igraph}` package, the Constant Potts Model object function is:

$$\frac{1}{2m}\sum_{ij}(A_{ij}-\gamma n_i n_j)\delta(\sigma_i,\sigma_j)$$

where *m* is the total tie weight, A_{ij} is the tie weight between *i* and *j*, γ is the so-called resolution parameter, n_i is the node weight of node *i*, and $\delta(\sigma_i,\sigma_j)=1$ if and only if *i* and *j* are in the same communities and 0 otherwise. Compared to the Louvain method, the Leiden algorithm additionally tries to avoid unconnected communities. Where `k` is given, the resolution parameter is searched for the value that returns that number of communities, and `resolution` is ignored.

Label propagation

Every node is initially given a unique label. Nodes are then visited in random order, each adopting whichever label is most frequent among its neighbours, until no node has a label that a majority of its neighbours does not share. Densely connected groups quickly converge on a common label, which is what makes the communities.

This is the fastest of the algorithms here, running in near-linear time, which makes it useful on large networks where the others are infeasible. The trade-off is that it is stochastic: because both the visiting order and ties between equally frequent labels are broken at random, repeated runs on the same network can return different partitions, and on sparse networks it may return a single community. Set a seed for reproducibility, or use `node_in_community()` to select among algorithms by modularity.

Where `k` is given, the algorithm becomes semi-supervised. The `k` nodes of highest degree are each given a distinct, fixed label, every other node starts with a label of its own, and propagation runs as normal. Seeding alone tends to leave more than `k` labels standing, so any surplus groups are then merged in the order that best preserves modularity, until exactly `k` communities remain.

References

On optimal community detection:

Brandes, Ulrik, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, Dorothea Wagner. 2008. "On Modularity Clustering", *IEEE Transactions on Knowledge and Data Engineering* 20(2):172-188.

On partitioning community detection:

Kernighan, Brian W., and Shen Lin. 1970. "An efficient heuristic procedure for partitioning graphs." *The Bell System Technical Journal* 49(2): 291-307. doi:10.1002/j.15387305.1970.tb01770.x

On infomap community detection:

Rosvall, M, and C. T. Bergstrom. 2008. "Maps of information flow reveal community structure in complex networks", *PNAS* 105:1118. doi:10.1073/pnas.0706851105

Rosvall, M., D. Axelsson, and C. T. Bergstrom. 2009. "The map equation", *Eur. Phys. J. Special Topics* 178: 13. doi:10.1140/epjst/e2010011791

On spinglass community detection:

Reichardt, Jorg, and Stefan Bornholdt. 2006. "Statistical Mechanics of Community Detection" *Physical Review E*, 74(1): 016110–14. doi:10.1073/pnas.0605965104

Traag, Vincent A., and Jeroen Bruggeman. 2009. "Community detection in networks with positive and negative links". *Physical Review E*, 80(3): 036115. doi:10.1103/PhysRevE.80.036115

On fluid community detection:

Parés Ferran, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguade, Jesus Labarta, Ulises Cortes, and Toyotaro Suzumura. 2018. "Fluid Communities: A Competitive, Scalable and Diverse Community Detection Algorithm". In: *Complex Networks & Their Applications VI* Springer, 689: 229. doi:10.1007/9783319721507_19

On Louvain community detection:

Blondel, Vincent, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre. 2008. "Fast unfolding of communities in large networks", *J. Stat. Mech.* P10008.

On Leiden community detection:

Traag, Vincent A., Ludo Waltman, and Nees Jan van Eck. 2019. "From Louvain to Leiden: guaranteeing well-connected communities", *Scientific Reports*, 9(1):5233. doi:10.1038/s41598-01941695z

On label propagation community detection:

Raghavan, Usha Nandini, Reka Albert, and Soundar Kumara. 2007. "Near linear time algorithm to detect community structures in large-scale networks", *Physical Review E*, 76(3):036106. doi:10.1103/PhysRevE.76.036106

See Also

Other community: `member_community`, `member_community_hier`

Other memberships: `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
node_in_optimal(ison_adolescents)
node_in_partition(ison_adolescents)
node_in_partition(ison_southern_women)
node_in_infomap(ison_adolescents)
node_in_spinglass(ison_adolescents)
node_in_fluid(ison_adolescents)
node_in_louvain(ison_adolescents)
node_in_leiden(ison_adolescents)
node_in_labels(ison_adolescents)
```

---

member_components *Memberships in components*

Description

These functions create a vector of nodes' memberships in components:

- `node_in_component()` assigns nodes' component membership, in either the strongly or the weakly connected components.

In graph theory, components, sometimes called connected components, are induced subgraphs from partitioning the nodes into disjoint sets. All nodes that are members of the same partition as *i* are reachable from *i*.

For directed networks, strongly connected components consist of subgraphs where there are paths in each direction between member nodes. Weakly connected components consist of subgraphs where there is a path in either direction between member nodes.

Usage

```
node_in_component(.data, connectivity = c("strong", "weak"))
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`connectivity`	Character string, "weak" treats a directed network's components as if the network were undirected, and "strong" requires ties in both directions between members. This is ignored for undirected networks, where the two notions coincide. Note that the default differs by function: functions that assert or count connectedness default to "strong", while functions that scope or split a network into components default to "weak".

Value

A `node_member` character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

See Also

Other memberships: member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_core, member_diffusion, member_equivalence

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
ison_monks |> to_uniplex("esteem") |>
  mutate_nodes(comp = node_in_component())
ison_monks |> to_uniplex("esteem") |>
  mutate_nodes(comp = node_in_component(connectivity = "weak"))
```

---

member_core *Memberships in core-periphery categories*

Description

node_in_core() categorizes nodes into two or more core/periphery categories based on their coreness.

Usage

```
node_in_core(
  .data,
  groups = 3,
  split = c("bins", "quantiles", "kmeans"),
  coreness = NULL,
  direction = c("all", "out", "in", "both"),
  cluster_by = NULL
)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
groups	Number of categories to create. Must be at least 2 and at most the number of nodes in the network. Default is 3.
split	Which method to use to split the coreness scores into the categories. One of "bins" (equal-width bins), "quantiles" (quantile-based bins), or "kmeans" (k-means clustering); see method_split for what each does. Default is "bins".

coreness	Which method to use to calculate nodes' coreness. One of "correlation", "rich", "transition", or "hub"; see method_coreness for what each does. By default NULL, which uses "rich" for a weighted, directed, or two-mode network, since it is the only method that reads those properties directly, and "correlation" otherwise.
direction	One of "all" (the default), "out", "in", or "both". For a directed network, "out" scores nodes on the ties they send and "in" on the ties they receive, while "both" returns the four categories described below. Ignored for undirected and two-mode networks.
cluster_by	Deprecated. The former spelling of split. Still accepted, but warns; please use split instead.

Value

A node_member character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

Core-periphery categories

This function categorizes nodes based on their coreness into a specified number of groups. The groups are labeled as "Core", "Semi-core", "Semi-periphery", and "Periphery" depending on the number of groups specified. The categorization can be done using different methods: equal-width bins, quantile-based bins, or k-means clustering.

Directed core-periphery

In a directed network a node can be core in whom it reaches and peripheral in who reaches it, which one core and one periphery cannot express. direction = "both" therefore returns the four categories that Elliott and colleagues distinguish:

- "Core" for nodes in both the out-core and the in-core,
- "Sender" for nodes in the out-core only,
- "Receiver" for nodes in the in-core only,
- "Periphery" for nodes in neither.

This uses coreness_hub(), so groups and split do not apply.

References

On core-periphery categorization:

Wallerstein, Immanuel. 1974. "Dependence in an Interdependent World: The Limited Possibilities of Transformation Within the Capitalist World Economy." *African Studies Review*, 17(1), 1-26. doi:10.2307/523574

On directed core-periphery:

Elliott, Andrew, Angus Chiu, Marya Bazzi, Gesine Reinert, and Mihai Cucuringu. 2020. "Core-periphery structure in directed networks". *Proceedings of the Royal Society A* 476(2241): 20190783. doi:10.1098/rspa.2019.0783

See Also

Other core-periphery: mark_core, measure_core

Other memberships: member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_diffusion, member_equivalence

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_in_core(ison_adolescents)
node_in_core(ison_networkers, direction = "both")
```

---

member_diffusion *Memberships in a diffusion process*

Description

node_in_adopter() classifies membership of nodes into diffusion categories by where on the distribution of adopters they fell. Valente (1995) defines five memberships:

- *Early adopter*: those with an adoption time less than the average adoption time minus one standard deviation of adoptions times
- *Early majority*: those with an adoption time between the average adoption time and the average adoption time minus one standard deviation of adoptions times
- *Late majority*: those with an adoption time between the average adoption time and the average adoption time plus one standard deviation of adoptions times
- *Laggard*: those with an adoption time greater than the average adoption time plus one standard deviation of adoptions times
- *Non-adopter*: those without an adoption time, i.e. never adopted

Usage

```
node_in_adopter(.data)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().

Value

A node_member character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

References

On adopter classes:

Valente, Tom W. 1995. *Network models of the diffusion of innovations* (2nd ed.). Cresskill N.J.: Hampton Press.

See Also

Other memberships: member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_equivalence

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Other diffusion: mark_diff, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, motif_exposure, motif_hazard

Examples

```
smeg <- generate_smallworld(15, 0.025)
smeg_diff <- play_diffusion(smeg, recovery = 0.2)
# To classify nodes by their position in the adoption curve
(adopts <- node_in_adopter(smeg_diff))
summary(adopts)
```

---

member_equivalence *Memberships in equivalent classes*

Description

These functions combine an appropriate node_x_*() function together with methods for calculating the hierarchical clusters provided by a certain distance calculation.

- node_in_equivalence() assigns nodes membership based on their equivalence with respective to some motif/class. The following functions call this function, together with an appropriate motif.
- node_in_structural() assigns nodes membership based on their having equivalent ties to the same other nodes.

- node_in_regular() assigns nodes membership based on their having equivalent patterns of ties to equivalent others.
- node_in_automorphic() assigns nodes membership based on their having equivalent distances to other nodes.
- node_in_motif() assigns nodes membership based on their participating in local structures at similar rates.

A plot() method exists for investigating the dendrogram of the hierarchical cluster and showing the returned cluster assignment.

