

# Package ‘migraph’

March 7, 2024

**Title** Many Network Measures, Motifs, Members, and Models

**Version** 1.3.4

**Date** 2024-03-07

**Description** A set of tools for analysing multimodal networks.

It includes functions for measuring centrality, centralization, cohesion, closure, constraint and diversity, as well as for network block-modelling, regression, and diffusion models.

The package is released as a complement to 'Multimodal Political Networks' (2021, ISBN:9781108985000), and includes various datasets used in the book.

Built on the 'manynet' package, all functions operate with matrices, edge lists, and 'igraph', 'network', and 'tidygraph' objects, and on one-mode, two-mode (bipartite), and sometimes three-mode networks.

**URL** <https://stocnet.github.io/migraph/>

**BugReports** <https://github.com/stocnet/migraph/issues>

**License** MIT + file LICENSE

**Language** en-GB

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.1

**Depends** R (>= 3.6.0), manynet

**Imports** dplyr (>= 1.1.0), generics, ggplot2, igraph (>= 1.5.0), network, future, furr, pillar, purrr, rlang, sna, tidygraph, tidyr

**Suggests** concaveman, covr, ggdendro, minMSE, oaqc, roxygen2, rsconnect, testthat, xml2

**NeedsCompilation** no

**Author** James Hollway [cre, aut, ctb] (IHEID,

<https://orcid.org/0000-0002-8361-9647>),

Henrique Sposito [ctb] (IHEID, <https://orcid.org/0000-0003-3420-6085>),

Jael Tan [ctb] (IHEID, <https://orcid.org/0000-0002-6234-9764>),

Bernhard Bieri [ctb] (<https://orcid.org/0000-0001-5943-9059>)

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

**Repository** CRAN

**Date/Publication** 2024-03-07 11:50:02 UTC

## R topics documented:

between_centrality . . . . .	3
brokerage_census . . . . .	4
cliques . . . . .	6
close_centrality . . . . .	7
closure . . . . .	9
cluster . . . . .	11
cohesion . . . . .	12
community . . . . .	13
components . . . . .	17
core . . . . .	19
degree_centrality . . . . .	20
eigenv_centrality . . . . .	23
equivalence . . . . .	25
features . . . . .	27
heterogeneity . . . . .	30
hierarchy . . . . .	33
holes . . . . .	34
kselect . . . . .	36
mpn_bristol . . . . .	37
mpn_cow . . . . .	38
mpn_elite_mex . . . . .	40
mpn_elite_usa . . . . .	42
mpn_evs . . . . .	44
mpn_ryanair . . . . .	45
mpn_senate112 . . . . .	46
network_census . . . . .	48
net_diffusion . . . . .	49
node_census . . . . .	52
node_diffusion . . . . .	54
over . . . . .	57
periods . . . . .	58
regression . . . . .	59
tests . . . . .	61

**Index**

**64**

---

between centrality      *Measures of betweenness-like centrality and centralisation*

---

## Description

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

- `node_betweenness()` measures the betweenness centralities of nodes in a network.
- `node_induced()` measures the induced betweenness centralities of nodes in a network.
- `node_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.
- `tie_betweenness()` measures the number of shortest paths going through a tie.
- `network_betweenness()` measures the betweenness centralization for 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

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

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

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

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

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

## Arguments

- `.data`      An object of a `{manynet}`-consistent class:
- `matrix` (adjacency or incidence) from `{base}` R
  - `edgelist`, a data frame from `{base}` R or tibble from `{tibble}`
  - `igraph`, from the `{igraph}` package
  - `network`, from the `{network}` package
  - `tbl_graph`, from the `{tidygraph}` package

normalized	Logical scalar, whether the centrality scores are normalized. Different denominators are used depending on whether the object is one-mode or two-mode, the type of centrality, and other arguments.
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.
direction	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. For two-mode networks, "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.

### Value

A numeric vector giving the betweenness centrality measure of each node.

### References

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

### See Also

Other centrality: [close\\_centrality](#), [degree\\_centrality](#), [eigenv\\_centrality](#)

Other measures: [close\\_centrality](#), [closure](#), [cohesion\(\)](#), [degree\\_centrality](#), [eigenv\\_centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

### Examples

```
node_betweenness(mpn_elite_mex)
node_betweenness(ison_southern_women)
node_induced(mpn_elite_mex)
(tb <- tie_betweenness(ison_adolescents))
plot(tb)
#ison_adolescents %>% mutate_ties(weight = tb) %>%
#  autographr()
network_betweenness(ison_southern_women, direction = "in")
```

---

brokerage\_census

*Censuses of brokerage motifs*

---

### Description

These functions include ways to take a census of the brokerage positions of nodes in a network:

- `node_brokerage_census()` returns the Gould-Fernandez brokerage roles played by nodes in a network.
- `network_brokerage_census()` returns the Gould-Fernandez brokerage roles in a network.

**Usage**

```
node_brokerage_census(.data, membership, standardized = FALSE)
network_brokerage_census(.data, membership, standardized = FALSE)
node_brokering_activity(.data, membership)
node_brokering_exclusivity(.data, membership)
node_brokering(.data, membership)
```

**Arguments**

.data	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
membership	A vector of partition membership as integers.
standardized	Whether the score should be standardized into a z-score indicating how many standard deviations above or below the average the score lies.

**References**

Gould, R.V. and Fernandez, R.M. 1989. "Structures of Mediation: A Formal Approach to Brokerage in Transaction Networks." *Sociological Methodology*, 19: 89-126.

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.

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 motifs: [network\\_census](#), [node\\_census](#)

**Examples**

```
node_brokerage_census(manynet::ison_networkers, "Discipline")
network_brokerage_census(manynet::ison_networkers, "Discipline")
node_brokering_exclusivity(ison_networkers, "Discipline")
```

cliques

*Clique partitioning algorithms***Description**

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

- `node_roulette()` assigns nodes to maximally diverse groups.

**Usage**

```
node_roulette(.data, num_groups, group_size, times = NULL)
```

**Arguments**

<code>.data</code>	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
<code>num_groups</code>	An integer indicating the number of groups desired.
<code>group_size</code>	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.
<code>times</code>	An integer of the number of search iterations the algorithm should complete. By default this is the number of nodes in the network multiplied by the number of groups. This heuristic may be insufficient for small networks and numbers of groups, and burdensome for large networks and numbers of groups, 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, will take place irrespective of whether it improves the objective function.

**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

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: [community](#), [components\(\)](#), [core](#), [equivalence](#)

---

close\_centrality

*Measures of closeness-like centrality and centralisation*

---

## Description

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

- `node_closeness()` measures the closeness centrality of nodes in a network.
- `node_reach()` measures nodes’ reach centrality, or how many nodes they can reach within  $k$  steps.
- `node_harmonic()` measures nodes’ harmonic centrality or valued centrality, which is thought to behave better than reach centrality for disconnected networks.
- `node_information()` measures nodes’ information centrality or current-flow closeness centrality.
- `tie_closeness()` measures the closeness of each tie to other ties in the network.
- `network_closeness()` measures a network’s closeness centralization.
- `network_reach()` measures a network’s reach centralization.
- `network_harmonic()` measures a network’s harmonic 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.

**Usage**

```

node_closeness(.data, normalized = TRUE, direction = "out", cutoff = NULL)

node_reach(.data, normalized = TRUE, k = 2)

node_harmonic(.data, normalized = TRUE, k = -1)

node_information(.data, normalized = TRUE)

tie_closeness(.data, normalized = TRUE)

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

network_reach(.data, normalized = TRUE, k = 2)

network_harmonic(.data, normalized = TRUE, k = 2)

```

**Arguments**

.data	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
normalized	Logical scalar, whether the centrality scores are normalized. Different denominators are used depending on whether the object is one-mode or two-mode, the type of centrality, and other arguments.
direction	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. For two-mode networks, "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.
cutoff	Maximum path length to use during calculations.
k	Integer of steps out to calculate reach

**References**

- Marchiori, M, and V Latora. 2000. "Harmony in the small-world". *Physica A* 285: 539-546.
- Dekker, Anthony. 2005. "Conceptual distance in social network analysis". *Journal of Social Structure* 6(3).

**See Also**

Other centrality: [between\\_centrality](#), [degree\\_centrality](#), [eigenv\\_centrality](#)



Other measures: [between centrality](#), [closure](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

### Examples

```
node_closeness(mpn_elite_mex)
node_closeness(ison_southern_women)
node_reach(ison_adolescents)
(ec <- tie_closeness(ison_adolescents))
plot(ec)
#ison_adolescents %>%
# activate(edges) %>% mutate(weight = ec) %>%
# autographr()
network_closeness(ison_southern_women, direction = "in")
```

---

closure

*Measures of network closure*

---

### Description

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

- `network_reciprocity()` measures reciprocity in a (usually directed) network.
- `node_reciprocity()` measures nodes' reciprocity.
- `network_transitivity()` measures transitivity in a network.
- `node_transitivity()` measures nodes' transitivity.
- `network_equivalency()` measures equivalence or reinforcement in a (usually two-mode) network.
- `network_congruency()` measures congruency across two two-mode networks.

### Usage

```
network_reciprocity(.data, method = "default")

node_reciprocity(.data)

network_transitivity(.data)

node_transitivity(.data)

network_equivalency(.data)

network_congruency(.data, object2)
```

**Arguments**

<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• <code>matrix</code> (adjacency or incidence) from <code>{base}</code> R</li> <li>• <code>edgelist</code>, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• <code>igraph</code>, from the <code>{igraph}</code> package</li> <li>• <code>network</code>, from the <code>{network}</code> package</li> <li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li> </ul>
<code>method</code>	For reciprocity, either <code>default</code> or <code>ratio</code> . See <code>?igraph::reciprocity</code>
<code>object2</code>	Optionally, a second (two-mode) matrix, <code>igraph</code> , or <code>tidygraph</code>

**Details**

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

For two-mode networks, `network_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, `network_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.

**Equivalency**

The `network_equivalency()` function calculates the Robins and Alexander (2004) clustering coefficient for two-mode networks. Note that for weighted two-mode networks, the result is divided by the average tie weight.

**References**

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.

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: [between centrality](#), [close centrality](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

**Examples**

```
network_reciprocity(ison_southern_women)
node_reciprocity(to_unweighted(ison_networkers))
network_transitivity(ison_adolescents)
node_transitivity(ison_adolescents)
network_equivalency(ison_southern_women)
```

---

cluster

*Methods for equivalence clustering*


---

### Description

These functions are used to cluster some 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).

