Package ‘migraph’

December 19, 2021

Title  Multimodal and Multilevel Network Analysis
Version  0.8.13
Date  2021-12-19

Description  A set of tools that extend common social network analysis packages for analysing multimodal and multilevel networks. It includes functions for one- and two-mode (and sometimes three-mode) centrality, centralization, clustering, and constraint, as well as for one- and two-mode network regression and block-modelling. All functions operate with matrices, edge lists, and 'igraph', 'network'/sna', and 'tidygraph' objects. The package is released as a complement to 'Multimodal Political Networks' (2021, ISBN:9781108985000), and includes various datasets used in the book.

URL  https://github.com/snlab-ch/migraph
BugReports  https://github.com/snlab-ch/migraph/issues
Depends  R (>= 3.6.0)
License  MIT + file LICENSE
Language  en-GB
Encoding  UTF-8
LazyData  true
RoxygenNote  7.1.2

Imports  dplyr, gg dendro, ggraph, ggplot2, gridExtra, igraph, magrittr, network, oaqc, purrr, RColorBrewer, rlang, sna, stringr, tibble, tidygraph, tidyr, concaveman, ggforce, readxl

Suggests  covr, knitr, patchwork, rmarkdown, roxygen2, testthat

VignetteBuilder  knitr
NeedsCompilation  no

Author  James Hollway [cph, cre, aut, ctb] (IHEID, <https://orcid.org/0000-0002-8361-9647>), Bernhard Bieri [ctb] (<https://orcid.org/0000-0001-5943-9059>)}
R topics documented:

Maintainer  James Hollway <james.hollway@graduateinstitute.ch>
Repository  CRAN
Date/Publication  2021-12-19 02:10:02 UTC

R topics documented:

add ......................................................... 3
autograph .................................................. 4
blockmodel .................................................. 5
blockmodel_vis .............................................. 6
brandes ..................................................... 7
census ....................................................... 7
centrality ................................................... 8
centralization .............................................. 10
cluster ....................................................... 11
coercion ..................................................... 12
cohesion ..................................................... 13
create ....................................................... 15
diversity ................................................... 16
degree ....................................................... 17
generate .................................................... 18

... (and more topics listed)
add

Adding and copying attributes from one graph to another

Description

Adding and copying attributes from one graph to another

Usage

add_node_attributes(object, attr_name, vector)
add_edge_attributes(object, object2)
copy_node_attributes(object, object2)
mutate_edges(object, object2, attr_name)

Arguments

object A migraph-consistent object.
attr_name Name of the new attribute in the resulting object.
vector A vector of values for the new attribute.
object2 A second object to copy nodes or edges from.

Examples

add_node_attributes(mpn_elite_mex, "wealth", 1:11)
add_node_attributes(mpn_elite_usa_advice, "wealth", 1:14)
autographr(mpn_elite_mex)
both <- mutate_edges(mpn_elite_mex, generate_random(mpn_elite_mex), "random")
autographr(both)
random <- to_uniplex(both, "random")
autographr(random)
autographr(to_uniplex(both, "orig"))
autographr  Quickly graph networks with sensible defaults

Description

The aim of this function is to provide users with a quick and easy graphing function that makes best use of the data, whatever its composition.

Usage

autographr(
  object,
  layout = "stress",
  labels = TRUE,
  node_color = NULL,
  node_group = NULL,
  node_shape = NULL,
  node_size = NULL,
  ...
)

Arguments

object       A migraph-consistent object.
layout       An igraph layout algorithm, currently defaults to ‘stress’.
labels       Logical, whether to print node names as labels if present.
node_color   Node variable in quotation marks to be used for colouring the nodes.
node_group   Node variable in quotation marks to be used for drawing convex but also con-
cave hulls around clusters of nodes. These groupings will be labelled with the categories of the variable passed.
node_shape   Node variable in quotation marks to be used for the shapes of the nodes. Shapes will be follow the ordering "circle", "square", "triangle", so this aesthetic should be used for a variable with only a few categories.
node_size    Node variable in quotation marks to be used for the size of the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all node-related statistics prior to using this function.
...

Extra arguments.

Examples

autographr(ison_coleman)
autographr(ison_karateka)
blockmodel  

Blockmodelling

Usage

blockmodel(object, clusters)

blockmodel_concor(
  object,
  p = 1,
  cutoff = 0.999,
  max.iter = 25,
  block.content = "density"
)

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

reduce_graph(blockmodel, block_labels = NULL)

summarise_statistics(node_measure, clusters = NULL, sumFUN = mean)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>A migraph-consistent object (matrix, igraph, tidygraph).</td>
</tr>
<tr>
<td>clusters</td>
<td>the vector of cluster membership for the blockmodel</td>
</tr>
<tr>
<td>p</td>
<td>An integer representing the desired number of partitions.</td>
</tr>
<tr>
<td>cutoff</td>
<td>A value between 0 and 1 used to determine convergence.</td>
</tr>
<tr>
<td>max.iter</td>
<td>An integer representing the maximum number of iterations.</td>
</tr>
<tr>
<td>block.content</td>
<td>A string indicating which method to use for calculating block content. Options are: &quot;density&quot;, &quot;sum&quot;, &quot;meanrowsum&quot;, &quot;meancolsum&quot;, &quot;median&quot;, &quot;min&quot;, &quot;max&quot;.</td>
</tr>
<tr>
<td>x</td>
<td>An object of class &quot;blockmodel&quot;</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments passed to generic print method</td>
</tr>
<tr>
<td>blockmodel</td>
<td>a blockmodel object</td>
</tr>
<tr>
<td>block_labels</td>
<td>A character vector manually providing labels for the blocks in the blockmodel</td>
</tr>
<tr>
<td>node_measure</td>
<td>A vector or matrix of node-level statistics, such as centrality measures or a census.</td>
</tr>
<tr>
<td>sumFUN</td>
<td>A function by which the values should be aggregated or summarised. By default mean.</td>
</tr>
</tbody>
</table>
Source