Usage

```
node_in_equivalence(
  .data,
  motif,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor", "cosine"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  max_k = 8L,
  Kmax = NULL
)

node_in_structural(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor", "cosine"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  max_k = 8L,
  Kmax = NULL
)

node_in_regular(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor", "cosine"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  max_k = 8L,
  regularity = c("rolesim", "rege"),
  decay = 0.15,
  beta = NULL,
  Kmax = NULL
)

node_in_motif(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor", "cosine"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  max_k = 8L,
```

```
  Kmax = NULL
)

node_in_automorphic(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor", "cosine"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  max_k = 8L,
  Kmax = NULL
)

node_in_block(.data, k = 2L, blocks = c("nul", "com"), times = NULL)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
motif	A matrix returned by a node_x_*() function.
k	Typically a character string indicating which method should be used to select the number of clusters to return. By default "silhouette", other options include "elbow" and "strict". "strict" returns classes with members only when strictly equivalent. "silhouette" and "elbow" select classes based on the distance between clusters or between nodes within a cluster. Fewer, identifiable letters, e.g. "e" for elbow, is sufficient. Alternatively, if k is passed an integer, e.g. k = 3, then all selection routines are skipped in favour of this number of clusters.
cluster	Character string indicating whether clusters should be clustered hierarchically ("hierarchical") or through convergence of correlations ("concor"). Fewer, identifiable letters, e.g. "c" for CONCOR, is sufficient.
distance	Character string indicating which distance metric to pass on to stats::dist. By default "euclidean", but other options include "maximum", "manhattan", "canberra", "binary", and "minkowski". Fewer, identifiable letters, e.g. "e" for Euclidean, is sufficient.
max_k	Integer indicating the maximum number of (k) clusters to evaluate. Ignored when k = "strict" or a discrete number is given for k.
Kmax	Deprecated. The former spelling of max_k. Still accepted, but warns; please use max_k instead.
regularity	Character string indicating which algorithm should be used to calculate how regularly equivalent nodes are. By default "rolesim"; "rege" is also available. Fewer, identifiable letters, e.g. "ro" for RoleSim, is sufficient. See regularity_rolesim() and regularity_rege() for how they differ.
decay	A proportion between 0 and 1 giving how much of a contribution survives each additional step of distance or walk length. Lower values discount more steeply, so that only nearby others count; higher values discount less, so that longer

	walks continue to contribute. The measures that take a `decay` differ in what they discount and in what value leaves the measure in its most familiar form, so each documents its own default.
`beta`	Deprecated; use `decay` instead.
`blocks`	A character vector of permitted ideal block types, or a list-matrix giving the permitted types per block position. See `net_by_inconsistency()` for the available types.
`times`	Integer number of search iterations. By default the number of nodes times the number of positions.

Value

A `node_member` character vector the length of the nodes in the network, of group memberships "A", "B", etc for each node. If the network is labelled, then the assignments will be labelled with the nodes' names.

Regular equivalence

Two nodes are regularly equivalent if each has ties to the same *kinds* of others, even where those others are not the same individuals and are not equally numerous. A manager with three subordinates and a manager with ten are regularly equivalent, because what makes them alike is that they both have subordinates, not how many or which.

The definition is recursive: nodes are equivalent if their alters are equivalent, whose equivalence depends in turn on *their* alters. `node_in_regular()` therefore computes a similarity matrix by iterating that definition to a fixed point, and then clusters it in the same way as the other functions here.

Note that this differs from `node_in_motif()`, which compares nodes on how often they appear embedded in local structures. Two nodes can have very similar triad profiles without being regularly equivalent, and vice versa, since a motif census counts a node's local configurations while regular equivalence asks who its alters are.

Motif equivalence

Where the other functions here compare nodes on *whom* they are tied to, `node_in_motif()` compares them on *what kinds of local structure* they sit in, by clustering a census of the triads (or, for two-mode networks, tetrads) each node participates in.

Note that the census counts the *types* of motif a node takes part in, and not the position it holds within them. In the path $i \to k \to j$, for example, all three nodes return a profile of one 021C triad, although i sends, k mediates and j receives. This is therefore neither Burt's role equivalence, which distinguishes those positions, nor the orbit-aware census of Ortmann and Brandes, which netrics does not yet offer.

What it captures is similarity of local embedding. It is well suited to distinguishing nodes that sit in dense, closed neighbourhoods from those that bridge open ones, but it is not regular equivalence: see `node_in_regular()` for that.

This function was called `node_in_regular()` prior to version 1.0.0.

Direct blockmodelling

The other functions here are *indirect*: they build a similarity between nodes, cluster it, and read a partition off the result. `node_in_block()` is *direct*. It searches the space of partitions for the one that best fits an ideal block structure, scoring each candidate with `net_by_inconsistency()` and keeping whichever is most consistent.

The advantage is that the criterion being optimised is the one you actually care about, rather than a similarity that stands in for it, and that ideal types other than "null and complete" become available — `blocks = c("nul", "reg")` searches directly for a regular-equivalence blockmodel. The cost is that the number of positions `k` must be chosen in advance, and that the search is stochastic: it explores by random restarts and perturbations, so repeated runs may return different partitions and a longer search is more likely to find a good one. Set a seed for reproducibility, and compare runs with `net_by_inconsistency()`.

Source

https://github.com/aslez/concoR

References

On role equivalence:

Burt, Ronald S. 1990. "Detecting role equivalence". *Social Networks* 12(1): 83-97. doi:10.1016/03788733(90)900233

On the orbit-aware census:

Ortmann, Mark, and Ulrik Brandes. 2017. "Efficient orbit-aware triad and quad census in directed and undirected graphs". *Applied Network Science* 2(1): 13. doi:10.1007/s4110901700272

On direct blockmodelling:

Doreian, Patrick, Vladimir Batagelj, and Anuska Ferligoj. 2005. *Generalized Blockmodeling*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511584176

See Also

Other memberships: `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
(nse <- node_in_structural(ison_algebra))
(nre <- node_in_regular(ison_southern_women))
(nme <- node_in_motif(ison_southern_women, cluster = "concor"))
```

```
if(require("sna", quietly = TRUE)){
(nae <- node_in_automorphic(ison_southern_women,
  k = "elbow"))
}
(nbm <- node_in_block(ison_adolescents, k = 3))
net_by_inconsistency(ison_adolescents, nbm)
```

`method_cluster` *Methods for equivalence clustering*

Description

These functions are used to cluster some motif census object:

- `cluster_hierarchical()` returns a hierarchical clustering object created by `stats::hclust()`.
- `cluster_concor()` returns a hierarchical clustering object created from a convergence of correlations procedure (CONCOR).
- `cluster_cosine()` returns a hierarchical clustering object created by `stats::hclust()` on cosine dissimilarities, rather than correlations, as created by `to_cosine()`.

These functions are not intended to be called directly, but are called within `node_in_equivalence()` and related functions. They are exported and listed here to provide more detailed documentation.

Usage

```
cluster_hierarchical(motif, distance)

cluster_cosine(motif, distance)

cluster_concor(.data, motif)
```

Arguments

`motif`	A matrix returned by a `node_x_*()` function.
`distance`	Character string indicating which distance metric to pass on to `stats::dist`. By default `"euclidean"`, but other options include `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"`, and `"minkowski"`. Fewer, identifiable letters, e.g. `"e"` for Euclidean, is sufficient.
`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A hierarchical clustering object created by `stats::hclust()`, with an additional `distances` attribute containing the distance matrix used for clustering.

Hierarchical clustering

This method uses `stats::hclust()` to create a hierarchical clustering object from a distance matrix created from the correlations between nodes' profiles in the given motif census. First a matrix of Pearson correlation coefficients between each pair of nodes' profiles in the given motif census is created. Then a distance matrix is created by subtracting these correlations from 1, and this is given to `stats::hclust()` to enable dendrogram construction etc.

Cosine similarity

This method is similar to the hierarchical clustering method, but uses cosine similarity rather than correlation as the clustering basis. First a matrix of cosine similarities between each pair of nodes' profiles in the given census is created. Then a distance matrix is created by subtracting these similarities from 1, and this is given to `stats::hclust` to enable dendrogram construction etc.

CONCOR

First a matrix of Pearson correlation coefficients between each pair of nodes' profiles in the given motif census is created. Then, again, we find the correlations of this square, symmetric matrix, and continue to do this iteratively until each entry is either 1 or -1. These values are used to split the data into two partitions, with members either holding the values 1 or -1. This procedure from census to convergence is then repeated within each block, allowing further partitions to be found. Unlike UCINET, partitions are continued until there are single members in each partition. Then a distance matrix is constructed from records of in which partition phase nodes were separated, and this is given to `stats::hclust()` so that dendrograms etc can be returned.

References

On CONCOR clustering:

Breiger, Ronald L., Scott A. Boorman, and Phipps Arabie. 1975. "An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling". *Journal of Mathematical Psychology*, 12: 328-83. doi:10.1016/0022-2496(75)900280.

`method_coreness` *Methods for calculating coreness*

Description

These functions calculate how core-like each node is, returning both a continuous coreness score and a core/periphery split that `node_is_core()`, `node_by_core()` and `node_in_core()` then use.

- `coreness_correlation()` fits the network to an ideal core-periphery pattern by correlation.
- `coreness_rich()` ranks nodes by strength and cuts where the tie weight to higher-ranked neighbours peaks.
- `coreness_transition()` scores nodes with a transition function whose sharpness and core size are free parameters.

- `coreness_hub()` scores nodes by how well they send to and receive from the core, which lets core and periphery differ by tie direction.

They differ in what they can use. `coreness_rich()` and `coreness_hub()` read tie direction and tie weights directly. `coreness_correlation()` and `coreness_transition()` compare the network against a symmetric ideal, so they symmetrise a directed network first and report that they have done so.