These functions are not intended to be called directly, but are called within `node_equivalence()` and related functions. They are exported and listed here to provide more detailed documentation.

### Usage

```
cluster_hierarchical(census, distance)
```

```
cluster_concor(.data, census)
```

### Arguments

census	A matrix returned by a <code>node*_census()</code> function.
distance	Character string indicating which distance metric to pass on to <code>stats::dist</code> . By default "euclidean", but other options include "maximum", "manhattan", "canberra", "binary", and "minkowski". Fewer, identifiable letters, e.g. "e" for Euclidean, is sufficient.
.data	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>

### CONCOR

First a matrix of Pearson correlation coefficients between each pair of nodes profiles in the given 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

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:10.1016/00222496(75)900280.

---

cohesion

*Measures of network cohesion or connectedness*

---

## Description

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

- `network_density()` measures the ratio of ties to the number of possible ties.
- `network_components()` measures the number of (strong) components in the network.
- `network_cohesion()` measures the minimum number of nodes to remove from the network needed to increase the number of components.
- `network_adhesion()` measures the minimum number of ties to remove from the network needed to increase the number of components.
- `network_diameter()` measures the maximum path length in the network.
- `network_length()` measures the average path length in the network.

## Usage

`network_density(.data)`

`network_components(.data)`

`network_cohesion(.data)`

`network_adhesion(.data)`

`network_diameter(.data)`

`network_length(.data)`

## Arguments

`.data`

An object of a {manynet}-consistent class:

- `matrix` (adjacency or incidence) from {base} R
- `edgelist`, a data frame from {base} R or tibble from {tibble}
- `igraph`, from the {igraph} package
- `network`, from the {network} package
- `tbl_graph`, from the {tidygraph} package

**Cohesion**

To get the 'weak' components of a directed graph, please use `manynet::to_undirected()` first.

**References**

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.

**See Also**

Other measures: [between\\_centrality](#), [close\\_centrality](#), [closure](#), [degree\\_centrality](#), [eigenv\\_centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

**Examples**

```
network_density(mpn_elite_mex)
network_density(mpn_elite_usa_advice)
  network_components(mpn_ryanair)
  network_components(manynet::to_undirected(mpn_ryanair))
network_cohesion(manynet::ison_marvel_relationships)
network_cohesion(manynet::to_giant(manynet::ison_marvel_relationships))
network_adhesion(manynet::ison_marvel_relationships)
network_adhesion(manynet::to_giant(manynet::ison_marvel_relationships))
network_diameter(manynet::ison_marvel_relationships)
network_diameter(manynet::to_giant(manynet::ison_marvel_relationships))
network_length(manynet::ison_marvel_relationships)
network_length(manynet::to_giant(manynet::ison_marvel_relationships))
```

---

community

*Community partitioning algorithms*


---

**Description**

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

- `node_optimal()` is a problem-solving algorithm that seeks to maximise modularity over all possible partitions.
- `node_kernaghinlin()` is a greedy, iterative, deterministic partitioning algorithm that results in two equally-sized communities.
- `node_edge_betweenness()` is a hierarchical, decomposition algorithm where edges are removed in decreasing order of the number of shortest paths passing through the edge.
- `node_fast_greedy()` is a hierarchical, agglomerative algorithm, that tries to optimize modularity in a greedy manner.
- `node_leading_eigen()` is a top-down, hierarchical algorithm.
- `node_walktrap()` is a hierarchical, agglomerative algorithm based on random walks.
- `node_infomap()` is a hierarchical algorithm based on the information in random walks.

- `node_springlass()` is a greedy, iterative, probabilistic algorithm, based on analogy to model from statistical physics.
- `node_fluid()` is a propagation-based partitioning algorithm, based on analogy to model from fluid dynamics.
- `node_louvain()` is an agglomerative multilevel algorithm that seeks to maximise modularity over all possible partitions.
- `node_leiden()` is an agglomerative multilevel algorithm that seeks to maximise the Constant Potts Model over all possible partitions.

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_optimal(.data)

node_kernighanlin(.data)

node_edge_betweenness(.data)

node_fast_greedy(.data)

node_leading_eigen(.data)

node_walktrap(.data, times = 50)

node_infomap(.data, times = 50)

node_springlass(.data, max_k = 200, resolution = 1)

node_fluid(.data)

node_louvain(.data, resolution = 1)

node_leiden(.data, resolution = 1)
```

### Arguments

<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• <code>igraph</code>, from the <code>{igraph}</code> package</li> <li>• <code>network</code>, from the <code>{network}</code> package</li> <li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li> </ul>
<code>times</code>	Integer indicating number of simulations/walks used. By default, <code>times=50</code> .
<code>max_k</code>	Integer constant, the number of spins to use as an upper limit of communities to be found. Some sets can be empty at the end.

**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.

### **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.

### **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.

### **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

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.

### 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  $j$ ,  $\gamma$  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.

### References

- 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.
- 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
- Newman, M, and M Girvan. 2004. "Finding and evaluating community structure in networks." *Physical Review E* 69: 026113.
- Clauset, A, MEJ Newman, MEJ and C Moore. "Finding community structure in very large networks."



- Newman, MEJ. 2006. "Finding community structure using the eigenvectors of matrices" *Physical Review E* 74:036104.
- Pons, Pascal, and Matthieu Latapy "Computing communities in large networks using random walks".
- 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
- Reichardt, Jorg, and Stefan Bornholdt. 2006. "Statistical Mechanics of Community Detection" *Physical Review E*, 74(1): 016110–14. doi:10.1073/pnas.0605965104
- Traag, VA, and Jeroen Bruggeman. 2008. "Community detection in networks with positive and negative links".
- Parés F, Gasulla DG, et. al. 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
- Blondel, Vincent, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre. 2008. "Fast unfolding of communities in large networks", *J. Stat. Mech.* P10008.
- Traag, V. A., L Waltman, and NJ van Eck. 2019. "From Louvain to Leiden: guaranteeing well-connected communities", *Scientific Reports*, 9(1):5233. doi:10.1038/s4159801941695z

### See Also

Other memberships: [cliques](#), [components\(\)](#), [core](#), [equivalence](#)

### Examples

```
node_optimal(ison_adolescents)
node_kernighanlin(ison_adolescents)
node_kernighanlin(ison_southern_women)
node_edge_betweenness(ison_adolescents)
node_fast_greedy(ison_adolescents)
node_leading_eigen(ison_adolescents)
node_walktrap(ison_adolescents)
node_infomap(ison_adolescents)
node_spinglass(ison_adolescents)
node_fluid(ison_adolescents)
node_louvain(ison_adolescents)
node_leiden(ison_adolescents)
```

## Description

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

- `node_components()` assigns nodes' component membership using edge direction where available.
- `node_weak_components()` assigns nodes' component membership ignoring edge direction.
- `node_strong_components()` assigns nodes' component membership based on edge direction.
- `node_roulette()`

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.

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.

## Usage

```
node_components(.data)
```

```
node_weak_components(.data)
```

```
node_strong_components(.data)
```

## Arguments

- |                    |  |
|--------------------|--|
| <code>.data</code> | <p>An object of a <code>{manynet}</code>-consistent class:</p> <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• <code>igraph</code>, from the <code>{igraph}</code> package</li> <li>• <code>network</code>, from the <code>{network}</code> package</li> <li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li> </ul> |
|--------------------|--|

## See Also

Other memberships: [cliques](#), [community](#), [core](#), [equivalence](#)

## Examples

```
node_components(mpn_bristol)
```

---

 core

*Core-periphery clustering algorithms*


---

### Description

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

- `node_core()` assigns nodes to either the core or periphery.
- `node_coreness()` assigns nodes to their level of k-core-ness.

### Usage

```
node_core(.data, method = c("degree", "eigenvector"))
```

```
node_coreness(.data)
```

### Arguments

<code>.data</code>	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
<code>method</code>	Which method to use to identify cores and periphery. By default this is "degree", which relies on the heuristic that high degree nodes are more likely to be in the core. An alternative is "eigenvector", which instead begins with high eigenvector nodes. Other methods, such as a genetic algorithm, CONCOR, and Rombach-Porter, can be added if there is interest.

### 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  $\{i, j, \dots, n\}$  are ordered in descending degree,  $A$  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.

## References

- Borgatti, Stephen P., & Everett, Martin G. 1999. Models of core /periphery structures. *Social Networks*, 21, 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 memberships: [cliques](#), [community](#), [components\(\)](#), [equivalence](#)

## Examples

```
#mpn_elite_usa_advice %>% as_tidygraph %>%
#   mutate(corep = node_core(mpn_elite_usa_advice)) %>%
#   autographr(node_color = "corep")
network_core(mpn_elite_usa_advice)
node_coreness(ison_adolescents)
```

---

degree centrality      *Measures of degree-like centrality and centralisation*

---

## Description

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

- `node_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_deg()` returns the unnormalised results.
  - `node_indegree()` returns the `direction = 'in'` results.
  - `node_outdegree()` returns the `direction = 'out'` results.
- `node_multidegree()` measures the ratio between types of ties in a multiplex network.
- `node_posneg()` measures the PN (positive-negative) centrality of a signed network.
- `tie_degree()` measures the degree centrality of ties in a network
- `network_degree()` measures a network’s degree centralization; there are several related shortcut functions:
  - `network_indegree()` returns the `direction = 'out'` results.
  - `network_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.

**Usage**

```

node_degree(
  .data,
  normalized = TRUE,
  alpha = 1,
  direction = c("all", "out", "in")
)

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

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

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

node_multidegree(.data, tie1, tie2)

node_posneg(.data)

tie_degree(.data, normalized = TRUE)

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

network_outdegree(.data, normalized = TRUE)

network_indegree(.data, normalized = TRUE)

```

**Arguments**

<code>.data</code>	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
<code>normalized</code>	Logical scalar, whether the centrality scores are normalized. Different denominators are used depending on whether the object is one-mode or two-mode, the type of centrality, and other arguments.
<code>alpha</code>	Numeric scalar, the positive tuning parameter introduced in Opsahl et al (2010) for trading off between degree and strength centrality measures. By default, <code>alpha = 0</code> , which ignores tie weights and the measure is solely based upon degree (the number of ties). <code>alpha = 1</code> 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. For two-mode networks, “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.
tie1	Character string indicating the first uniplex network.
tie2	Character string indicating the second uniplex network.

### Value

A single centralization score if the object was one-mode, and two centralization scores if the object was two-mode.

Depending on how and what kind of an object is passed to the function, the function will return a `tidygraph` object where the nodes have been updated

### References

- 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.
- 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
- 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

`to_undirected()` for removing edge directions and `to_unweighted()` for removing weights from a graph.

Other centrality: `between_centrality`, `close_centrality`, `eigenv_centrality`

Other measures: `between_centrality`, `close_centrality`, `closure`, `cohesion()`, `eigenv_centrality`, `features`, `heterogeneity`, `hierarchy`, `holes`, `net_diffusion`, `node_diffusion`, `periods`

### Examples

```
node_degree(mpn_elite_mex)
node_degree(ison_southern_women)
tie_degree(ison_adolescents)
network_degree(ison_southern_women, direction = "in")
```

---

eigenv\_centrality      *Measures of eigenvector-like centrality and centralisation*

---

### Description

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

- `node_eigenvector()` measures the eigenvector centrality of nodes in a network.
- `node_power()` measures the Bonacich, beta, or power centrality of nodes in a network.
- `node_alpha()` measures the alpha or Katz centrality of nodes in a network.
- `node_pagerank()` measures the pagerank centrality of nodes in a network.
- `tie_eigenvector()` measures the eigenvector centrality of ties in a network.
- `network_eigenvector()` measures the eigenvector centralization for 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

```
node_eigenvector(.data, normalized = TRUE, scale = FALSE)
```

```
node_power(.data, normalized = TRUE, scale = FALSE, exponent = 1)
```

```
node_alpha(.data, alpha = 0.85)
```

```
node_pagerank(.data)
```

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

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

### Arguments

<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• igraph, from the <code>{igraph}</code> package</li> <li>• network, from the <code>{network}</code> package</li> <li>• tbl_graph, from the <code>{tidygraph}</code> package</li> </ul>
<code>normalized</code>	Logical scalar, whether the centrality scores are normalized. Different denominators are used depending on whether the object is one-mode or two-mode, the type of centrality, and other arguments.

scale	Logical scalar, whether to rescale the vector so the maximum score is 1.
exponent	Decay rate for the Bonacich power centrality score.
alpha	A constant that trades off the importance of external influence against the importance of connection. When $\alpha = 0$ , only the external influence matters. As $\alpha$ gets larger, only the connectivity matters and we reduce to eigenvector centrality. By default $\alpha = 0.85$ .

### Details

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

### Value

A numeric vector giving the eigenvector centrality measure of each node.

A numeric vector giving each node's power centrality measure.

### 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  $\lambda$  is a constant representing the principal eigenvalue. Rather than performing this iteration, most routines solve the eigenvector equation  $Ax = \lambda x$ .

### Power centrality

Power or beta (or Bonacich) centrality

### Alpha centrality

Alpha or Katz (or Katz-Bonacich) centrality operates better than eigenvector centrality for directed networks. 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,  $\lambda$  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  $\alpha$ , 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  $x = (I - \frac{1}{\lambda} A^T)^{-1} e$ .



## References

- Bonacich, Phillip. 1991. "Simultaneous Group and Individual Centralities." *Social Networks* 13(2):155–68. doi:10.1016/03788733(91)90018O.
- Bonacich, Phillip. 1987. "Power and Centrality: A Family of Measures." *The American Journal of Sociology*, 92(5): 1170–82. doi:10.1086/228631.
- 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.
- 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.

## See Also

Other centrality: [between\\_centrality](#), [close\\_centrality](#), [degree\\_centrality](#)

Other measures: [between\\_centrality](#), [close\\_centrality](#), [closure](#), [cohesion\(\)](#), [degree\\_centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

## Examples

```
node_eigenvector(mpn_elite_mex)
node_eigenvector(ison_southern_women)
node_power(ison_southern_women, exponent = 0.5)
tie_eigenvector(ison_adolescents)
network_eigenvector(mpn_elite_mex)
network_eigenvector(ison_southern_women)
```

---

equivalence

*Equivalence clustering algorithms*

---

## Description

These functions combine an appropriate `_census()` function together with methods for calculating the hierarchical clusters provided by a certain distance calculation.

- `node_equivalence()` assigns nodes membership based on their equivalence with respective to some census/class. The following functions call this function, together with an appropriate census.
  - `node_structural_equivalence()` assigns nodes membership based on their having equivalent ties to the same other nodes.
  - `node_regular_equivalence()` assigns nodes membership based on their having equivalent patterns of ties.
  - `node_automorphic_equivalence()` assigns nodes membership based on their having equivalent distances to other nodes.

A `plot()` method exists for investigating the dendrogram of the hierarchical cluster and showing the returned cluster assignment.

**Usage**

```
node_equivalence(
  .data,
  census,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor"),
  distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  range = 8L
)
```

```
node_structural_equivalence(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor"),
  distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  range = 8L
)
```

```
node_regular_equivalence(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor"),
  distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  range = 8L
)
```

```
node_automorphic_equivalence(
  .data,
  k = c("silhouette", "elbow", "strict"),
  cluster = c("hierarchical", "concor"),
  distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  range = 8L
)
```

**Arguments**

<code>.data</code>	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
<code>census</code>	A matrix returned by a <code>node_*_census()</code> function.
<code>k</code>	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 <code>stats::dist</code> . By default "euclidean", but other options include "maximum", "manhattan", "canberra", "binary", and "minkowski". Fewer, identifiable letters, e.g. "e" for Euclidean, is sufficient.
range	Integer indicating the maximum number of (k) clusters to evaluate. Ignored when k = "strict" or a discrete number is given for k.

### Source

<https://github.com/aslez/concoR>

### See Also

Other memberships: [cliques](#), [community](#), [components\(\)](#), [core](#)

### Examples

```
(nse <- node_structural_equivalence(mpn_elite_usa_advice))
plot(nse)
```

```
(nre <- node_regular_equivalence(mpn_elite_usa_advice,
  cluster = "concor"))
plot(nre)
```

```
(nae <- node_automorphic_equivalence(mpn_elite_usa_advice,
  k = "elbow"))
plot(nae)
```

## Description

These functions measure certain topological features of networks:

- `network_core()` measures the correlation between a network and a core-periphery model with the same dimensions.
- `network_richclub()` measures the rich-club coefficient of a network.
- `network_factions()` measures the correlation between a network and a component model with the same dimensions. If no 'membership' vector is given for the data, `node_kernighanlin()` is used to partition nodes into two groups.
- `network_modularity()` measures the modularity of a network based on nodes' membership in defined clusters.
- `network_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.
- `network_scalefree()` measures the exponent of a fitted power-law distribution. An exponent between 2 and 3 usually indicates a power-law distribution.
- `network_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.
- `network_change()` measures the Hamming distance between two or more networks.
- `network_stability()` measures the Jaccard index of stability between two or more networks.

These `network_*`() functions return a single numeric scalar or value.

## Usage

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

```
network_richclub(.data)
```

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

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

```
network_smallworld(.data, method = c("omega", "sigma", "SWI"), times = 100)
```

```
network_scalefree(.data)
```

```
network_balance(.data)
```

## Arguments

- |                    |   |
|--------------------|---|
| <code>.data</code> | An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• igraph, from the <code>{igraph}</code> package</li> <li>• network, from the <code>{network}</code> package</li> </ul> |
|--------------------|---|

	<ul style="list-style-type: none"> <li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li> </ul>
membership	A vector of partition membership.
resolution	A proportion indicating the resolution scale. By default 1.
method	There are three small-world measures implemented: <ul style="list-style-type: none"> <li>• "sigma" is the original equation from Watts and Strogatz (1998),</li> </ul>

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

, 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.  $\omega$  ranges between 0 and 1, where 1 is as close to a small-world as possible.

- "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.  $SWI$  also ranges between 0 and 1 with the same interpretation, but where there may not be a network for which  $SWI = 1$ .

times	Integer of number of simulations.
-------	-----------------------------------

## 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, and ranges between -1 and +1. Modularity scores of +1 mean that ties only appear within communities, while -1 would mean that ties only appear between communities. 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.

## Source

`{signnet}` by David Schoch

## References

- 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
- 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
- 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.

## See Also

[network\\_transitivity\(\)](#) and [network\\_equivalency\(\)](#) for how clustering is calculated

Other measures: [between\\_centrality](#), [close\\_centrality](#), [closure](#), [cohesion\(\)](#), [degree\\_centrality](#), [eigenv\\_centrality](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

## Examples

```
network_core(ison_adolescents)
network_core(ison_southern_women)
network_richclub(ison_adolescents)
  network_factions(mpn_elite_mex)
  network_factions(ison_southern_women)
network_modularity(ison_adolescents,
  node_kernighanlin(ison_adolescents))
network_modularity(ison_southern_women,
  node_kernighanlin(ison_southern_women))
network_smallworld(ison_brandes)
network_smallworld(ison_southern_women)
network_scalefree(ison_adolescents)
network_scalefree(generate_scalefree(50, 1.5))
network_scalefree(create_lattice(100))
network_balance(ison_marvel_relationships)
```

---

heterogeneity

*Measures of network diversity*

---

## Description

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

- [network\\_richness\(\)](#) measures the number of unique categories in a network attribute.

- `node_richness()` measures the number of unique categories of an attribute to which each node is connected.
- `network_diversity()` measures the heterogeneity of ties across a network or within clusters by node attributes.
- `node_diversity()` measures the heterogeneity of each node's local neighbourhood.
- `network_heterophily()` measures how embedded nodes in the network are within groups of nodes with the same attribute.
- `node_heterophily()` measures each node's embeddedness within groups of nodes with the same attribute.
- `network_assortativity()` measures the degree assortativity in a network.
- `network_spatial()` measures the spatial association/autocorrelation ( global Moran's I) in a network.

### Usage