Usage

```
coreness_correlation(.data, direction = c("all", "out", "in"), starts = 5L)

coreness_rich(.data, direction = c("all", "out", "in"))

coreness_transition(
  .data,
  direction = c("all", "out", "in"),
  alpha = seq(0.2, 0.8, 0.2),
  beta = seq(0.2, 0.8, 0.2)
)

coreness_hub(.data, direction = c("all", "out", "in"))
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`direction`	One of "all" (the default), "out", or "in". For a directed network, "out" scores nodes on the ties they send and "in" on the ties they receive. Ignored for undirected and two-mode networks.
`starts`	Integer number of starting points for the search, at most 9. By default 5. The starting points are fixed rather than random, so that two calls on the same network return the same answer.
`alpha`	Numeric vector of boundary sharpness values between 0 and 1, to aggregate over. By default `seq(0.2, 0.8, 0.2)`.
`beta`	Numeric vector of core size values between 0 and 1, to aggregate over. By default `seq(0.2, 0.8, 0.2)`.

Value

A list with two elements:

- `coreness`: a numeric vector between 0 and 1, one value per node, for how core-like each node is.
- `core`: a logical vector, one value per node, TRUE for the core.

`coreness_hub()` adds `out_core` and `in_core`, the two core sets that a directed core-periphery structure distinguishes.

Correlation

Borgatti and Everett's continuous model gives each node a coreness c_i between 0 and 1, and compares the network against the ideal pattern $c_i c_j$ in which two nodes are tied to the extent that both are core:

$$\rho = \text{cor}(A_{ij}, c_i c_j), i \neq j$$

The coreness vector that maximises ρ is the fitted model. Self-ties are excluded from the correlation, since no node is tied to itself and including the diagonal pulls every coreness toward zero.

The problem is not convex, so the search is run from several starting points, ordered by degree, and the best fit is kept. A weighted network is fitted to its weights, which means that the ideal pattern is read as how *strongly* two core nodes should be tied. To fit the pattern of ties instead of their weights, use `manynet::to_unweighted()` first.

The search has one free value per node, so its cost grows quickly with the size of the network. On a large network, lower `starts`, or use `coreness_rich()`, which needs no search at all.

Rich-core

Ma and Mondragon rank the nodes by strength, from strongest to weakest, and give each node the total weight of its ties to nodes that rank above it:

$$\sigma_i^+ = \sum_{j:r_j<r_i} w_{ij}$$

Walking down the ranking, σ^+ rises while the nodes added are still tied to those already above them, and falls once they are not. The rank at which it peaks is the boundary of the rich core.

The method needs no parameters and no optimisation, and it reads tie weights and tie direction directly, which makes it the method this package uses by default for a weighted, directed, or two-mode network. For a two-mode network the nodes of both modes are ranked together, so the core may span both.

Note that the core it finds is one whose members are tied to *each other*. Where a directed network instead has one set that sends and a different set that receives, σ^+ never rises, and the method returns a core of one or two nodes. Use `coreness_hub()` for that structure, which keeps the two sets apart rather than trying to merge them.

A rich core is not a rich club, which is why this method is not named for one. A rich club requires the high-degree nodes to be densely tied to one another, and `net_by_richclub()` measures that density. A rich core only marks the rank at which nodes stop linking upward, so a network can have a rich core whose members are not densely tied. The rich core also needs no null model, where the rich-club coefficient does, since that coefficient rises with degree even in a random network.

Transition

Rombach and colleagues score the node at rank m with a transition function

$$C_m = \frac{1}{1 + \exp(-(m - N\beta)\tan(\pi\alpha/2))}$$

where α sets how sharp the boundary between core and periphery is, from fuzziest at 0 to a clean step at 1, and β sets how large the core is, from every node at 0 to none at 1. The ordering that maximises the core quality $R = \sum_{ij} A_{ij} C_i C_j$ is the fitted model.

No single α and β is right for every network, so the score is aggregated over a grid of both, weighting each by the core quality it achieves, and scaled so that the most core-like node is 1.

Hub

In a directed network a node can be core in whom it reaches and peripheral in who reaches it. Elliott and colleagues therefore keep two core sets rather than one: an out-core of nodes that send to the core, and an in-core of nodes that receive from it.

The two are read from the hub and authority scores that `node_by_hub()` and `node_by_authority()` already provide: a hub is a node that points to good authorities, and an authority is a node that good hubs point to, which is the same mutual definition the two core sets have. Each set is then cut by the same rule the other methods use. With `direction = "all"` the returned coreness is the geometric mean of the two scores, and the core is the set of nodes in both.

References

On the correlation method:

Borgatti, Stephen P., and Martin G. Everett. 2000. "Models of core/periphery structures". *Social Networks* 21(4): 375-395. doi:10.1016/S03788733(99)000192

Lip, Sean Z. W. 2011. "A fast algorithm for the discrete core/periphery bipartitioning problem". doi:10.48550/arXiv.1102.5511

On the rich-core method:

Ma, Athen, and Raul J. Mondragon. 2015. "Rich-cores in networks". *PLoS ONE* 10(3): e0119678. doi:10.1371/journal.pone.0119678

On the transition method:

Rombach, Puck, Mason A. Porter, James H. Fowler, and Peter J. Mucha. 2017. "Core-periphery structure in networks (revisited)". *SIAM Review* 59(3): 619-646. doi:10.1137/17M1130046

On the hub method:

Elliott, Andrew, Angus Chiu, Marya Bazzi, Gesine Reinert, and Mihai Cucuringu. 2020. "Core-periphery structure in directed networks". *Proceedings of the Royal Society A* 476(2241): 20190783. doi:10.1098/rspa.2019.0783

See Also

Other methods: `method_regularity`

Examples

```
coreness_correlation(ison_adolescents)
coreness_rich(ison_networkers)
coreness_transition(ison_adolescents)
coreness_hub(ison_networkers)
```

`method_kselect` *Methods for selecting clusters*

Description

Finding the optimal number of clusters is generally a balance between optimal fit statistics, parsimony, and interpretability. These functions help select the number of clusters to return from `hc`, some hierarchical clustering object:

- `k_strict()` selects a number of clusters in which there is no distance between cluster members.
- `k_elbow()` selects a number of clusters in which there is a fair trade-off between parsimony and fit according to the elbow method.
- `k_silhouette()` selects a number of clusters that maximises the silhouette score.

These functions are generally not user-facing but used internally in e.g. the `*_equivalence()` functions.

Usage

```
k_strict(hc, .data)

k_elbow(hc, .data, motif, max_k)

k_silhouette(hc, .data, max_k)

k_gap(hc, motif, max_k, sims = 100)
```

Arguments

`hc`	A hierarchical clustering object.
`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`motif`	A motif census object.
`max_k`	An integer indicating the maximum number of options to consider. The minimum of this and the number of nodes in the network is used.
`sims`	Integer of how many simulations should be generated as a reference distribution.

Value

A single integer indicating the number of clusters to return.

Strict method

The strict method selects the number of clusters in which there is no distance between cluster members. This is a very conservative method that may be appropriate when the goal is to identify clusters of nodes that are exactly the same. However, it may not be appropriate in cases where the data is noisy or when the clusters are not well-defined, as it may result in a large number of small clusters.

Elbow method

The elbow method is a heuristic used in cluster analysis to determine the optimal number of clusters. It is based on the idea of plotting the within cluster correlation as a function of the number of clusters and looking for an "elbow" where there is a significant decrease in the rate of improvement in correlation as the number of clusters increases. The point at which the elbow occurs is often considered a good choice for the number of clusters, as it represents a balance between model complexity and fit to the data.

The elbow is located geometrically. A straight line is drawn between the first and the last point of the curve. The perpendicular distance from each point to this line is measured, and the point at the greatest distance is the elbow. Note that where the curve is close to a straight line, no point stands out and the method returns one of the endpoints.

Silhouette method

The silhouette method is based on the concept of cohesion and separation. Cohesion refers to how closely related the nodes within a cluster are, while separation refers to how distinct the clusters are from each other. The silhouette score combines these two concepts into a single metric that can be used to evaluate the quality of a clustering solution. The silhouette score is calculated as follows: For each node, calculate the average distance to all other nodes in the same cluster (a) and the average distance to all other nodes in the next nearest cluster (b). The silhouette score for each node is then calculated as:

$$S(i) = \frac{b - a}{\max(a, b)}$$

A higher silhouette score indicates that the node is well-matched to its own cluster and poorly matched to neighboring clusters. The silhouette score for the entire clustering is the average silhouette score across all nodes. Maximizing the silhouette score across a range of potential clusterings allows researchers to identify the number of clusters that best captures the underlying structure of the data. It is particularly useful when the clusters are well-separated.

References

On the elbow method:

Thorndike, Robert L. 1953. "Who Belongs in the Family?". *Psychometrika*, 18(4): 267–76. doi:10.1007/BF02289263.

On the silhouette method:

Rousseeuw, Peter J. 1987. “Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis.” *Journal of Computational and Applied Mathematics*, 20: 53–65. doi:10.1016/03770427(87)901257.

`method_regularity` *Methods for calculating regularity*

Description

These functions calculate how regularly equivalent each pair of nodes is, returning a similarity matrix that `node_in_regular()` then clusters.

- `regularity_rolesim()` calculates RoleSim similarity.
- `regularity_rege()` calculates REGE similarity.

Both are recursive: two nodes are similar to the extent that their alters are similar, which is the defining property of regular equivalence. They differ in how they pair up two nodes' alters.

Usage

```
regularity_rolesim(.data, decay = 0.15, beta = NULL)

regularity_rege(.data, iterations = 3)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`decay`	A proportion between 0 and 1 giving how much of a contribution survives each additional step of distance or walk length. Lower values discount more steeply, so that only nearby others count; higher values discount less, so that longer walks continue to contribute. The measures that take a `decay` differ in what they discount and in what value leaves the measure in its most familiar form, so each documents its own default.
`beta`	Deprecated; use `decay` instead.
`iterations`	Integer number of iterations. By default 3 for `regularity_rege()`; `regularity_rolesim()` iterates to convergence.

Value

A square similarity matrix with one row and column per node.

RoleSim

RoleSim pairs up two nodes' alters by finding the *maximal matching* between them, that is, the one-to-one pairing that maximises total similarity, and then averages over it:

$$s(u,v) = (1-\delta)\frac{\sum_{(x,y)\in M} s(x,y)}{|N(u)| + |N(v)| - |M|} + \delta$$

where M is that matching and δ is `decay`, which RoleSim calls β; by default 0.15. Because each alter can be used only once, two nodes are similar only if their neighbourhoods can be lined up as wholes.

RoleSim satisfies the automorphic confirmation property, meaning that automorphically equivalent nodes always score 1, and it is a metric. It converges to a unique solution regardless of where it starts, so the result does not depend on initialisation.

REGE

REGE instead pairs each alter with its *best* counterpart, allowing the same alter to be used more than once:

$$s(u,v) = \frac{\sum_{x \in N(u)} \max_{y \in N(v)} s(x,y) + \sum_{y \in N(v)} \max_{x \in N(u)} s(x,y)}{|N(u)| + |N(v)|}$$

Matching with replacement makes REGE more permissive than RoleSim: a node with many alters can be judged similar to one with few, if those few resemble all of the many. Which behaviour is wanted depends on whether having more alters of a kind is itself part of the role.

REGE is the algorithm UCINET implements, so use it when comparing results against that software. Unlike RoleSim it has no convergence guarantee and is sensitive to the number of iterations, so this is fixed rather than run to convergence.

Note that REGE is defined for *valued* networks, and weights each matched pair by how similar the two ties' strengths are. On an unweighted, connected network it is degenerate: since every node has an alter that matches every other node's alter perfectly, all nodes come out maximally equivalent, which is the correct but uninformative answer that the maximal regular equivalence of a connected graph is a single class. Use `regularity_rolesim()` for unweighted networks.

References

On RoleSim:

Jin, Ruoming, Victor E. Lee, and Hui Hong. 2011. "Axiomatic ranking of network role similarity". *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: 922-930. doi:10.1145/2020408.2020561

On REGE:

White, Douglas R., and Karl P. Reitz. 1983. "Graph and semigroup homomorphisms on networks of relations". *Social Networks* 5(2): 193-234. doi:10.1016/03788733(83)900254

See Also

Other methods: `method_coreness`

`method_split` *Methods for splitting a continuous score into ordered groups*

Description

These functions split a continuous score, such as a coreness score, into an ordered set of groups:

- `split_bins()` cuts the range into equal-width bins.
- `split_quantiles()` cuts at the quantiles, so each group holds a similar number of nodes.
- `split_kmeans()` clusters the scores by k-means, so the cuts fall where the scores themselves are furthest apart.

These functions are not intended to be called directly, but are called within `node_in_core()` and related functions. They are exported and listed here to provide more detailed documentation.

Usage

```
split_bins(scores, groups)

split_quantiles(scores, groups)

split_kmeans(scores, groups)
```

Arguments

`scores`	A numeric vector of scores to split.
`groups`	An integer indicating the number of groups to split into.

Value

An integer vector the length of `scores`, giving each score's group index, numbered from the lowest score upwards.

Bins

Cuts the observed range into `groups` intervals of equal width. Where the scores are unevenly spread, a bin can end up empty, so this returns the coarsest picture of the three.

Quantiles

Cuts at the quantiles of the scores, so each group holds a similar number of nodes whatever the shape of the distribution.

K-means

Clusters the scores by k-means, so the cuts fall where the scores are furthest apart rather than at fixed widths or counts.

Examples

```
split_bins(c(0, 0.1, 0.4, 0.9, 1), 3)
split_quantiles(c(0, 0.1, 0.4, 0.9, 1), 3)
split_kmeans(c(0, 0.1, 0.4, 0.9, 1), 3)
```

`motif_brokerage_net` *Motifs of network brokerage*

Description

`net_x_brokerage()` returns the Gould-Fernandez brokerage roles in a network.