```
network_richness(.data, attribute)
```

```
node_richness(.data, attribute)
```

```
network_diversity(.data, attribute, clusters = NULL)
```

```
node_diversity(.data, attribute)
```

```
network_heterophily(.data, attribute)
```

```
node_heterophily(.data, attribute)
```

```
network_assortativity(.data)
```

```
network_spatial(.data, attribute)
```

### Arguments

<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"><li>• <code>matrix</code> (adjacency or incidence) from <code>{base}</code> R</li><li>• <code>edgelist</code>, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li><li>• <code>igraph</code>, from the <code>{igraph}</code> package</li><li>• <code>network</code>, from the <code>{network}</code> package</li><li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li></ul>
<code>attribute</code>	Name of a nodal attribute or membership vector to use as categories for the diversity measure.
<code>clusters</code>	A nodal cluster membership vector or name of a vertex attribute.

**network\_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  $i$ th 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.

**network\_homophily**

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.

**References**

- Blau, Peter M. (1977). *Inequality and heterogeneity*. New York: Free Press.
- Krackhardt, David and Robert N. Stern (1988). Informal networks and organizational crises: an experimental simulation. *Social Psychology Quarterly* 51(2), 123-140.
- Moran, Patrick Alfred Pierce. 1950. "Notes on Continuous Stochastic Phenomena". *Biometrika* 37(1): 17-23. [doi:10.2307/2332142](https://doi.org/10.2307/2332142)

**See Also**

Other measures: [between centrality](#), [close centrality](#), [closure](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [hierarchy](#), [holes](#), [net diffusion](#), [node diffusion](#), [periods](#)

**Examples**

```
network_richness(mpn_bristol)
node_richness(mpn_bristol, "type")
marvel_friends <- manynet::to_unsigned(manyonet::ison_marvel_relationships, "positive")
network_diversity(marvel_friends, "Gender")
network_diversity(marvel_friends, "Attractive")
network_diversity(marvel_friends, "Gender", "Rich")
node_diversity(marvel_friends, "Gender")
node_diversity(marvel_friends, "Attractive")
network_heterophily(marvel_friends, "Gender")
network_heterophily(marvel_friends, "Attractive")
node_heterophily(marvel_friends, "Gender")
node_heterophily(marvel_friends, "Attractive")
```



```
network_assortativity(mpn_elite_mex)
network_spatial(ison_lawfirm, "age")
```

---

hierarchy

*Graph theoretic dimensions of hierarchy*


---

## Description

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

- `network_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.
- `network_efficiency()` measures the Krackhardt efficiency score.
- `network_upperbound()` measures the Krackhardt (least) upper bound score.

## Usage

```
network_connectedness(.data)
```

```
network_efficiency(.data)
```

```
network_upperbound(.data)
```

## Arguments

- `.data` An object of a `{manynet}`-consistent class:
- `matrix` (adjacency or incidence) from `{base}` R
  - `edgelist`, a data frame from `{base}` R or tibble from `{tibble}`
  - `igraph`, from the `{igraph}` package
  - `network`, from the `{network}` package
  - `tbl_graph`, from the `{tidygraph}` package

## References

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: [between\\_centrality](#), [close\\_centrality](#), [closure](#), [cohesion\(\)](#), [degree\\_centrality](#), [eigenv\\_centrality](#), [features](#), [heterogeneity](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

**Examples**

```

network_connectedness(ison_networkers)
1 - network_reciprocity(ison_networkers)
network_efficiency(ison_networkers)
network_upperbound(ison_networkers)

```

---

holes

---

*Measures of structural holes*


---

**Description**

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

- `node_bridges()` measures the sum of bridges to which each node is adjacent.
- `node_redundancy()` measures the redundancy of each nodes' contacts.
- `node_effsize()` measures nodes' effective size.
- `node_efficiency()` measures nodes' efficiency.
- `node_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_hierarchy()` measures nodes' exposure to hierarchy, where only one or two contacts are the source of closure.
- `node_eccentricity()` measures nodes' eccentricity or Koenig number, a measure of fairness based on number of links needed to reach most distant node in the network.
- `node_neighbours_degree()` measures nodes' average nearest neighbors degree, or *knn*, a measure of the type of local environment a node finds itself in
- `tie_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

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_bridges(.data)

node_redundancy(.data)

node_effsize(.data)

node_efficiency(.data)

node_constraint(.data)

```

```
node_hierarchy(.data)
node_eccentricity(.data)
node_neighbours_degree(.data)
tie_cohesion(.data)
```

## Arguments

`.data` An object of a {manynet}-consistent class:

- matrix (adjacency or incidence) from {base} R
- edgelist, a data frame from {base} R or tibble from {tibble}
- igraph, from the {igraph} package
- network, from the {network} package
- tbl\_graph, from the {tidygraph} package

## 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.

## References

- 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.
- 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 measures: [between centrality](#), [close centrality](#), [closure](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [net\\_diffusion](#), [node\\_diffusion](#), [periods](#)

**Examples**

```

node_bridges(ison_adolescents)
node_bridges(ison_southern_women)
node_redundancy(ison_adolescents)
node_redundancy(ison_southern_women)
node_effsize(ison_adolescents)
node_effsize(ison_southern_women)
node_efficiency(ison_adolescents)
node_efficiency(ison_southern_women)
node_constraint(ison_southern_women)
node_hierarchy(ison_adolescents)
node_hierarchy(ison_southern_women)

```

---

kselect

*Methods for selecting clusters*


---

**Description**

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 optimises 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, census, range)
```

```
k_silhouette(hc, .data, range)
```

**Arguments**

<code>hc</code>	A hierarchical clustering object.
<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• <code>igraph</code>, from the <code>{igraph}</code> package</li> <li>• <code>network</code>, from the <code>{network}</code> package</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>tbl_graph</code>, from the <code>{tidygraph}</code> package</li> </ul>
<code>census</code>	A motif census object.
<code>range</code>	An integer indicating the maximum number of options to consider. The minimum of this and the number of nodes in the network is used.

## References

- Thorndike, Robert L. 1953. "Who Belongs in the Family?". *Psychometrika*, 18(4): 267–76. doi:10.1007/BF02289263.
- 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.

---

<code>mpn_bristol</code>	<i>Multimodal (3) Bristol protest events, 1990-2002 (Diani and Bison 2004)</i>
--------------------------	--

---

## Description

A multimodal network with three levels representing ties between individuals, civic organisations in Bristol, and major protest and civic events that occurred between 1990 and 2000. The data contains individuals' affiliations to civic organizations in Bristol, the participation of these individuals in major protest and civic events between 1990-2002, and the involvement of the civic organizations in these events.

## Usage

```
data(mpn_bristol)
```

## Format

```
#> # A labelled, two-mode network with 264 nodes and 1496 ties
#> # A tibble: 264 x 3
#>   name type   lvl
#>   <chr> <lg1> <dbl>
#> 1 101  FALSE     1
#> 2 102  FALSE     1
#> 3 103  FALSE     1
#> 4 104  FALSE     1
#> 5 105  FALSE     1
#> 6 106  FALSE     1
#> # i 258 more rows
#> # A tibble: 1,496 x 2
#>   from  to
#>   <int> <int>
#> 1    36  151
```

```
#> 2    40   151
#> 3    73   151
#> 4    94   151
#> 5   138   151
#> 6   145   151
#> # i 1,490 more rows
```

## Details

Although represented as a two-mode network, it contains three levels:

1. 150 Individuals, anonymised with numeric ID
2. 97 Bristol civic organizations
3. 17 Major protest and civic events in Bristol, 1990-2002

The network represents ties between level 1 (individuals) and level 2 (organisations), level 1 (individuals) and level 3 (events), as well as level 2 (organisations) and level 3 (events). The network is simple, undirected, and named. For a complete list of civic organisations and protest/civic events included in the data, please see Appendix 6.1 in *Multimodal Political Networks* (Knoke et al., 2021).

## Source

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

## References

Diani, Mario, and Ivano Bison. 2004. "Organizations, Coalitions, and Movements." *Theory and Society* 33(3-4):281-309. doi:10.1023/B:RYSO.0000038610.00045.07.

---

mpn\_cow

*One-mode interstate trade relations and two-mode state membership in IGOs (COW)*

---

## Description

One-mode interstate trade relations and two-mode state membership in IGOs (COW)

## Usage

```
data(mpn_cow_trade)
```

```
data(mpn_cow_igo)
```

**Format**

```

#> # A labelled, weighted, directed network with 116 nodes and 11489 arcs
#> # A tibble: 116 x 1
#>   name
#>   <chr>
#> 1 United States of America
#> 2 Canada
#> 3 Cuba
#> 4 Dominican Republic
#> 5 Jamaica
#> 6 Trinidad and Tobago
#> # i 110 more rows
#> # A tibble: 11,489 x 3
#>   from to weight
#>   <int> <int> <dbl>
#> 1     1     2 180387
#> 2     1     3   587.
#> 3     1     4  5511.
#> 4     1     5  1896.
#> 5     1     6  2188.
#> 6     1     7 123677
#> # i 11,483 more rows

#> # A labelled, weighted, two-mode network with 152 nodes and 839 ties
#> # A tibble: 152 x 3
#>   name      type polity2
#>   <chr>    <lg1> <dbl>
#> 1 Afghanistan FALSE   -7
#> 2 Albania   FALSE    5
#> 3 Algeria   FALSE   -3
#> 4 Angola    FALSE   -6
#> 5 Argentina FALSE    8
#> 6 Australia FALSE   10
#> # i 146 more rows
#> # A tibble: 839 x 3
#>   from to weight
#>   <int> <int> <dbl>
#> 1     1   113     1
#> 2     1   114     1
#> 3     1   115     0
#> 4     1   116     0
#> 5     1   117     1
#> 6     1   118     0
#> # i 833 more rows

```

**Details**

mpn\_cow\_trade is a one-mode matrix representing the trade relations between 116 states. The

data is derived from the Correlates of War Project (COW) Trade Dataset (v3.0), which contains the annual dyadic and national trade figures for states (listed in COW) between 1870 to 2009. This network is based only on the dyadic trade figures in 2009 for the 116 states listed in Appendix 7.1 in *Multimodal Political Networks* (Knoke et al., 2021). The value in each cell of the matrix represents the value of exports from the 116 row states to the 116 column states.

mpn\_cow\_igo is a two-mode graph representing the membership of 116 states in 40 intergovernmental organizations (IGOs). The data is derived from the Correlates of War Project (COW) Intergovernmental Organizations Dataset (v3.0), which contains information about intergovernmental organizations from 1815-2014, such as founding year and membership. This network contains only a subset of the states and IGOs listed in COW, with 116 states listed in Appendix 7.1 in *Multimodal Political Networks* and 40 IGOs from Table 7.1 in *Multimodal Political Networks* that also overlap with the COW dataset (Knoke et al., 2021).

### Source

The Correlates of War Project. 2012. *Trade*.

Barbieri, Katherine and Omar Keshk. 2012. Correlates of War Project Trade Data Set Codebook, Version 3.0.

The Correlates of War Project. 2019. *Intergovernmental Organization v3*.

### References

Barbieri, Katherine, Omar M. G. Keshk, and Brian Pollins. 2009. "TRADING DATA: Evaluating our Assumptions and Coding Rules." *Conflict Management and Peace Science* 26(5): 471-491. doi:10.1177/0738894209343887.

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

Pevehouse, Jon C.W., Timothy Nordstrom, Roseanne W McManus, Anne Spencer Jamison. 2020. "Tracking Organizations in the World: The Correlates of War IGO Version 3.0 datasets". *Journal of Peace Research* 57(3): 492-503. doi:10.1177/0022343319881175.

---

mpn\_elite\_mex

*One-mode Mexican power elite database (Knoke 1990)*

---

### Description

This data contains the full network of 35 members of the Mexican power elite. The undirected lines connecting pairs of men represent any formal, informal, or organizational relation between a dyad; for example, "common belonging (school, sports, business, political participation), or a common interest (political power)" (Mendieta et al. 1997: 37). Additional nodal attributes include their full name, place of birth, state, and region (1=North, 2=Centre, 3=South, original coding added by Frank Heber), as well as their year of entry into politics and whether they are civilian (0) or affiliated with the military (1). An additional variable "in\_mpn" can be used to subset this network to a network of 11 core members of the 1990s Mexican power elite (Knoke 2017), three of which were successively elected presidents of Mexico: José López Portillo (1976-82), Miguel de la Madrid (1982-88), and Carlos Salinas de Gortari (1988-94, who was also the son of another core member, Raúl Salinas Lozano).



**Usage**

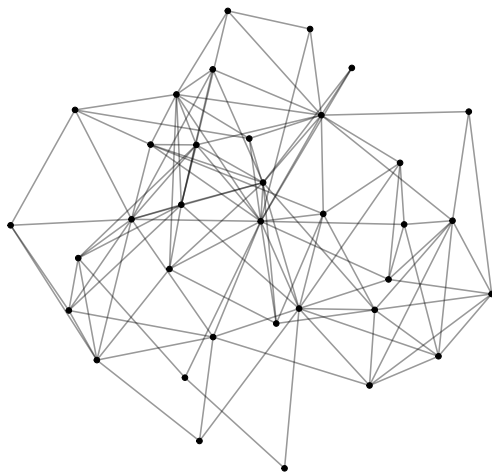
```
data(mpn_elite_mex)
```

**Format**

```
#> # A labelled, undirected network with 35 nodes and 117 ties
#> # A tibble: 35 x 8
#>   name      full_name      entry_year military in_mpn PlaceOfBirth state region
#>   <chr>    <chr>                <dbl>    <dbl> <dbl> <chr>      <chr> <dbl>
#> 1 Trevino  Trevino, Jacint~      1910      1     0 Guerrero  Coah~    1
#> 2 Madero   Madero, Francis~      1911      0     0 Parras de l~ Coah~    1
#> 3 Carranza Carranza, Venus~      1913      1     0 Cuatro Cien~ Coah~    1
#> 4 Aguilar  Aguilar, Candido      1918      1     0 Cordoba     Vera~    3
#> 5 Obregon  Obregon, Alvaro       1920      1     0 Siquisiva, ~ Sono~    1
#> 6 Calles   Calles, Plutarc~      1924      1     0 Guaymas     Sono~    1
#> # i 29 more rows
#> # A tibble: 117 x 2
#>   from  to
#>   <int> <int>
#> 1     2   3
#> 2     2   5
#> 3     2   6
#> 4     2   4
#> 5     1   2
#> 6     2   8
#> # i 111 more rows
```

**Details**

mpn\_elite\_mex



**Source**

Knoke, David. 1990. *Political Networks*.

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

---

mpn_elite_usa	<i>Two-mode and three-mode American power elite database (Domhoff 2016)</i>
---------------	---

---

**Description**

mpn\_elite\_usa\_advice is a 2-mode network of persons serving as directors or trustees of think tanks. Think tanks are “public-policy research analysis and engagement organizations that generate policy-oriented research, analysis, and advice on domestic and international issues, thereby enabling policymakers and the public to make informed decisions about public policy” (McGann 2016: 6). The Power Elite Database (Domhoff 2016) includes information on the directors of 33 prominent think tanks in 2012. Here we include only 14 directors who held three or more seats among 20 think tanks.

mpn\_elite\_usa\_money is based on 26 elites who sat on the boards of directors for at least two of six economic policy making organizations (Domhoff 2016), and also made campaign contributions to one or more of six candidates running in the primary election contests for the 2008 Presidential nominations of the Republican Party (Rudy Giuliani, John McCain, Mitt Romney) or the Democratic Party (Hillary Clinton, Christopher Dodd, Barack Obama).

**Usage**

```
data(mpn_elite_usa_advice)
```

```
data(mpn_elite_usa_money)
```

**Format**

```
#> # A labelled, two-mode network with 34 nodes and 46 ties
#> # A tibble: 34 x 2
#>   type name
#>   <lg1> <chr>
#> 1 FALSE Albright
#> 2 FALSE Argyros
#> 3 FALSE Armitage
#> 4 FALSE Curry
#> 5 FALSE Fukuyama
#> 6 FALSE Gray
#> # i 28 more rows
#> # A tibble: 46 x 2
#>   from to
#>   <int> <int>
```

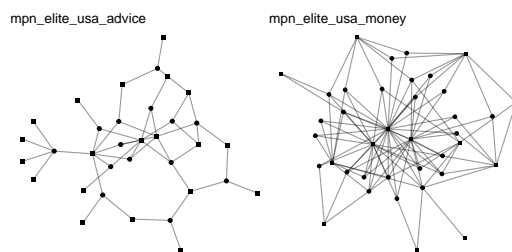
```

#> 1      1      17
#> 2      1      19
#> 3      1      21
#> 4      2      22
#> 5      2      23
#> 6      2      27
#> # i 40 more rows

#> # A labelled, two-mode network with 38 nodes and 103 ties
#> # A tibble: 38 x 2
#>   type name
#>   <lg1> <chr>
#> 1 FALSE Adkerson
#> 2 FALSE Akins
#> 3 FALSE Banga
#> 4 FALSE Boyce
#> 5 FALSE Britt
#> 6 FALSE Cannon
#> # i 32 more rows
#> # A tibble: 103 x 2
#>   from to
#>   <int> <int>
#> 1     1  27
#> 2     1  28
#> 3     1  34
#> 4     1  36
#> 5     2  28
#> 6     2  32
#> # i 97 more rows

```

## Details



## References

- Domhoff, G William. 2016. “Who Rules America? Power Elite Database.”
- The Center for Responsive Politics. 2019. “OpenSecrets.”
- Knoke, David, Mario Diani, James Hollway, and Dimitris C Christopoulos. 2021. *Multimodal Political Networks*. Cambridge University Press. Cambridge University Press.

---

mpn\_evs

*Two-mode European Values Survey, 1990 and 2008 (EVS 2020)*

---

### Description

6 two-mode matrices containing individuals' memberships to 14 different types of associations in three countries (Italy, Germany, and the UK) in 1990 and 2008. The Italy data has 658 respondents in 1990 and 540 in 2008. The Germany data has 1369 respondents in 1990 and 503 in 2008. The UK data has 738 respondents in 1990 and 664 in 2008.

### Usage

`data(mpn_IT_1990)`

`data(mpn_IT_1990)`

`data(mpn_IT_2008)`

`data(mpn_DE_1990)`

`data(mpn_DE_2008)`

`data(mpn_UK_1990)`

`data(mpn_UK_2008)`

### Format

`tbl_graph` object based on an association matrix with 14 columns:

**Welfare** 1 if individual associated

**Religious** 1 if individual associated

**Education.culture** 1 if individual associated

**Unions** 1 if individual associated

**Parties** 1 if individual associated

**Local.political.groups** 1 if individual associated

**Human.rights** 1 if individual associated

**Environmental.animal** 1 if individual associated

**Professional** 1 if individual associated

**Youth** 1 if individual associated

**Sports** 1 if individual associated

**Women** 1 if individual associated

**Peace** 1 if individual associated

**Health** 1 if individual associated

An object of class tbl\_graph (inherits from igraph) of length 672.

An object of class tbl\_graph (inherits from igraph) of length 554.

An object of class tbl\_graph (inherits from igraph) of length 1383.

An object of class tbl\_graph (inherits from igraph) of length 517.

An object of class tbl\_graph (inherits from igraph) of length 752.

An object of class tbl\_graph (inherits from igraph) of length 678.

## Source

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

## References

EVS (2020). European Values Study Longitudinal Data File 1981-2008 (EVS 1981-2008). GESIS Data Archive, Cologne. ZA4804 Data file Version 3.1.0, doi:10.4232/1.13486.

---

mpn\_ryanair

*One-mode EU policy influence network, June 2004 (Christopoulos 2006)*

---

## Description

Network of anonymised actors reacting to the Ryanair/Charleroi decision of the EU Commission in February 2004. The relationships mapped comprise an account of public records of interaction supplemented with the cognitive network of key informants. Examination of relevant communiques, public statements and a number of off-the-record interviews provides confidence that the network mapped closely approximated interactions between 29 January and 12 February 2004. The time point mapped is at the height of influence and interest intermediation played by actors in the AER, a comparatively obscure body representing the interests of a number of European regional bodies at the EU institutions.

## Usage

```
data(mpn_ryanair)
```

## Format

```
#> # A labelled, weighted, directed network with 20 nodes and 177 arcs
#> # A tibble: 20 x 1
#>   name
#>   <chr>
#> 1 1 AER
#> 2 2 AER
```