Usage

```
net_x_brokerage(.data, membership, standardized = FALSE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`membership`	A character string naming an existing node attribute in the network, or a categorical vector of the same length as the number of nodes in the network where each element indicates the group membership of the corresponding node. While this may often be a vector created using `node_in_*()` functions, it can be any character vector that assigns nodes to groups or categories.
`standardized`	Logical scalar. Where `TRUE`, the counts are returned as z-scores against a null model rather than as raw counts. This is a different quantity from `normalized`, which divides by a theoretical maximum, and from `scaled`, which divides by the observed maximum: a z-score says how far the count departs from what the null model expects, so it can be negative and has no fixed range. By default `FALSE`.

Value

A `network_motif` named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

See Also

Other brokerage: `measure_broker_node`, `measure_broker_tie`, `measure_brokerage`, `member_brokerage`, `motif_brokerage_node`

Other motifs: `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_hazard`, `motif_hierarchy`, `motif_homophily`, `motif_net`, `motif_node`, `motif_path`, `motif_periods`

Examples

```
net_x_brokerage(ison_networkers, "Discipline")
```

`motif_brokerage_node` *Motifs of nodes brokerage*

Description

`node_x_brokerage()` returns the Gould-Fernandez brokerage roles played by nodes in a network.

Usage

```
node_x_brokerage(.data, membership, standardized = FALSE)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`membership`	A character string naming an existing node attribute in the network, or a categorical vector of the same length as the number of nodes in the network where each element indicates the group membership of the corresponding node. While this may often be a vector created using `node_in_*()` functions, it can be any character vector that assigns nodes to groups or categories.
`standardized`	Logical scalar. Where `TRUE`, the counts are returned as z-scores against a null model rather than as raw counts. This is a different quantity from `normalized`, which divides by a theoretical maximum, and from `scaled`, which divides by the observed maximum: a z-score says how far the count departs from what the null model expects, so it can be negative and has no fixed range. By default `FALSE`.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

References

On brokerage motifs:

Gould, Roger V., and Roberto M. Fernandez. 1989. "Structures of Mediation: A Formal Approach to Brokerage in Transaction Networks." *Sociological Methodology*, 19: 89-126. doi:10.2307/270949

Jasny, Lorien, and Mark Lubell. 2015. "Two-Mode Brokerage in Policy Networks." *Social Networks* 41:36–47. doi:10.1016/j.socnet.2014.11.005

See Also

Other brokerage: measure_broker_node, measure_broker_tie, measure_brokerage, member_brokerage, motif_brokerage_net

Other motifs: motif_brokerage_net, motif_clique, motif_composition, motif_exposure, motif_hazard, motif_hierarchy, motif_homophily, motif_net, motif_node, motif_path, motif_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_clique, motif_composition, motif_exposure, motif_node, motif_path

Examples

```
node_x_brokerage(ison_networkers, "Discipline")
```

---

motif_clique *Motifs of clique participation*

Description

`node_x_clique()` returns which maximal cliques each node belongs to.

A clique is a set of nodes every one of which is tied to every other, and it is maximal if no further node can be added without breaking that. Cliques are the strictest notion of a cohesive subgroup, and unlike the communities returned by `node_in_*()` functions they *overlap*: a node may belong to many cliques at once, or to none. That is why this returns an incidence table rather than a membership vector.

Usage

```
node_x_clique(.data, min_clique_size = 3)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`min_clique_size`	Integer, the minimum size of clique to return. By default 3, since dyads and isolates are trivially cliques. For a two-mode network, a vector of two values giving the minimum number of nodes from each mode, by default `c(3, 3)`.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

Bicliques

In a two-mode network no two nodes of the same mode are ever tied directly, so no set of them is a clique in the ordinary sense. The two-mode analogue is a *biclique*: a set of nodes from each mode such that every node of the one is tied to every node of the other. `node_x_clique()` detects these by connecting nodes that share a partner before searching, so that a biclique becomes an ordinary clique, and then keeping only those cliques with at least `min_clique_size` nodes from each mode.

Signed networks

Since a clique is a maximally cohesive subgroup, negative ties cannot contribute to one. Where the network is signed, only its positive ties are considered. Use `manynet::to_unsigned()` first to control this yourself.

References

On cliques:

Luce, R. Duncan, and Albert D. Perry. 1949. "A method of matrix analysis of group structure". *Psychometrika* 14(2): 95-116. doi:10.1007/BF02289146

See Also

Other motifs: `motif_brokerage_net`, `motif_brokerage_node`, `motif_composition`, `motif_exposure`, `motif_hazard`, `motif_hierarchy`, `motif_homophily`, `motif_net`, `motif_node`, `motif_path`, `motif_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_composition`, `motif_exposure`, `motif_node`, `motif_path`

Examples

```
node_x_clique(ison_adolescents)
node_x_clique(ison_southern_women, min_clique_size = c(3, 3))
```

`motif_composition` *Motifs of ego-network composition*

Description

These functions describe the composition of each node's ego-network, that is, what the ties and alters surrounding each node look like:

- `node_x_ties()` describes the distribution of each node's tie values, or, in a multiplex network, how its ties are spread across layers.
- `node_x_alters()` describes the composition of each node's alters on some attribute.

- `node_x_similarity()` describes how similar each node is to its alters on some attribute, or, in a two-mode network, to those it shares a node of the other mode with.

Where the corresponding `node_by_*()` measures collapse this information into a single score per node, these return the whole table, which is often what is wanted when exploring ego-networks. Each branches internally on the type of network or attribute given, so the same function serves weighted, multiplex, and two-mode networks, and categorical as well as continuous attributes.

Usage

```
node_x_ties(.data, direction = c("all", "out", "in"))

node_x_alters(.data, attribute)

node_x_similarity(.data, attribute)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`direction`	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. By default "all".
`attribute`	Name of a nodal attribute, mark, measure, or membership vector.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

Multiplex networks

`node_x_ties()` returns one column per layer, plus a Diversity column, rather than the distribution of tie values it returns otherwise. Layers are taken by name, so a network multiplexed on any attribute is covered, not only one multiplexed on `type`. Every column stays the length of the whole nodeset, so a node holding no tie in a layer scores 0 there rather than dropping out.

Tie composition

For a weighted network this returns the distribution of each node's tie values: how many ties it has, and the sum, mean, standard deviation, and quartiles of their strengths. Two nodes may have the same weighted degree while one spreads its involvement evenly and the other concentrates it in a single strong tie, and it is the spread rather than the total that distinguishes them.

For a multiplex network it instead returns one column per layer, giving each node's degree in that layer (or its strength, where the layer is itself weighted), together with `Diversity`, the index of qualitative variation across the layers. This is 0 where a node's ties all fall in a single layer, and

1 where they are spread evenly across all of them. Where the interest is in just two of the layers, `node_by_multidegree()` gives the ratio between them.

For an unweighted, uniplex network only the degree is available, so this returns that alone. Isolates have no ties to summarise and so take `NA` for the distributional columns.

In a directed network, `direction` selects whose ties are described: a node's outgoing ties, its incoming ties, or both together. Note that under `"all"` a reciprocated pair is treated as a single relationship of combined strength, so `Ties` counts a node's distinct alters rather than its arcs, while `Sum` matches its total degree.

Alter composition

Where the attribute is categorical, this returns how many of each node's alters fall into each category, weighted by tie strength where the network is weighted. Where it is continuous, this returns the sum, mean, tie-strength weighted mean, minimum, maximum, range, and standard deviation of the attribute across each node's alters.

The weighted mean differs from the mean wherever a node's ties are of unequal strength: it describes the attribute of the alters a node is most involved with, rather than of its alters as an undifferentiated set. Isolates have no alters and so take `NA`.

Any tie counts as a tie here, whatever its sign. Apply `manynet::to_unsigned()` first to consider only positive or only negative ties.

Ego-alter similarity

Where the attribute is categorical, this returns each node's own two-by-two table of whether a tie is present and whether the alter shares its category, together with the summaries built from it: the proportion of a node's ties that are to others of the same category (`PctSame`), the EI index (`EI`), which runs from -1 where all of a node's ties are internal to its own category to +1 where all are external, the odds ratio and its logarithm, and Yule's Q.

The EI index and the odds ratio answer different questions. EI describes the mix of a node's ties, and so is sensitive to how large its category is: in a small category even an indifferent node will have mostly external ties. The odds ratio and Yule's Q instead compare the ties a node made against the ties it could have made, and so are not.

Where the attribute is continuous, this returns the mean difference, mean absolute difference, and mean squared difference between a node and its alters, followed by three measures of dyadic similarity averaged over a node's alters: Zegers' coefficient, the ratio of the smaller value to the larger, and the product.

Tertius similarity

In a two-mode network no two nodes of the same mode are ever tied, so similarity to one's alters cannot be measured directly. Instead, each node is compared here with those it shares a node of the other mode with, that is, its alters at distance two. This is the tertius neighbourhood used by the `tertius()` effect in `{migraph}` and `{goldfish}`, and described in Haunss and Hollway (2023): in a discourse network, for example, the actors an actor is compared with are those making claims about the same concepts.

The same columns are returned as for a one-mode network, but read at distance two: a node's alters are those it shares some other-mode node with, however many they share, and the non-alters are the

remaining nodes of its own mode. Nodes of the other mode are neither alters nor non-alters, and so are excluded rather than counted as absent ties. Since a node's alters are always of its own mode, only that mode's values of the attribute are used; where an attribute is held by one mode alone, the other mode's nodes take NA.

References

On tertius effects:

Haunss, Sebastian, and James Hollway. 2023. "Multimodal mechanisms of political discourse dynamics and the case of Germany's nuclear energy phase-out". *Network Science* 11(2): 205-223. doi:10.1017/nws.2022.31

On the EI index:

Krackhardt, David, and Robert N. Stern. 1988. "Informal Networks and Organizational Crises: An Experimental Simulation". *Social Psychology Quarterly* 51(2): 123-140. doi:10.2307/2786835

On Yule's Q:

Yule, G. Udny. 1912. "On the Methods of Measuring Association Between Two Attributes". *Journal of the Royal Statistical Society* 75(6): 579-652. doi:10.2307/2340126

See Also

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_exposure, motif_hazard, motif_hierarchy, motif_homophily, motif_net, motif_node, motif_path, motif_periods

Other diversity: measure_assort_net, measure_assort_node, measure_diverse_net, measure_diverse_node, motif_homophily

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_exposure, motif_node, motif_path

Examples

```
node_x_ties(ison_networkers)
node_x_ties(ison_algebra)
node_x_ties(fict_marvel)
node_x_alters(ison_networkers, "Discipline")
node_x_alters(ison_networkers, "Citations")
node_x_similarity(ison_networkers, "Discipline")
node_x_similarity(ison_southern_women, "Title")
```

`motif_exposure` *Motifs of nodes exposure*

Description

`node_x_exposure()` produces a motif matrix of nodes' exposure to infection/adoption by time step.

Usage

```
node_x_exposure(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

See Also

Other diffusion: `mark_diff`, `measure_diffusion_infection`, `measure_diffusion_net`, `measure_diffusion_node`, `member_diffusion`, `motif_hazard`

Other motifs: `motif_brokerage_net`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_hazard`, `motif_hierarchy`, `motif_homophily`, `motif_net`, `motif_node`, `motif_path`, `motif_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`, `measure_central_degree`, `measure_central_eigen`, `measure_closure_node`, `measure_core`, `measure_diffusion_node`, `measure_diverse_node`, `member_brokerage`, `member_cliques`, `member_community`, `member_community_hier`, `member_community_non`, `member_components`, `member_core`, `member_diffusion`, `member_equivalence`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_node`, `motif_path`

Examples

```
node_x_exposure(play_diffusion(create_tree(12)))
```

`motif_hazard` *Motifs of network hazard*

Description

`net_x_hazard()` measures the hazard rate or instantaneous probability that nodes will adopt/become infected at that time.

Usage

```
net_x_hazard(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `network_motif` named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

Hazard rate

The hazard rate is the instantaneous probability of adoption/infection at each time point (Allison 1984). In survival analysis, hazard rate is formally defined as:

$$\lambda(t) = \lim_{h \to +0} \frac{F(t+h) - F(t)}{h} \frac{1}{1 - F(t)}$$

By approximating $h = 1$, we can rewrite the equation as

$$\lambda(t) = \frac{F(t+1) - F(t)}{1 - F(t)}$$

If we estimate $F(t)$, the probability of not having adopted the innovation in time t, from the proportion of adopters in that time, such that $F(t) \sim q_t/n$, we now have (ultimately for $t > 1$):

$$\lambda(t) = \frac{q_{t+1}/n - q_t/n}{1 - q_t/n} = \frac{q_{t+1} - q_t}{n - q_t} = \frac{q_t - q_{t-1}}{n - q_{t-1}}$$

where q_i is the number of adopters in time t, and n is the number of vertices in the graph.

The shape of the hazard rate indicates the pattern of new adopters over time. Rapid diffusion with convex cumulative adoption curves will have hazard functions that peak early and decay over time. Slow concave cumulative adoption curves will have hazard functions that are low early and rise over time. Smooth hazard curves indicate constant adoption whereas those that oscillate indicate variability in adoption behavior over time.

Source

{netdiffuseR}

References

On hazard rates:

Allison, Paul D. 1984. *Event history analysis: Regression for longitudinal event data*. London: Sage Publications. doi:10.4135/9781412984195

Wooldridge, Jeffrey M. 2010. *Econometric Analysis of Cross Section and Panel Data* (2nd ed.). Cambridge: MIT Press.

See Also

Other diffusion: mark_diff, measure_diffusion_infection, measure_diffusion_net, measure_diffusion_node, member_diffusion, motif_exposure

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_hierarchy, motif_homophily, motif_net, motif_node, motif_path, motif_periods

Examples

```
# To calculate the hazard rates at each time point
  smeg <- generate_smallworld(15, 0.025)
net_x_hazard(play_diffusion(smeg, transmissibility = 0.3))
```

---

motif_hierarchy *Motifs of network hierarchy*

Description

net_x_hierarchy() collects the measures of hierarchy into a single motif, which can be used to compare the relative hierarchy of different networks. The measures of hierarchy are:

- net_by_connectedness() measures the proportion of dyads in the network that are reachable to one another, or the degree to which network is a single component.
- net_by_efficiency() measures the Krackhardt efficiency score.
- net_by_upperbound() measures the Krackhardt (least) upper bound score.
- net_by_reciprocity() measures the proportion of ties in the network that are reciprocated, which is a measure of the degree to which the network is non-hierarchical.

Usage

```
net_x_hierarchy(.data)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().

Value

A network_motif named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

References

On hierarchy:

Krackhardt, David. 1994. Graph theoretical dimensions of informal organizations. In Carley and Prietula (eds) *Computational Organizational Theory*, Hillsdale, NJ: Lawrence Erlbaum Associates. Pp. 89-111.

Everett, Martin, and David Krackhardt. 2012. "A second look at Krackhardt's graph theoretical dimensions of informal organizations." *Social Networks*, 34: 159-163. doi:10.1016/j.socnet.2011.10.006

See Also

Other hierarchy: measure_hierarchy

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_hazard, motif_homophily, motif_net, motif_node, motif_path, motif_periods

Examples

```
net_x_hierarchy(ison_networkers)
```

---

motif_homophily *Motifs of network homophily*

Description

net_x_homophily() returns the two-by-two table from which network-level homophily is calculated, together with the summaries built from it.

Where net_by_heterophily() returns the EI index alone, this returns the counts it rests on, so that the index can be interpreted against the network's own composition.

Note that on a weighted network the two report different values. A contingency table counts ties, so net_x_homophily() treats every tie alike, whereas net_by_heterophily() sums tie weights and so gives more say to stronger ties. On unweighted networks the two agree exactly. Apply manynet::to_unweighted() first to compare them directly.

Usage

```
net_x_homophily(.data, attribute)
```

Arguments

.data	A network object of class stocnet, igraph, tbl_graph, network, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. manynet::as_stocnet().
attribute	Name of a nodal attribute, mark, measure, or membership vector.

Value

A network_motif named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

Expected EI

The EI index depends on how large the categories are, not only on how nodes choose between them. A network split into two equal groups will have a lower EI than one in which a small minority is surrounded by a large majority, even if nodes in both are equally indifferent to category.

ExpectedEI gives the EI that would be observed if ties were distributed at random across all possible pairs, holding category sizes fixed. Comparing EI against it separates the network's mixing from its composition: an EI above the expected value indicates more crossing of category boundaries than chance alone would produce, and one below it indicates less.

References

On the EI index:

Krackhardt, David, and Robert N. Stern. 1988. "Informal Networks and Organizational Crises: An Experimental Simulation". *Social Psychology Quarterly* 51(2): 123-140. doi:10.2307/2786835

See Also

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_hazard, motif_hierarchy, motif_net, motif_node, motif_path, motif_periods

Other diversity: measure_assort_net, measure_assort_node, measure_diverse_net, measure_diverse_node, motif_composition

Examples

```
net_x_homophily(ison_networkers, "Discipline")
```

`motif_net` *Motifs of network cohesion*

Description

These functions include ways to take a census of the graphlets in a network:

- `net_x_dyad()` returns a census of dyad motifs in a network.
- `net_x_triad()` returns a census of triad motifs in a network.
- `net_x_tetrad()` returns a census of tetrad motifs in a network.

See also graph classes.

Usage

```
net_x_dyad(.data)

net_x_triad(.data, object2 = NULL)

net_x_tetrad(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`object2`	A second, two-mode network object. Only `net_x_triad()` uses this, and only to take a multilevel census; see its Mixed census section.

Value

A `network_motif` named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

Multiplex networks

`net_x_triad()` takes the mixed census on a multiplex network, splitting it into layers by mode rather than by position. To census one layer alone, take it first with `manynet::to_uniplex()`. `net_x_dyad()` and `net_x_tetrad()` count every tie whatever its layer.

Dyad census

The dyad census counts the number of mutual, asymmetric, and null dyads in a network. For directed networks,

- Mutual dyads have ties in both directions

- Asymmetric dyads have a tie in one direction only
- Null dyads have no ties

Note that for undirected and two-mode networks, only mutual and null dyads are possible, as the concept of an asymmetric dyad does not apply.

Triad census

The triad census counts the number of three-node configurations in the network. The function returns a matrix with a special naming convention:

- 003: This is an empty triad; no ties
- 012: This triad includes one tie
- 102: This triad includes two ties, but they are not reciprocated
- 021D: This triad includes two ties, one of which is reciprocated, and the other is directed towards the reciprocated tie
- 021U: This triad includes two ties, one of which is reciprocated, and the other is directed away from the reciprocated tie
- 021C: This triad includes two ties, one of which is reciprocated, and the other is directed between the two non-reciprocated nodes
- 111D: This triad includes three ties, two of which are reciprocated, and the other is directed towards the reciprocated ties
- 111U: This triad includes three ties, two of which are reciprocated, and the other is directed away from the reciprocated ties
- 030T: This triad includes three ties, all of which are directed in a transitive manner (i.e. A->B, B->C, A->C)
- 030C: This triad includes three ties, all of which are directed in a cyclic manner (i.e. A->B, B->C A->C)
- 201: This triad includes three ties, all of which are reciproc ated (i.e. A<->B, B<->C, A<->C)
- 120D: This triad includes four ties, three of which are reciprocated, and the other is directed towards the reciprocated ties
- 120U: This triad includes four ties, three of which are reciprocated, and the other is directed away from the reciprocated ties
- 120C: This triad includes four ties, three of which are reciprocated, and the other is directed between the two non-reciprocated
- 210: This triad includes five ties, four of which are reciprocated, and the other is directed between the two non-reciprocated
- 300: This triad includes six ties, all of which are reciprocated

Note that for undirected and two-mode networks, only 003, 102, and 201 are possible, as the other configurations rely on the concept of directionality.

Mixed census

Where a one-mode and a two-mode network are given together, a multilevel census of the triads that span them is taken instead, after Hollway et al. (2017). Its ten motifs are labelled by how many ties join the pair of nodes at each level, so that `"21"` counts triads whose two nodes are reciprocally tied in the one-mode network and share a partner in the two-mode network.

There are two ways to ask for it. Supply the two networks as `.data` and `object2`, the one-mode network first. Or give a single multiplex network holding exactly one one-mode layer and one two-mode layer, such as `fict_marvel`, and the two layers are used in that order. A two-mode network that carries no one-mode layer is not enough, and remains unavailable.

Since a census counts configurations, only the presence of a tie counts. Weights and signs are set aside, as `igraph::triad_census()` also does.

`node_x_triad()` reports the same ten motifs for each node.

Tetrad census

The tetrad census counts the number of four-node configurations in the network. The function returns a matrix with a special naming convention:

- E4 (aka co-K4): This is an empty set of four nodes; no ties
- I4 (aka co-diamond): This is a set of four nodes with just one tie
- H4 (aka co-C4): This set of four nodes includes two non-adjacent ties
- L4 (aka co-paw): This set of four nodes includes two adjacent ties
- D4 (aka co-claw): This set of four nodes includes three adjacent ties, in the form of a triangle with one isolate
- U4 (aka P4, four-actor line): This set of four nodes includes three ties arranged in a line
- Y4 (aka claw): This set of four nodes includes three ties all adjacent to a single node
- P4 (aka paw, kite): This set of four nodes includes four ties arranged as a triangle with an extra tie hanging off of one of the nodes
- C4 (aka bifan): This is a symmetric box or 4-cycle or set of shared choices
- Z4 (aka diamond): This resembles C4 but with an extra tie cutting across the box
- X4 (aka K4): This resembles C4 but with two extra ties cutting across the box; a realisation of all possible ties

Graphs of these motifs can be shown using `plot(net_x_tetrad(ison_southern_women))`.

Source

Mixed census adapted from Alejandro Espinosa 'netmem'

References

On the dyad census:

Holland, Paul W., and Samuel Leinhardt. 1970. "A Method for Detecting Structure in Sociometric Data". *American Journal of Sociology*, 76: 492-513. doi:10.1016/B9780124424500.500286

Wasserman, Stanley, and Katherine Faust. 1994. "Social Network Analysis: Methods and Applications". Cambridge: Cambridge University Press.

On the triad census:

Davis, James A., and Samuel Leinhardt. 1967. "The Structure of Positive Interpersonal Relations in Small Groups." 55.