```

#> 3 5 AER/COR
#> 4 7 RYANAIR
#> 5 8 DG TRANSPORT
#> 6 9 COR
#> # i 14 more rows
#> # A tibble: 177 x 3
#>   from   to weight
#>   <int> <int> <dbl>
#> 1     1     2     1
#> 2     1     3     1
#> 3     1     4     1
#> 4     1     5     1
#> 5     1     6     1
#> 6     1     7     1
#> # i 171 more rows

```

### Source

Christopoulos, Dimitrios C. 2006. “Relational Attributes of Political Entrepreneurs: a Network Perspective.” *Journal of European Public Policy* 13(5): 757–78. doi:10.1080/13501760600808964.

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

---

mpn\_senate112

*Two-mode 112th Congress Senate Voting (Knoke et al. 2021)*


---

### Description

These datasets list the U.S. Senators who served in the 112th Congress, which met from January 3, 2011 to January 3, 2013. Although the Senate has 100 seats, 103 persons served during this period due to two resignations and a death. However, the third replacement occurred only two days before the end and cast no votes on the bills investigated here. Hence, the number of Senators analyzed is 102.

CQ Almanac identified 25 key bills on which the Senate voted during the 112th Congress, and which Democratic and Republican Senators voting “yea” and “nay” on each proposal.

Lastly, we obtained data on campaign contributions made by 92 PACs from the Open Secrets Website. We recorded all contributions made during the 2008, 2010, and 2012 election campaigns to the 102 persons who were Senators in the 112th Congress. The vast majority of PAC contributions to a candidate during a campaign was for \$10,000 (the legal maximum is \$5,000 each for a primary and the general election). We aggregated the contributions across all three electoral cycles, then dichotomized the sums into no contribution (0) and any contribution (1).

### Usage

```
data(mpn_DemSxP)
```

```
data(mpn_RepSxP)
```

```
data(mpn_OverSxP)
```

### Format

```
#> # A labelled, weighted, two-mode network with 114 nodes and 2791 ties
```

```
#> # A tibble: 114 x 2
```

```
#>   type name
```

```
#>   <lg1> <chr>
```

```
#> 1 FALSE Baucus
```

```
#> 2 FALSE Begich
```

```
#> 3 FALSE Bennet
```

```
#> 4 FALSE Blumenthal
```

```
#> 5 FALSE Boxer
```

```
#> 6 FALSE BrownSh
```

```
#> # i 108 more rows
```

```
#> # A tibble: 2,791 x 3
```

```
#>   from to weight
```

```
#>   <int> <int> <dbl>
```

```
#> 1     1     52     1
```

```
#> 2     1     53     1
```

```
#> 3     1     54     1
```

```
#> 4     1     55     1
```

```
#> 5     1     56     1
```

```
#> 6     1     57     1
```

```
#> # i 2,785 more rows
```

```
#> # A labelled, weighted, two-mode network with 134 nodes and 3675 ties
```

```
#> # A tibble: 134 x 2
```

```
#>   type name
```

```
#>   <lg1> <chr>
```

```
#> 1 FALSE Alexander
```

```
#> 2 FALSE Ayotte
```

```
#> 3 FALSE Barrasso
```

```
#> 4 FALSE Baucus
```

```
#> 5 FALSE Blunt
```

```
#> 6 FALSE Boozman
```

```
#> # i 128 more rows
```

```
#> # A tibble: 3,675 x 3
```

```
#>   from to weight
```

```
#>   <int> <int> <dbl>
```

```
#> 1     1     64     1
```

```
#> 2     1     66     1
```

```
#> 3     1     67     1
```

```
#> 4     1     70     1
```

```
#> 5     1     71     1
```

```
#> 6     1     72     1
```

```
#> # i 3,669 more rows
```

```
#> # A labelled, weighted, two-mode network with 52 nodes and 614 ties
#> # A tibble: 52 x 2
#>   type name
#>   <lg1> <chr>
#> 1 FALSE Baucus
#> 2 FALSE Cardin
#> 3 FALSE Carper
#> 4 FALSE Casey
#> 5 FALSE Collins
#> 6 FALSE Feinstein
#> # i 46 more rows
#> # A tibble: 614 x 3
#>   from to weight
#>   <int> <int> <dbl>
#> 1     1     21     1
#> 2     1     22     1
#> 3     1     23     1
#> 4     1     24     1
#> 5     1     25     1
#> 6     1     26     1
#> # i 608 more rows
```

## References

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

---

network\_census

*Censuses of motifs at the network level*

---

## Description

These functions include ways to take a census of the positions of nodes in a network:

- `network_dyad_census()` returns a census of dyad motifs in a network.
- `network_triad_census()` returns a census of triad motifs in a network.
- `network_mixed_census()` returns a census of triad motifs that span a one-mode and a two-mode network.

## Usage

```
network_dyad_census(.data)
```

```
network_triad_census(.data)
```

```
network_mixed_census(.data, object2)
```



**Arguments**

.data	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
object2	A second, two-mode migraph-consistent object.

**Source**

Alejandro Espinosa 'netmem'

**References**

Davis, James A., and Samuel Leinhardt. 1967. "The Structure of Positive Interpersonal Relations in Small Groups." 55.

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

**See Also**

Other motifs: [brokerage\\_census](#), [node\\_census](#)

**Examples**

```
network_dyad_census(manynet::ison_algebra)
network_triad_census(manynet::ison_adolescents)
marvel_friends <- manynet::to_unsigned(manynet::ison_marvel_relationships, "positive")
(mixed_cen <- network_mixed_census(marvel_friends, manynet::ison_marvel_teams))
```

---

net\_diffusion

*Measures of network diffusion*

---

**Description**

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

- `network_transmissibility()` measures the average transmissibility observed in a diffusion simulation, or the number of new infections over the number of susceptible nodes.
- `network_infection_length()` measures the average number of time steps nodes remain infected once they become infected.
- `network_reproduction()` measures the observed reproductive number in a diffusion simulation as the network's transmissibility over the network's average infection length.

- `network_immunity()` measures the proportion of nodes that would need to be protected through vaccination, isolation, or recovery for herd immunity to be reached.
- `network_hazard()` measures the hazard rate or instantaneous probability that nodes will adopt/become infected at that time

### Usage

`network_transmissibility(diff_model)`

`network_infection_length(diff_model)`

`network_reproduction(diff_model)`

`network_immunity(diff_model)`

`network_hazard(diff_model)`

### Arguments

`diff_model`      A valid network diffusion model, as created by `as_diffusion()` or `play_diffusion()`.

### Transmissibility

`network_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.

### Infection length

`network_infection_length()` 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

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

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

where  $T$  is the observed transmissibility in a diffusion and  $IL$  is the observed infection length in a diffusion. Since  $IL$  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

`network_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 `network_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.

### 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:

$$\lambda(t) = \lim_{h \rightarrow +0} \frac{F(t+h) - F(t)}{h} \frac{1}{1 - F(t)}$$

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**

Kermack, W. and McKendrick, A., 1927. "A contribution to the mathematical theory of epidemics". *Proc. R. Soc. London A* 115: 700-721.

Allison, P. 1984. *Event history analysis regression for longitudinal event data*. London: Sage Publications.

Wooldridge, J. M. 2010. *Econometric Analysis of Cross Section and Panel Data* (2nd ed.). Cambridge: MIT Press.

**See Also**

Other measures: [between centrality](#), [close centrality](#), [closure](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [node\\_diffusion](#), [periods](#)

Other diffusion: [node\\_diffusion](#)

**Examples**

```
smeg <- manynet::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
network_transmissibility(smeg_diff)
# To calculate the average infection length for a given diffusion model
network_infection_length(smeg_diff)
# To calculate the reproduction number for a given diffusion model
network_reproduction(smeg_diff)
# Calculating the proportion required to achieve herd immunity
network_immunity(smeg_diff)
# To find the number of nodes to be vaccinated
ceiling(network_immunity(smeg_diff) * manynet::network_nodes(smeg))
# To calculate the hazard rates at each time point
network_hazard(play_diffusion(smeg, transmissibility = 0.3))
```

---

node\_census

*Censuses of nodes' motifs*


---

**Description**

These functions include ways to take a census of the positions of nodes in a network:

- `node_tie_census()` 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_triad_census()` returns a census of the triad configurations nodes are embedded in.

- `node_quad_census()` returns a census of nodes' positions in motifs of four nodes.
- `node_path_census()` returns the shortest path lengths of each node to every other node in the network.

### Usage

```
node_tie_census(.data)
```

```
node_triad_census(.data)
```

```
node_quad_census(.data)
```

```
node_path_census(.data)
```

### Arguments

- `.data` An object of a `{manynet}`-consistent class:
- matrix (adjacency or incidence) from `{base}` R
  - edgelist, a data frame from `{base}` R or tibble from `{tibble}`
  - `igraph`, from the `{igraph}` package
  - `network`, from the `{network}` package
  - `tbl_graph`, from the `{tidygraph}` package

### Quad census

The quad census uses the `{oaqc}` package to do the heavy lifting of counting the number of each orbits. See `vignette('oaqc')`. However, our function relabels some of the motifs to avoid conflicts and improve some consistency with other census-labelling practices. The letter-number pairing of these labels indicate the number and configuration of ties. For now, we offer a rough translation:

<code>migraph</code>	Ortmann and Brandes
<code>E4</code>	<code>co-K4</code>
<code>I40, I41</code>	<code>co-diamond</code>
<code>H4</code>	<code>co-C4</code>
<code>L42, L41, L40</code>	<code>co-paw</code>
<code>D42, D40</code>	<code>co-claw</code>
<code>U42, U41</code>	<code>P4</code>
<code>Y43, Y41</code>	<code>claw</code>
<code>P43, P42, P41</code>	<code>paw</code>
<code>O4</code>	<code>C4</code>
<code>Z42, Z43</code>	<code>diamond</code>
<code>X4</code>	<code>K4</code>

See also [this list of graph classes](#).

## References

- Davis, James A., and Samuel Leinhardt. 1967. "The Structure of Positive Interpersonal Relations in Small Groups." 55.
- 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.
- 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: [brokerage\\_census](#), [network\\_census](#)

## Examples

```
task_eg <- manynet::to_named(manynet::to_uniplex(manynet::ison_algebra, "tasks"))
(tie_cen <- node_tie_census(task_eg))
(triad_cen <- node_triad_census(task_eg))
node_quad_census(manynet::ison_southern_women)
node_path_census(manynet::ison_adolescents)
node_path_census(manynet::ison_southern_women)
```

---

node\_diffusion

*Measures of nodes in a diffusion*

---

## Description

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

- `node_adoption_time()`: Measures the number of time steps until nodes adopt/become infected
- `node_adopter()`: Classifies membership of nodes into diffusion categories
- `node_thresholds()`: Measures nodes' thresholds from the amount of exposure they had when they became infected
- `node_infection_length()`: Measures the average length nodes that become infected remain infected in a compartmental model with recovery
- `node_exposure()`: Measures how many exposures nodes have to a given mark
- `node_is_exposed()`: Marks the nodes that are susceptible, i.e. are in the immediate neighbourhood of given mark vector

**Usage**

```
node_adoption_time(diff_model)

node_adopter(diff_model)

node_thresholds(diff_model)

node_infection_length(diff_model)

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

**Arguments**

diff_model	A valid network diffusion model, as created by <code>as_diffusion()</code> or <code>play_diffusion()</code> .
.data	An object of a {manynet}-consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from {base} R</li> <li>• edgelist, a data frame from {base} R or tibble from {tibble}</li> <li>• igraph, from the {igraph} package</li> <li>• network, from the {network} package</li> <li>• tbl_graph, from the {tidygraph} package</li> </ul>
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.