On the mixed census:

Hollway, James, Alessandro Lomi, Francesca Pallotti, and Christoph Stadtfeld. 2017. "Multilevel Social Spaces: The Network Dynamics of Organizational Fields." *Network Science* 5(2): 187–212. doi:10.1017/nws.2017.8

On the tetrad census:

Ortmann, Mark, and Ulrik Brandes. 2017. "Efficient Orbit-Aware Triad and Quad Census in Directed and Undirected Graphs." *Applied Network Science* 2(1):13. doi:10.1007/s41109017-00272.

McMillan, Cassie, and Diane Felmlee. 2020. "Beyond Dyads and Triads: A Comparison of Tetrads in Twenty Social Networks". *Social Psychology Quarterly* 83(4): 383-404. doi:10.1177/0190272520944151

See Also

Other cohesion: `mark_triangles`, `measure_breadth`, `measure_cohesion`, `measure_fragmentation`, `motif_node`

Other motifs: `motif_brokerage_net`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_hazard`, `motif_hierarchy`, `motif_homophily`, `motif_node`, `motif_path`, `motif_periods`

Examples

```
net_x_dyad(manynet::ison_algebra)
net_x_triad(manynet::ison_adolescents)
net_x_triad(fict_marvel)
net_x_tetrad(ison_southern_women)
```

`motif_node` *Motifs of nodes cohesion*

Description

These functions include ways to take a census of the positions of nodes in a network:

- `node_x_tie()` returns a census of the ties in a network. For directed networks, out-ties and in-ties are bound together. For multiplex networks, the various types of ties are bound together.
- `node_x_triad()` returns a census of the triad configurations nodes are embedded in.
- `node_x_tetrad()` returns a census of nodes' positions in motifs of four nodes.
- `node_x_path()` returns the shortest path lengths of each node to every other node in the network.

Usage

```
node_x_dyad(.data)

node_x_triad(.data)

node_x_tetrad(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

Tetrad census

The nodal tetrad census counts the number of four-node configurations that each node is embedded in. The function returns a matrix with a special naming convention:

- E4 (aka co-K4): This is an empty set of four nodes; no ties
- I4 (aka co-diamond): This is a set of four nodes with just one tie
- H4 (aka co-C4): This set of four nodes includes two non-adjacent ties
- L4 (aka co-paw): This set of four nodes includes two adjacent ties
- D4 (aka co-claw): This set of four nodes includes three adjacent ties, in the form of a triangle with one isolate
- U4 (aka P4, four-actor line): This set of four nodes includes three ties arranged in a line
- Y4 (aka claw): This set of four nodes includes three ties all adjacent to a single node
- P4 (aka paw, kite): This set of four nodes includes four ties arranged as a triangle with an extra tie hanging off of one of the nodes
- C4 (aka bifan): This is a symmetric box or 4-cycle or set of shared choices
- Z4 (aka diamond): This resembles C4 but with an extra tie cutting across the box
- X4 (aka K4): This resembles C4 but with two extra ties cutting across the box; a realisation of all possible ties

Graphs of these motifs can be shown using `plot(node_by_tetrad(ison_southern_women))`.

References

On the dyad census:

Holland, Paul W., and Samuel Leinhardt. 1970. "A Method for Detecting Structure in Sociometric Data". *American Journal of Sociology*, 76: 492-513. doi:10.1016/B9780124424500.500286

On the triad census:

Davis, James A., and Samuel Leinhardt. 1967. "The Structure of Positive Interpersonal Relations in Small Groups." 55.

On the tetrad census:

Ortmann, Mark, and Ulrik Brandes. 2017. "Efficient Orbit-Aware Triad and Quad Census in Directed and Undirected Graphs." *Applied Network Science* 2(1):13. doi:10.1007/s41109017-00272.

McMillan, Cassie, and Diane Felmlee. 2020. "Beyond Dyads and Triads: A Comparison of Tetrads in Twenty Social Networks". *Social Psychology Quarterly* 83(4): 383-404. doi:10.1177/0190272520944151

See Also

Other cohesion: mark_triangles, measure_breadth, measure_cohesion, measure_fragmentation, motif_net

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_hazard, motif_hierarchy, motif_homophily, motif_net, motif_path, motif_periods

Other nodal: mark_core, mark_degree, mark_diff, mark_nodes, mark_select_node, measure_assort_node, measure_broker_node, measure_brokerage, measure_central_between, measure_central_close, measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_path

Examples

```
node_x_dyad(ison_networkers)
task_eg <- to_named(to_uniplex(ison_algebra, "tasks"))
(triad_cen <- node_x_triad(task_eg))
node_x_tetrad(ison_southern_women)
```

`motif_path` *Motifs of nodes pathing*

Description

These functions include ways to take a census of the positions of nodes in a network:

- `node_x_tie()` returns a census of the ties in a network. For directed networks, out-ties and in-ties are bound together. For multiplex networks, the various types of ties are bound together.
- `node_x_path()` returns the shortest path lengths of each node to every other node in the network.

Usage

```
node_x_tie(.data)

node_x_path(.data)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.

Value

A `node_motif` matrix with one row for each node in the network and a column for each motif type, giving the count of each motif in which each node participates. It is printed as a tibble, however, to avoid greedy printing. If the network is labelled, then the node names will be in a column named `names`.

Multiplex networks

`node_x_tie()` binds the layers together, giving one block of columns per layer, whatever attribute the network is multiplexed on. Each block stays the length of the whole nodeset. To census one layer alone, take it first with `manynet::to_uniplex()`.

References

On paths:

Dijkstra, Edsger W. 1959. "A note on two problems in connexion with graphs". *Numerische Mathematik* 1, 269-71. doi:10.1007/BF01386390.

Opsahl, Tore, Filip Agneessens, and John Skvoretz. 2010. "Node centrality in weighted networks: Generalizing degree and shortest paths". *Social Networks* 32(3): 245-51. doi:10.1016/j.socnet.2010.03.006.

See Also

Other motifs: `motif_brokerage_net`, `motif_brokerage_node`, `motif_clique`, `motif_composition`, `motif_exposure`, `motif_hazard`, `motif_hierarchy`, `motif_homophily`, `motif_net`, `motif_node`, `motif_periods`

Other nodal: `mark_core`, `mark_degree`, `mark_diff`, `mark_nodes`, `mark_select_node`, `measure_assort_node`, `measure_broker_node`, `measure_brokerage`, `measure_central_between`, `measure_central_close`,

measure_central_degree, measure_central_eigen, measure_closure_node, measure_core, measure_diffusion_node, measure_diverse_node, member_brokerage, member_cliques, member_community, member_community_hier, member_community_non, member_components, member_core, member_diffusion, member_equivalence, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_node

Examples

```
task_eg <- to_named(to_uniplex(ison_algebra, "tasks"))
(tie_cen <- node_x_tie(task_eg))
node_x_path(ison_adolescents)
node_x_path(ison_southern_women)
```

---

motif_periods *Motifs of network change*

Description

These functions measure certain topological features of networks:

- `net_x_change()` measures the Hamming distance between two or more networks.
- `net_x_stability()` measures the Jaccard index of stability between two or more networks.
- `net_x_correlation()` measures the product-moment correlation between two networks.

These `net_*()` functions return a numeric vector the length of the number of networks minus one. E.g., the periods between waves.

Usage

```
net_x_change(.data, object2)

net_x_stability(.data, object2)

net_x_correlation(.data, object2)
```

Arguments

`.data`	A network object of class `stocnet`, `igraph`, `tbl_graph`, `network`, or similar. Internally any of these will be coerced to an efficient implementation. For more information on possible coercions, see e.g. `manynet::as_stocnet()`.
`object2`	A network object.

Value

A `network_motif` named numeric vector or sometimes a data frame with one row and a column for each motif type, giving the count of each motif in the network. This is printed as a tibble to avoid greedy printing of long vectors.

See Also

Other change: measure_periods

Other motifs: motif_brokerage_net, motif_brokerage_node, motif_clique, motif_composition, motif_exposure, motif_hazard, motif_hierarchy, motif_homophily, motif_net, motif_node, motif_path

Examples

```
net_x_change(ison_monks)
```

Index

∗ **betweenness**
measure_central_between, 33
measure_central_tie_between, 52
measure_centralisation_between, 26
∗ **brokerage**
measure_broker_node, 22
measure_broker_tie, 25
measure_brokerage, 21
member_brokerage, 88
motif_brokerage_net, 120
motif_brokerage_node, 121
∗ **centrality**
measure_central_between, 33
measure_central_close, 36
measure_central_degree, 43
measure_central_eigen, 46
measure_central_tie_between, 52
measure_central_tie_close, 53
measure_central_tie_degree, 54
measure_central_tie_eigen, 56
measure_centralisation_between, 26
measure_centralisation_close, 28
measure_centralisation_degree, 30
measure_centralisation_eigen, 31
∗ **change**
measure_periods, 87
motif_periods, 139
∗ **closeness**
measure_central_close, 36
measure_central_tie_close, 53
measure_centralisation_close, 28
∗ **cohesion**
mark_triangles, 13
measure_breadth, 20
measure_cohesion, 61
measure_fragmentation, 84
motif_net, 132
motif_node, 135
∗ **community**
member_community, 91
member_community_hier, 93
member_community_non, 96
∗ **core-periphery**
mark_core, 3
measure_core, 63
member_core, 102
∗ **degree**
mark_degree, 5
measure_central_degree, 43
measure_central_tie_degree, 54
measure_centralisation_degree, 30
∗ **diffusion**
mark_diff, 6
measure_diffusion_infection, 65
measure_diffusion_net, 67
measure_diffusion_node, 70
member_diffusion, 104
motif_exposure, 127
motif_hazard, 128
∗ **diversity**
measure_assort_net, 15
measure_assort_node, 18
measure_diverse_net, 72
measure_diverse_node, 75
motif_composition, 123
motif_homophily, 130
∗ **eigenvector**
measure_central_eigen, 46
measure_central_tie_eigen, 56
measure_centralisation_eigen, 31
∗ **features**
measure_features, 76
measure_fit, 80
∗ **hierarchy**
measure_hierarchy, 86
motif_hierarchy, 129
∗ **marks**
mark_core, 3

mark_degree, 5
mark_diff, 6
mark_dyads, 7
mark_nodes, 8
mark_select_node, 10
mark_select_tie, 11
mark_ties, 12
mark_triangles, 13
∗ **measures**
measure_assort_net, 15
measure_assort_node, 18
measure_breadth, 20
measure_broker_node, 22
measure_broker_tie, 25
measure_brokerage, 21
measure_central_between, 33
measure_central_close, 36
measure_central_degree, 43
measure_central_eigen, 46
measure_central_tie_between, 52
measure_central_tie_close, 53
measure_central_tie_degree, 54
measure_central_tie_eigen, 56
measure_closure, 57
measure_closure_node, 59
measure_cohesion, 61
measure_core, 63
measure_diffusion_infection, 65
measure_diffusion_net, 67
measure_diffusion_node, 70
measure_diverse_net, 72
measure_diverse_node, 75
measure_features, 76
measure_fit, 80
measure_fragmentation, 84
measure_hierarchy, 86
measure_periods, 87
∗ **memberships**
member_brokerage, 88
member_cliques, 89
member_community, 91
member_community_hier, 93
member_community_non, 96
member_components, 101
member_core, 102
member_diffusion, 104
member_equivalence, 105
∗ **methods**
method_coreness, 111
method_regularity, 117
∗ **motifs**
motif_brokerage_net, 120
motif_brokerage_node, 121
motif_clique, 122
motif_composition, 123
motif_exposure, 127
motif_hazard, 128
motif_hierarchy, 129
motif_homophily, 130
motif_net, 132
motif_node, 135
motif_path, 137
motif_periods, 139
∗ **nodal**
mark_core, 3
mark_degree, 5
mark_diff, 6
mark_nodes, 8
mark_select_node, 10
measure_assort_node, 18
measure_broker_node, 22
measure_brokerage, 21
measure_central_between, 33
measure_central_close, 36
measure_central_degree, 43
measure_central_eigen, 46
measure_closure_node, 59
measure_core, 63
measure_diffusion_node, 70
measure_diverse_node, 75
member_brokerage, 88
member_cliques, 89
member_community, 91
member_community_hier, 93
member_community_non, 96
member_components, 101
member_core, 102
member_diffusion, 104
member_equivalence, 105
motif_brokerage_node, 121
motif_clique, 122
motif_composition, 123
motif_exposure, 127
motif_node, 135
motif_path, 137
∗ **selection**

mark_select_node, 10
mark_select_tie, 11
∗ **tie**
mark_dyads, 7
mark_select_tie, 11
mark_ties, 12
mark_triangles, 13
measure_broker_tie, 25
measure_central_tie_between, 52
measure_central_tie_close, 53
measure_central_tie_degree, 54
measure_central_tie_eigen, 56

cluster_concor *(*method_cluster*)*, 110
cluster_cosine *(*method_cluster*)*, 110
cluster_hierarchical *(*method_cluster*)*, 110
coreness_correlation *(*method_coreness*)*, 111
coreness_hub *(*method_coreness*)*, 111
coreness_hub(), *103*, *113*
coreness_rich *(*method_coreness*)*, 111
coreness_rich(), *113*
coreness_transition *(*method_coreness*)*, 111

k_elbow *(*method_kselect*)*, 115
k_gap *(*method_kselect*)*, 115
k_silhouette *(*method_kselect*)*, 115
k_strict *(*method_kselect*)*, 115

manynet::as_stocnet(), *3*, *5*, *6*, *8–10*, *12–14*, *16*, *18*, *20*, *21*, *23*, *25*, *27*, *28*, *30*, *32*, *33*, *38*, *44*, *47*, *52*, *54–57*, *60*, *62*, *64*, *70*, *73*, *75*, *77*, *81*, *85–88*, *90*, *92*, *94*, *97*, *101*, *102*, *104*, *107*, *110*, *112*, *115*, *117*, *120–122*, *124*, *127*, *128*, *130–132*, *136*, *138*, *139*
manynet::is_twomode(), *78*
manynet::to_undirected(), *43*, *64*
manynet::to_uniplex(), *44*, *132*, *138*
manynet::to_unsigned(), *20*, *62*, *123*, *125*
manynet::to_unweighted(), *64*, *113*, *130*
mark_core, 3, *6–8*, *10–14*, *19*, *22*, *24*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *137*, *138*
mark_degree, *4*, 5, *7*, *8*, *10–14*, *19*, *22*, *24*, *31*, *36*, *42*, *46*, *51*, *55*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *137*, *138*
mark_diff, *4*, *6*, 6, *8*, *10–14*, *19*, *22*, *24*, *36*, *42*, *46*, *51*, *61*, *65*, *66*, *69*, *71*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *129*, *137*, *138*
mark_dyads, *4*, *6*, *7*, 7, *10–14*, *26*, *53–55*, *57*
mark_nodes, *4*, *6–8*, 8, *11–14*, *19*, *22*, *24*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *137*, *138*
mark_select_node, *4*, *6–8*, *10*, 10, *12–14*, *19*, *22*, *24*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *137*, *138*
mark_select_tie, *4*, *6–8*, *10*, *11*, 11, *13*, *14*, *26*, *53–55*, *57*
mark_ties, *4*, *6–8*, *10–12*, 12, *14*, *26*, *53–55*, *57*
mark_triangles, *4*, *6–8*, *10–13*, 13, *21*, *26*, *53–55*, *57*, *63*, *85*, *135*, *137*
measure_assort_net, 15, *19*, *21*, *22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–56*, *59*, *61*, *63*, *65*, *66*, *69*, *71*, *74*, *76*, *79*, *84*, *85*, *87*, *88*, *126*, *131*
measure_assort_node, *4*, *6*, *7*, *10*, *11*, *17*, 18, *21*, *22*, *24*, *26*, *35*, *36*, *42*, *46*, *51*, *53–56*, *59*, *61*, *63*, *65*, *66*, *69*, *71*, *72*, *74*, *76*, *79*, *84*, *85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *122*, *123*, *126*, *127*, *131*, *137*, *138*
measure_breadth, *14*, *17*, *19*, 20, *22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–56*, *59*, *61*, *63*, *65*, *66*, *69*, *71*, *74*, *76*, *79*, *84*, *85*, *87*, *88*, *135*, *137*
measure_broker_node, *4*, *6*, *7*, *10*, *11*, *17*, *19*, *21*, *22*, 22, *26*, *35*, *36*, *42*, *46*, *51*, *53–56*, *59*, *61*, *63*, *65*, *66*, *69*, *71*, *72*, *74*, *76*, *79*, *84*, *85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104*, *105*, *109*, *120*, *122*, *123*, *126*, *127*, *137*, *138*
measure_broker_tie, *8*, *12–14*, *17*, *19*, *21*, *22*, *24*, 25, *35*, *42*, *46*, *51*, *53–57*, *59*, *61*, *63*, *65*, *66*, *69*, *71*, *74*, *76*, *79*, *84*, *85*, *87–89*, *120*, *122*
measure_brokerage, *4*, *6*, *7*, *10*, *11*, *17*, *19*, *21*, 21, *24*, *26*, *35*, *36*, *42*, *46*, *51*,

53–56, *59*, *61*, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, *126, 127*, *137, 138*
measure_central_between, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, 33, *42*, *46*, *51*, *53–56*, *59*, *61*, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137, 138*
measure_central_close, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35, 36*, 36, *46*, *51*, *53–56*, *59*, *61*, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137, 138*
measure_central_degree, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35, 36*, *42*, 43, *51*, *53–57*, *59*, *61*, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
measure_central_eigen, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35, 36*, *42*, *46*, 46, *53–57*, *59*, *61*, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
measure_central_tie_between, *8*, *12–14*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35*, *42*, *46*, *51*, 52, *54–57*, *59*, *61*, *63*, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*
measure_central_tie_close, *8*, *12–14*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35*, *42*, *46*, *51*, *53*, 53, *55–57*, *59*, *61*, *63*, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*
measure_central_tie_degree, *6*, *8*, *12–14*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35*, *42*, *46*, *51*, *53, 54*, 54, *56, 57*, *59*, *61*, *63*, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*
measure_central_tie_eigen, *8*, *12–14*, *17*, *19*, *21, 22*, *24*, *26, 27*, *29*, *31, 32*, *35*, *42*, *46*, *51*, *53–55*, 56, *59*, *61*, *63*, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*
measure_centralisation_between, 26, *29*, *31, 32*, *35*, *42*, *46*, *51*, *53–56*
measure_centralisation_close, *27*, 28, *31, 32*, *35*, *42*, *46*, *51*, *53–56*
measure_centralisation_degree, *6*, *27*, *29*, 30, *32*, *35*, *42*, *46*, *51*, *53–56*
measure_centralisation_eigen, *27*, *29*, *31*, 31, *35*, *42*, *46*, *51*, *53–56*
measure_closure, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, 57, *61*, *63*, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*
measure_closure_node, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26*, *35, 36*, *42*, *46*, *51*, *53–55*, *57*, *59*, 59, *63*, *65, 66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
measure_cohesion, *14*, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, 61, *65, 66*, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*, *135*, *137*
measure_core, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26*, *35, 36*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, 63, *66*, *69*, *71, 72*, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
measure_diffusion_infection, *7*, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65*, 65, *69*, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*, *105*, *127*, *129*
measure_diffusion_net, *7*, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, 67, *71*, *74*, *76*, *79*, *84, 85*, *87, 88*, *105*, *127*, *129*
measure_diffusion_node, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24*, *26*, *35, 36*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, 70, *74*, *76*, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *129*, *137*, *139*
measure_diverse_net, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, 72, *76*, *79*, *84, 85*, *87, 88*, *126*, *131*

measure_diverse_node, *4*, *6, 7*, *10, 11*, *17*, *19*, *21, 22*, *24–26*, *35, 36*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, 75, *79*, *84, 85*, *87–89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *131*, *137*, *139*
measure_features, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, *76*, 76, *80*, *84, 85*, *87, 88*
measure_fit, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, *76*, *79*, 80, *85*, *87, 88*
measure_fragmentation, *14*, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, *76*, *79*, *84*, 84, *87, 88*, *135*, *137*
measure_hierarchy, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, *76*, *79*, *84, 85*, 86, *88*, *130*
measure_periods, *17*, *19*, *21, 22*, *24*, *26*, *35*, *42*, *46*, *51*, *53–55*, *57*, *59*, *61*, *63*, *65, 66*, *69*, *72*, *74*, *76*, *79*, *84, 85*, *87*, 87, *140*
member_brokerage, *4*, *6, 7*, *10, 11*, *19*, *22*, *24–26*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, 88, *91*, *93*, *95*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, *126, 127*, *137*, *139*
member_cliques, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, 89, *93*, *95*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_community, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, 91, *95, 96*, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_community_hier, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, 93, *100*, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_community_non, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95, 96*, 96, *102*, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_components, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, 101, *104, 105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_core, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, 102, *105*, *109*, *122, 123*, *126, 127*, *137*, *139*
member_diffusion, *4*, *6, 7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65, 66*, *69*, *71, 72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104*, 104, *109*, *122, 123*, *126, 127*, *129*, *137*, *139*
member_equivalence, *5–7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *95*, *100*, *102*, *104, 105*, 105, *122, 123*, *126, 127*, *137*, *139*
method_cluster, 110
method_coreness, *4*, *64*, *81*, *103*, 111, *118*
method_kselect, 115
method_regularity, *114*, 117
method_split, *102*, 119
mode_by_betweenness *(*measure_centralisation_between*)*, 26
mode_by_closeness *(*measure_centralisation_close*)*, 28
mode_by_degree *(*measure_centralisation_degree*)*, 30
mode_by_eigenvector *(*measure_centralisation_eigen*)*, 31
mode_by_indegree *(*measure_centralisation_degree*)*, 30
mode_by_outdegree *(*measure_centralisation_degree*)*, 30
motif_brokerage_net, *22*, *24*, *26*, *89*, 120, *122, 123*, *126, 127*, *129–131*, *135*, *137, 138*, *140*
motif_brokerage_node, *5–7*, *10, 11*, *19*, *22*, *24–26*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *96*, *100*, *102*, *104, 105*, *109*, *120*, 121, *123*, *126, 127*, *129–131*, *135*, *137–140*
motif_clique, *5–7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *96*,