**Adoption time**

`node_adoption_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.

**Adopter class**

`node_adopter()` classifies the nodes involved in a diffusion 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

## Thresholds

node\_thresholds() 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.

## Infection length

node\_infection\_length() 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

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

## See Also

Other measures: [between centrality](#), [close centrality](#), [closure](#), [cohesion\(\)](#), [degree centrality](#), [eigenv centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [periods](#)

Other diffusion: [net\\_diffusion](#)

## Examples

```
smeg <- manynet::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_adoption_time(smeg_diff))
# To classify nodes by their position in the adoption curve
(adopts <- node_adopter(smeg_diff))
summary(adopts)
summary(times, membership = adopts)
# To infer nodes' thresholds
node_thresholds(smeg_diff)
# To measure how long each node remains infected for
node_infection_length(smeg_diff)
# To measure how much exposure nodes have to a given mark
node_exposure(smeg, mark = c(1,3))
node_exposure(smeg_diff)
```



**Description**

Helper functions for measuring over splits of networks

**Usage**

```
over_waves(
  .data,
  FUN,
  ...,
  attribute = "wave",
  strategy = "sequential",
  verbose = FALSE
)

over_time(
  .data,
  FUN,
  ...,
  attribute = "time",
  slice = NULL,
  strategy = "sequential",
  verbose = FALSE
)
```

**Arguments**

<code>.data</code>	An object of a <code>{manynet}</code> -consistent class: <ul style="list-style-type: none"> <li>• matrix (adjacency or incidence) from <code>{base}</code> R</li> <li>• edgelist, a data frame from <code>{base}</code> R or tibble from <code>{tibble}</code></li> <li>• igraph, from the <code>{igraph}</code> package</li> <li>• network, from the <code>{network}</code> package</li> <li>• tbl_graph, from the <code>{tidygraph}</code> package</li> </ul>
<code>FUN</code>	A function to run over all splits.
<code>...</code>	Further arguments to be passed on to <code>FUN</code> .
<code>attribute</code>	A string naming the attribute to be split upon.
<code>strategy</code>	If <code>{furrr}</code> is installed, then multiple cores can be used to accelerate the function. By default "sequential", but if multiple cores available, then "multisession" or "multicore" may be useful. Generally this is useful only when times > 1000. See <code>{furrr}</code> for more.
<code>verbose</code>	Whether the function should report on its progress. By default FALSE. See <code>{progressr}</code> for more.

slice                      Optionally, a vector of specific slices. Otherwise all observed slices will be returned.

## Functions

- `over_waves()`: Runs a function, e.g. a measure, over waves of a panel network
- `over_time()`: Runs a function, e.g. a measure, over time slices of a dynamic network

---

periods

*Measures of network change*

---

## Description

These functions measure certain topological features of networks:

- `network_change()` measures the Hamming distance between two or more networks.
- `network_stability()` measures the Jaccard index of stability between two or more networks.

These `network_*`() functions return a numeric vector the length of the number of networks minus one. E.g., the periods between waves.

## Usage

```
network_change(.data, object2)
```

```
network_stability(.data, object2)
```

## Arguments

`.data`                      An object of a {manynet}-consistent class:

- matrix (adjacency or incidence) from {base} R
- edgelist, a data frame from {base} R or tibble from {tibble}
- igraph, from the {igraph} package
- network, from the {network} package
- tbl\_graph, from the {tidygraph} package

`object2`                      A network object.

## See Also

Other measures: [between\\_centrality](#), [close\\_centrality](#), [closure](#), [cohesion\(\)](#), [degree\\_centrality](#), [eigenv\\_centrality](#), [features](#), [heterogeneity](#), [hierarchy](#), [holes](#), [net\\_diffusion](#), [node\\_diffusion](#)

**Description**

This function provides an implementation of the multiple regression quadratic assignment procedure (MRQAP) for both one-mode and two-mode network linear models. It offers several advantages:

- it works with combined graph/network objects such as `igraph` and `network` objects by constructing the various dependent and independent matrices for the user.
- it uses a more intuitive formula-based system for specifying the model, with several ways to specify how nodal attributes should be handled.
- it can handle categorical variables (factors/characters) and interactions intuitively, naming the reference variable where appropriate.
- it relies on `{furrr}` for parallelising and `{progressr}` for reporting progress to the user, which can be useful when many simulations are required.
- results are `{broom}`-compatible, with `tidy()` and `glance()` reports to facilitate comparison with results from different models. Note that a *t*- or *z*-value is always used as the test statistic, and properties of the dependent network – modes, directedness, loops, etc – will always be respected in permutations and analysis.

**Usage**

```
network_reg(
  formula,
  .data,
  method = c("qap", "qapy"),
  times = 1000,
  strategy = "sequential",
  verbose = FALSE
)
```

**Arguments**

`formula` A formula describing the relationship being tested. Several additional terms are available to assist users investigate the effects they are interested in. These include:

- `ego()` constructs a matrix where the cells reflect the value of a named nodal attribute for an edge's sending node
- `alter()` constructs a matrix where the cells reflect the value of a named nodal attribute for an edge's receiving node
- `same()` constructs a matrix where a 1 reflects if two nodes' attribute values are the same
- `dist()` constructs a matrix where the cells reflect the absolute difference between the attribute's values for the sending and receiving nodes

- `sim()` constructs a matrix where the cells reflect the proportional similarity between the attribute's values for the sending and receiving nodes
- `tertius()` constructs a matrix where the cells reflect some aggregate of an attribute associated with a node's other ties. Currently "mean" and "sum" are available aggregating functions. 'ego' is excluded from these calculations. See Haunss and Hollway (2023) for more on this effect.
- dyadic covariates (other networks) can just be named

`.data` An object of a `{manynet}`-consistent class:

- `matrix` (adjacency or incidence) from `{base}` R
- `edgelist`, a data frame from `{base}` R or tibble from `{tibble}`
- `igraph`, from the `{igraph}` package
- `network`, from the `{network}` package
- `tbl_graph`, from the `{tidygraph}` package

`method` A method for establishing the null hypothesis. Note that "qap" uses Dekker et al's (2007) double semi-partialling technique, whereas "qapy" permutes only the `$y` variable. "qap" is the default.

`times` Integer indicating number of simulations used for quantile estimation. (Relevant to the null hypothesis test only - the analysis itself is unaffected by this parameter.) Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, `times=1000`. 1,000 - 10,000 repetitions recommended for publication-ready results.

`strategy` If `{furrr}` is installed, then multiple cores can be used to accelerate the function. By default "sequential", but if multiple cores available, then "multisession" or "multicore" may be useful. Generally this is useful only when `times > 1000`. See `{furrr}` for more.

`verbose` Whether the function should report on its progress. By default FALSE. See `{progressr}` for more.

## References

- Krackhardt, David. 1988. "Predicting with Networks: Nonparametric Multiple Regression Analysis of Dyadic Data." *Social Networks* 10(4):359–81. doi:10.1016/03788733(88)900044.
- Dekker, David, David Krackhardt, and Tom A. B. Snijders. 2007. "Sensitivity of MRQAP tests to collinearity and autocorrelation conditions." *Psychometrika* 72(4): 563-581. doi:10.1007/s11336-00790161.

## See Also

`vignette("p7linearmodel")`  
 Other models: [tests](#)

## Examples

```
networkers <- ison_networkers %>% to_subgraph(Discipline == "Sociology")
model1 <- network_reg(weight ~ alter(Citations) + sim(Citations),
  networkers, times = 20)
```

```
# Should be run many more `times` for publication-ready results
tidy(model1)
glance(model1)
plot(model1)
```

---

tests

*Conditional uniform graph and permutation tests*


---

## Description

These functions conduct tests of any network-level statistic:

- `test_random()` performs a conditional uniform graph (CUG) test of a measure against a distribution of measures on random networks of the same dimensions.
- `test_permutation()` performs a quadratic assignment procedure (QAP) test of a measure against a distribution of measures on permutations of the original network.
- `test_gof()` performs a chi-squared test on the squared Mahalanobis distance between a `diff_model` and `diff_models` objects.

## Usage

```
test_random(
  .data,
  FUN,
  ...,
  times = 1000,
  strategy = "sequential",
  verbose = FALSE
)

test_permutation(
  .data,
  FUN,
  ...,
  times = 1000,
  strategy = "sequential",
  verbose = FALSE
)

test_gof(diff_model, diff_models)
```

## Arguments

- `.data` An object of a `{manynet}`-consistent class:
- matrix (adjacency or incidence) from `{base}` R
  - edgelist, a data frame from `{base}` R or tibble from `{tibble}`

- `igraph`, from the `{igraph}` package
- `network`, from the `{network}` package
- `tbl_graph`, from the `{tidygraph}` package

FUN	A graph-level statistic function to test.
...	Additional arguments to be passed on to FUN, e.g. the name of the attribute.
times	Integer indicating number of simulations used for quantile estimation. (Relevant to the null hypothesis test only - the analysis itself is unaffected by this parameter.) Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, <code>times=1000</code> . 1,000 - 10,000 repetitions recommended for publication-ready results.
strategy	If <code>{furrr}</code> is installed, then multiple cores can be used to accelerate the function. By default <code>"sequential"</code> , but if multiple cores available, then <code>"multisession"</code> or <code>"multicore"</code> may be useful. Generally this is useful only when <code>times &gt; 1000</code> . See <code>{furrr}</code> for more.
verbose	Whether the function should report on its progress. By default <code>FALSE</code> . See <code>{progressr}</code> for more.
diff_model	A <code>diff_model</code> object is returned by <code>play_diffusion()</code> or <code>as_diffusion()</code> and contains a single empirical or simulated diffusion.
diff_models	A <code>diff_models</code> object is returned by <code>play_diffusions()</code> and contains a series of diffusion simulations.