100, *102*, *104, 105*, *109*, *120*, *122*, 122, *126, 127*, *129–131*, *135*, *137–140*
motif_composition, *5–7*, *10, 11*, *17*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *74*, *76*, *89*, *91*, *93*, *96*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, 123, *127*, *129–131*, *135*, *137–140*
motif_exposure, *5–7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65, 66*, *69*, *71, 72*, *76*, *89*, *91*, *93*, *96*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, *126*, 127, *129–131*, *135*, *137–140*
motif_hazard, *7*, *66*, *69*, *71*, *105*, *120*, *122, 123*, *126, 127*, 128, *130, 131*, *135*, *137, 138*, *140*
motif_hierarchy, *87*, *120*, *122, 123*, *126, 127*, *129*, 129, *131*, *135*, *137, 138*, *140*
motif_homophily, *17*, *19*, *74*, *76*, *120*, *122, 123*, *126, 127*, *129, 130*, 130, *135*, *137, 138*, *140*
motif_net, *14*, *21*, *63*, *85*, *120*, *122, 123*, *126, 127*, *129–131*, 132, *137, 138*, *140*
motif_node, *5–7*, *10, 11*, *14*, *19*, *21, 22*, *25*, *36*, *42*, *46*, *51*, *61*, *63*, *65*, *72*, *76*, *85*, *89*, *91*, *93*, *96*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, *126, 127*, *129–131*, *135*, 135, *138–140*
motif_path, *5–7*, *10, 11*, *19*, *22*, *25*, *36*, *42*, *46*, *51*, *61*, *65*, *72*, *76*, *89*, *91*, *93*, *96*, *100*, *102*, *104, 105*, *109*, *120*, *122, 123*, *126, 127*, *129–131*, *135*, *137*, 137, *140*
motif_periods, *88*, *120*, *122, 123*, *126, 127*, *129–131*, *135*, *137, 138*, 139

net_by_adhesion *(*measure_fragmentation*)*, 84
net_by_assortativity *(*measure_assort_net*)*, 15
net_by_balance *(*measure_features*)*, 76
net_by_betweenness *(*measure_centralisation_between*)*, 26
net_by_bipartivity *(*measure_features*)*, 76
net_by_bipartivity(), *48*
net_by_closeness *(*measure_centralisation_close*)*, 28
net_by_cohesion *(*measure_fragmentation*)*, 84
net_by_compactness *(*measure_cohesion*)*, 61
net_by_components *(*measure_cohesion*)*, 61
net_by_congruency *(*measure_closure*)*, 57
net_by_connectedness *(*measure_hierarchy*)*, 86
net_by_connectedness(), *62*
net_by_core *(*measure_fit*)*, 80
net_by_cyclicality *(*measure_closure*)*, 57
net_by_decay *(*measure_centralisation_close*)*, 28
net_by_degree *(*measure_centralisation_degree*)*, 30
net_by_density *(*measure_cohesion*)*, 61
net_by_diameter *(*measure_breadth*)*, 20
net_by_diversity *(*measure_diverse_net*)*, 72
net_by_efficiency *(*measure_hierarchy*)*, 86
net_by_efficiency(), *63*
net_by_eigenvector *(*measure_centralisation_eigen*)*, 31
net_by_equivalency *(*measure_closure*)*, 57
net_by_equivalency(), *79*
net_by_factions *(*measure_fit*)*, 80
net_by_factions(), *83*
net_by_harmonic *(*measure_centralisation_close*)*, 28
net_by_heterophily *(*measure_assort_net*)*, 15
net_by_heterophily(), *130*
net_by_homophily *(*measure_assort_net*)*, 15
net_by_immunity *(*measure_diffusion_net*)*, 67
net_by_inconsistency *(*measure_fit*)*, 80
net_by_inconsistency(), *108, 109*
net_by_indegree *(*measure_centralisation_degree*)*, 30

net_by_independence (measure_cohesion), 61
net_by_infection_complete (measure_diffusion_infection), 65
net_by_infection_peak (measure_diffusion_infection), 65
net_by_infection_total (measure_diffusion_infection), 65
net_by_integration (measure_centralisation_close), 28
net_by_length (measure_breadth), 20
net_by_modularity (measure_fit), 80
net_by_outdegree (measure_centralisation_degree), 30
net_by_reach (measure_centralisation_close), 28
net_by_reciprocity (measure_closure), 57
net_by_recovery (measure_diffusion_net), 67
net_by_reproduction (measure_diffusion_net), 67
net_by_richclub (measure_features), 76
net_by_richclub(), *113*
net_by_richness (measure_diverse_net), 72
net_by_scalefree (measure_features), 76
net_by_smallworld (measure_features), 76
net_by_spatial (measure_assort_net), 15
net_by_strength (measure_fragmentation), 84
net_by_toughness (measure_fragmentation), 84
net_by_transitivity (measure_closure), 57
net_by_transitivity(), *79*
net_by_transmissibility (measure_diffusion_net), 67
net_by_upperbound (measure_hierarchy), 86
net_by_waves (measure_periods), 87
net_x_brokerage (motif_brokerage_net), 120
net_x_change (motif_periods), 139
net_x_correlation (motif_periods), 139
net_x_dyad (motif_net), 132
net_x_hazard (motif_hazard), 128
net_x_hierarchy (motif_hierarchy), 129
net_x_homophily (motif_homophily), 130
net_x_stability (motif_periods), 139
net_x_tetrad (motif_net), 132
net_x_triad (motif_net), 132
node_by_adopt_exposure (measure_diffusion_node), 70
node_by_adopt_recovery (measure_diffusion_node), 70
node_by_adopt_threshold (measure_diffusion_node), 70
node_by_adopt_time (measure_diffusion_node), 70
node_by_alpha (measure_central_eigen), 46
node_by_authority (measure_central_eigen), 46
node_by_authority(), *114*
node_by_betweenness (measure_central_between), 33
node_by_bridges (measure_broker_node), 22
node_by_brokering_activity (measure_brokerage), 21
node_by_brokering_exclusivity (measure_brokerage), 21
node_by_closeness (measure_central_close), 36
node_by_constraint (measure_broker_node), 22
node_by_core (measure_core), 63
node_by_core(), *111*
node_by_decay (measure_central_close), 36
node_by_deg (measure_central_degree), 43
node_by_degree (measure_central_degree), 43
node_by_distance (measure_central_close), 36
node_by_diversity (measure_diverse_node), 75
node_by_eccentricity (measure_central_close), 36
node_by_efficiency

(measure_broker_node), 22
node_by_efficiency(), *63*
node_by_effsize (measure_broker_node), 22
node_by_eigenvector (measure_central_eigen), 46
node_by_equivalency (measure_closure_node), 59
node_by_flow (measure_central_between), 33
node_by_harmonic (measure_central_close), 36
node_by_harmonic(), *62*
node_by_heterophily (measure_assort_node), 18
node_by_hierarchy (measure_broker_node), 22
node_by_homophily (measure_assort_node), 18
node_by_hub (measure_central_eigen), 46
node_by_hub(), *114*
node_by_indegree (measure_central_degree), 43
node_by_induced (measure_central_between), 33
node_by_induced(), *41*
node_by_information (measure_central_close), 36
node_by_integration (measure_central_close), 36
node_by_kcoreness (measure_core), 63
node_by_leverage (measure_central_degree), 43
node_by_multidegree (measure_central_degree), 43
node_by_multidegree(), *125*
node_by_neighbours_degree (measure_broker_node), 22
node_by_outdegree (measure_central_degree), 43
node_by_pagerank (measure_central_eigen), 46
node_by_posneg (measure_central_eigen), 46
node_by_power (measure_central_eigen), 46
node_by_radiality (measure_central_close), 36
node_by_randomwalk (measure_central_close), 36
node_by_reach (measure_central_close), 36
node_by_reciprocity (measure_closure_node), 59
node_by_redundancy (measure_broker_node), 22
node_by_richness (measure_diverse_node), 75
node_by_stress (measure_central_between), 33
node_by_subgraph (measure_central_eigen), 46
node_by_subgraph(), *78*
node_by_transitivity (measure_closure_node), 59
node_by_vitality (measure_central_close), 36
node_by_vitality(), *34*
node_in_adopter (member_diffusion), 104
node_in_automorphic (member_equivalence), 105
node_in_betweenness (member_community_hier), 93
node_in_betweenness(), *52*
node_in_block (member_equivalence), 105
node_in_brokering (member_brokerage), 88
node_in_community (member_community), 91
node_in_component (member_components), 101
node_in_core (member_core), 102
node_in_core(), *111*
node_in_eigen (member_community_hier), 93
node_in_equivalence (member_equivalence), 105
node_in_fluid (member_community_non), 96
node_in_greedy (member_community_hier), 93
node_in_infomap (member_community_non), 96
node_in_labels (member_community_non), 96
node_in_leiden (member_community_non), 96
node_in_louvain (member_community_non), 96

node_in_motif (member_equivalence), 105
node_in_optimal (member_community_non), 96
node_in_partition (member_community_non), 96
node_in_partition(), *80*
node_in_regular (member_equivalence), 105
node_in_regular(), *117*
node_in_roulette (member_cliques), 89
node_in_spinglass (member_community_non), 96
node_in_structural (member_equivalence), 105
node_in_walktrap (member_community_hier), 93
node_is_core (mark_core), 3
node_is_core(), *82*, *111*
node_is_cutpoint (mark_nodes), 8
node_is_exposed (mark_diff), 6
node_is_fold (mark_nodes), 8
node_is_independent (mark_nodes), 8
node_is_infected (mark_diff), 6
node_is_isolate (mark_degree), 5
node_is_latent (mark_diff), 6
node_is_max (mark_select_node), 10
node_is_mean (mark_select_node), 10
node_is_mentor (mark_nodes), 8
node_is_min (mark_select_node), 10
node_is_neighbor (mark_nodes), 8
node_is_pendant (mark_degree), 5
node_is_random (mark_select_node), 10
node_is_recovered (mark_diff), 6
node_is_universal (mark_degree), 5
node_x_alters (motif_composition), 123
node_x_brokerage (motif_brokerage_node), 121
node_x_clique (motif_clique), 122
node_x_dyad (motif_node), 135
node_x_exposure (motif_exposure), 127
node_x_path (motif_path), 137
node_x_similarity (motif_composition), 123
node_x_tetrad (motif_node), 135
node_x_tie (motif_path), 137
node_x_ties (motif_composition), 123
node_x_triad (motif_node), 135

regularity_rege (method_regularity), 117
regularity_rege(), *107*
regularity_rolesim (method_regularity), 117
regularity_rolesim(), *107*

split_bins (method_split), 119
split_kmeans (method_split), 119
split_quantiles (method_split), 119

tie_by_betweenness (measure_central_tie_between), 52
tie_by_betweenness(), *33*
tie_by_closeness (measure_central_tie_close), 53
tie_by_cohesion (measure_broker_tie), 25
tie_by_degree (measure_central_tie_degree), 54
tie_by_eigenvector (measure_central_tie_eigen), 56
tie_is_bridge (mark_ties), 12
tie_is_cyclical (mark_triangles), 13
tie_is_feedback (mark_ties), 12
tie_is_imbalanced (mark_triangles), 13
tie_is_loop (mark_ties), 12
tie_is_max (mark_select_tie), 11
tie_is_min (mark_select_tie), 11
tie_is_multiple (mark_dyads), 7
tie_is_path (mark_ties), 12
tie_is_random (mark_select_tie), 11
tie_is_reciprocated (mark_dyads), 7
tie_is_simmelian (mark_triangles), 13
tie_is_transitive (mark_triangles), 13
tie_is_triangular (mark_triangles), 13
tie_is_triplet (mark_triangles), 13
to_undirected(), *26*, *28*, *30*, *32*, *33*, *37*, *47*, *52*, *53*, *55*, *56*