### Mahalanobis distance

`test_gof()` takes a single `diff_model` object, which may be a single empirical or simulated diffusion, and a `diff_models` object containing many simulations. Note that currently only the goodness of fit of the

It returns a tibble (compatible with `broom::glance()`) that includes the Mahalanobis distance statistic between the observed and simulated distributions. It also includes a p-value summarising a chi-squared test on this statistic, listing also the degrees of freedom and number of observations. If the p-value is less than the convention 0.05, then one can argue that the first diffusion is not well captured by

### See Also

Other models: [regression](#)

### Examples

```
marvel_friends <- to_unsigned(ison_marvel_relationships)
marvel_friends <- to_giant(marvel_friends) %>%
  to_subgraph(PowerOrigin == "Human")
(cugtest <- test_random(marvel_friends, network_heterophily, attribute = "Attractive",
  times = 200))
plot(cugtest)
(qaptest <- test_permutation(marvel_friends,
  network_heterophily, attribute = "Attractive",
  times = 200))
```

```
plot(qptest)
# Playing a reasonably quick diffusion
x <- play_diffusion(generate_random(15), transmissibility = 0.7)
# Playing a slower diffusion
y <- play_diffusions(generate_random(15), transmissibility = 0.1, times = 40)
plot(x)
plot(y)
test_gof(x, y)
```

# Index

## \* centrality

between centrality, 3  
close centrality, 7  
degree centrality, 20  
eigenv centrality, 23

## \* datasets

mpn\_bristol, 37  
mpn\_cow, 38  
mpn\_elite\_mex, 40  
mpn\_elite\_usa, 42  
mpn\_evs, 44  
mpn\_ryanair, 45  
mpn\_senate112, 46

## \* diffusion

net\_diffusion, 49  
node\_diffusion, 54

## \* measures

between centrality, 3  
close centrality, 7  
closure, 9  
cohesion, 12  
degree centrality, 20  
eigenv centrality, 23  
features, 27  
heterogeneity, 30  
hierarchy, 33  
holes, 34  
net\_diffusion, 49  
node\_diffusion, 54  
periods, 58

## \* memberships

cliques, 6  
community, 13  
components, 17  
core, 19  
equivalence, 25

## \* models

regression, 59  
tests, 61

## \* motifs

brokerage\_census, 4  
network\_census, 48  
node\_census, 52

between centrality, 3, 8–10, 13, 22, 25, 30,  
32, 33, 35, 52, 56, 58

brokerage\_census, 4, 49, 54

cliques, 6, 17, 18, 20, 27

close centrality, 4, 7, 10, 13, 22, 25, 30,  
32, 33, 35, 52, 56, 58

closure, 4, 9, 9, 13, 22, 25, 30, 32, 33, 35, 52,  
56, 58

cluster, 11

cluster\_concor (cluster), 11

cluster\_hierarchical (cluster), 11

cohesion, 4, 9, 10, 12, 22, 25, 30, 32, 33, 35,  
52, 56, 58

community, 7, 13, 18, 20, 27

components, 7, 17, 17, 20, 27

core, 7, 17, 18, 19, 27

degree centrality, 4, 8–10, 13, 20, 25, 30,  
32, 33, 35, 52, 56, 58

eigenv centrality, 4, 8–10, 13, 22, 23, 30,  
32, 33, 35, 52, 56, 58

equivalence, 7, 17, 18, 20, 25

features, 4, 9, 10, 13, 22, 25, 27, 32, 33, 35,  
52, 56, 58

heterogeneity, 4, 9, 10, 13, 22, 25, 30, 30,  
33, 35, 52, 56, 58

hierarchy, 4, 9, 10, 13, 22, 25, 30, 32, 33, 35,  
52, 56, 58

holes, 4, 9, 10, 13, 22, 25, 30, 32, 33, 34, 52,  
56, 58

k\_elbow (kselect), 36



- k\_silhouette (kselect), 36
- k\_strict (kselect), 36
- kselect, 36
  
- mpn\_bristol, 37
- mpn\_cow, 38
- mpn\_cow\_igo (mpn\_cow), 38
- mpn\_cow\_trade (mpn\_cow), 38
- mpn\_DE\_1990 (mpn\_evs), 44
- mpn\_DE\_2008 (mpn\_evs), 44
- mpn\_DemSxP (mpn\_senate112), 46
- mpn\_elite\_mex, 40
- mpn\_elite\_usa, 42
- mpn\_elite\_usa\_advice (mpn\_elite\_usa), 42
- mpn\_elite\_usa\_money (mpn\_elite\_usa), 42
- mpn\_evs, 44
- mpn\_IT\_1990 (mpn\_evs), 44
- mpn\_IT\_2008 (mpn\_evs), 44
- mpn\_OverSxP (mpn\_senate112), 46
- mpn\_RepSxP (mpn\_senate112), 46
- mpn\_ryanair, 45
- mpn\_senate112, 46
- mpn\_UK\_1990 (mpn\_evs), 44
- mpn\_UK\_2008 (mpn\_evs), 44
  
- net\_diffusion, 4, 9, 10, 13, 22, 25, 30, 32, 33, 35, 49, 56, 58
- network\_adhesion (cohesion), 12
- network\_assortativity (heterogeneity), 30
- network\_balance (features), 27
- network\_betweenness (between centrality), 3
- network\_brokerage\_census (brokerage\_census), 4
- network\_census, 5, 48, 54
- network\_change (periods), 58
- network\_closeness (close centrality), 7
- network\_cohesion (cohesion), 12
- network\_components (cohesion), 12
- network\_congruency (closure), 9
- network\_connectedness (hierarchy), 33
- network\_core (features), 27
- network\_degree (degree centrality), 20
- network\_density (cohesion), 12
- network\_diameter (cohesion), 12
- network\_diversity (heterogeneity), 30
- network\_dyad\_census (network\_census), 48
- network\_efficiency (hierarchy), 33
- network\_eigenvector (eigenv centrality), 23
- network\_equivalency (closure), 9
- network\_equivalency(), 30
- network\_factions (features), 27
- network\_harmonic (close centrality), 7
- network\_hazard (net\_diffusion), 49
- network\_heterophily (heterogeneity), 30
- network\_immunity (net\_diffusion), 49
- network\_indegree (degree centrality), 20
- network\_infection\_length (net\_diffusion), 49
- network\_length (cohesion), 12
- network\_mixed\_census (network\_census), 48
- network\_modularity (features), 27
- network\_outdegree (degree centrality), 20
- network\_reach (close centrality), 7
- network\_reciprocity (closure), 9
- network\_reg (regression), 59
- network\_reproduction (net\_diffusion), 49
- network\_richclub (features), 27
- network\_richness (heterogeneity), 30
- network\_scalefree (features), 27
- network\_smallworld (features), 27
- network\_spatial (heterogeneity), 30
- network\_stability (periods), 58
- network\_transitivity (closure), 9
- network\_transitivity(), 30
- network\_transmissibility (net\_diffusion), 49
- network\_triad\_census (network\_census), 48
- network\_upperbound (hierarchy), 33
- node\_adopter (node\_diffusion), 54
- node\_adoption\_time (node\_diffusion), 54
- node\_alpha (eigenv centrality), 23
- node\_automorphic\_equivalence (equivalence), 25
- node\_betweenness (between centrality), 3
- node\_bridges (holes), 34
- node\_brokerage\_census (brokerage\_census), 4
- node\_brokering (brokerage\_census), 4
- node\_brokering\_activity (brokerage\_census), 4
- node\_brokering\_exclusivity

- (brokerage\_census), 4
- node\_census, 5, 49, 52
- node\_closeness (close\_centrality), 7
- node\_components (components), 17
- node\_constraint (holes), 34
- node\_core (core), 19
- node\_coreness (core), 19
- node\_deg (degree\_centrality), 20
- node\_degree (degree\_centrality), 20
- node\_diffusion, 4, 9, 10, 13, 22, 25, 30, 32, 33, 35, 52, 54, 58
- node\_diversity (heterogeneity), 30
- node\_eccentricity (holes), 34
- node\_edge\_betweenness (community), 13
- node\_efficiency (holes), 34
- node\_effsize (holes), 34
- node\_eigenvector (eigenv\_centrality), 23
- node\_equivalence (equivalence), 25
- node\_exposure (node\_diffusion), 54
- node\_fast\_greedy (community), 13
- node\_flow (between\_centrality), 3
- node\_fluid (community), 13
- node\_harmonic (close\_centrality), 7
- node\_heterophily (heterogeneity), 30
- node\_hierarchy (holes), 34
- node\_indegree (degree\_centrality), 20
- node\_induced (between\_centrality), 3
- node\_infection\_length (node\_diffusion), 54
- node\_infomap (community), 13
- node\_information (close\_centrality), 7
- node\_kernighanlin (community), 13
- node\_leading\_eigen (community), 13
- node\_leiden (community), 13
- node\_louvain (community), 13
- node\_multidegree (degree\_centrality), 20
- node\_neighbours\_degree (holes), 34
- node\_optimal (community), 13
- node\_outdegree (degree\_centrality), 20
- node\_pagerank (eigenv\_centrality), 23
- node\_path\_census (node\_census), 52
- node\_posneg (degree\_centrality), 20
- node\_power (eigenv\_centrality), 23
- node\_quad\_census (node\_census), 52
- node\_reach (close\_centrality), 7
- node\_reciprocity (closure), 9
- node\_redundancy (holes), 34
- node\_regular\_equivalence (equivalence), 25
- node\_richness (heterogeneity), 30
- node\_roulette (cliques), 6
- node\_springlass (community), 13
- node\_strong\_components (components), 17
- node\_structural\_equivalence (equivalence), 25
- node\_thresholds (node\_diffusion), 54
- node\_tie\_census (node\_census), 52
- node\_transitivity (closure), 9
- node\_triad\_census (node\_census), 52
- node\_walktrap (community), 13
- node\_weak\_components (components), 17
- over, 57
- over\_time (over), 57
- over\_waves (over), 57
- periods, 4, 9, 10, 13, 22, 25, 30, 32, 33, 35, 52, 56, 58
- regression, 59, 62
- test\_gof (tests), 61
- test\_permutation (tests), 61
- test\_random (tests), 61
- tests, 60, 61
- tie\_betweenness (between\_centrality), 3
- tie\_closeness (close\_centrality), 7
- tie\_cohesion (holes), 34
- tie\_degree (degree\_centrality), 20
- tie\_eigenvector (eigenv\_centrality), 23
- to\_undirected(), 3, 7, 20, 22, 23
- to\_unweighted(), 22