https://github.com/aslez/concoR

References


Examples

```r
mex_concor <- blockmodel_concor(mpn_elite_mex)
mex_concor
plot(mex_concor)
usa_concor <- blockmodel_concor(mpn_elite_usa_advice)
usa_concor
plot(usa_concor)
summarise_statistics(node_degree(mpn_elite_mex),
  cutree(cluster_structural_equivalence(mpn_elite_mex), 3))
summarise_statistics(node_triad_census(mpn_elite_mex),
  cutree(cluster_structural_equivalence(mpn_elite_mex), 3))
```

---

**blockmodel_vis**  
ggplot2-based plotting of blockmodel results

**Description**

ggplot2-based plotting of blockmodel results

Plots for deciding on the number of network clusters

**Usage**

```r
## S3 method for class 'blockmodel'
plot(x, ...)

ggtree(hc, k = NULL)

ggidentify_clusters(hc, census, method = c("elbow", "strict"))
```

**Arguments**

- `x` A blockmodel-class object.
- `...` Additional arguments passed on to ggplot2.
- `hc` a hierarchical cluster object
- `k` number of clusters. By default NULL, but, if specified, ggtree will color branches and add a line to indicate where the corresponding cluster cut would be.
census output from some node_*_census function
method only "elbow" is currently implemented.

Examples
usa_concor <- blockmodel_concor(mpn_elite_usa_advice)
plot(usa_concor)
res <- cluster_regular_equivalence(mpn_elite_mex)
ggtree(res, 4)
ggidentify_clusters(res, node_triad_census(mpn_elite_mex))

---

Description
One-mode centrality demonstration structure

Usage
data(brandes)

Format
A tidygraph tbl_graph with 11 nodes and 24 edges.

---

census Census by nodes or clusters

Description
These functions include ways to take a census of the positions of nodes in a network. These include a triad census based on the triad profile of nodes, but also a tie census based on the particular tie partners of nodes. Included also are group census functions for summarising the profiles of clusters of nodes in a network.

Usage
node_tie_census(object)
node_triad_census(object)
node_quad_census(object)

group_tie_census(object, clusters, decimals = 2)
group_triad_census(object, clusters, decimals = 2)
Arguments

- **object**: A migraph-consistent object.
- **clusters**: a vector of cluster assignment.
- **decimals**: Number of decimal points to round to.

Examples

```r
task_eg <- to_named(to_uniplex(ison_m182, "task_tie"))
tie_cen <- node_tie_census(task_eg)
triad_cen <- node_triad_census(task_eg)
quad_cen <- node_quad_census(southern_women)
group_tie_census(task_eg, cutree(cluster_structural_equivalence(task_eg), 4))
group_triad_census(task_eg, cutree(cluster_regular_equivalence(task_eg), 4))
```

---

**centrality** *Centrality for one- and two-mode networks*

**Description**

These functions calculate common centrality measures for both one- and two-mode networks. They accept as objects matrices and igraph graphs, and can be used within a tidygraph workflow. Importantly, these functions also offer correct normalization for two-mode networks.

**Usage**

```r
node_degree(
  object,
  weights = NULL,
  mode = "out",
  loops = TRUE,
  normalized = FALSE
)
```

```r
node_closeness(
  object,
  weights = NULL,
  mode = "out",
  normalized = FALSE,
  cutoff = NULL
)
```

```r
node_betweenness(
  object,
  weights = NULL,
  directed = TRUE,
  cutoff = NULL
)
```
node_eigenvector(
  object,
  weights = NULL,
  directed = FALSE,
  options = igraph::arpack_defaults,
  scale = FALSE,
  normalized = FALSE
)

Arguments

object Either an igraph graph object or a matrix.
weights The weight of the edges to use for the calculation. Will be evaluated in the context of the edge data.
mode How should edges be followed. Ignored for undirected graphs
loops Should loops be included in the calculation
normalized For one-mode networks, should Borgatti and Everett normalization be applied?
cutoff maximum path length to use during calculations
directed Should direction of edges be used for the calculations
nobigint Should big integers be avoided during calculations
options Settings passed on to igraph::arpack()
Scale Should the scores be scaled to range between 0 and 1?

Value

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

References


See Also

Other two-mode measures: centralization, cohesion, node_constraint(), node_smallworld()
Other node-level measures: node_constraint(), node_smallworld()
Examples

node_degree(mpn_elite_mex)
node_degree(southern_women)
node_closeness(mpn_elite_mex)
node_closeness(southern_women)
node_betweenness(mpn_elite_mex)
node_betweenness(southern_women)
node_eigenvector(mpn_elite_mex)
node_eigenvector(southern_women)

centralization  Centralization for one- and two-mode networks

Description

These functions measure the overall centralization for a network.

Usage

```r
graph_degree(
  object,
  directed = c("all", "out", "in", "total"),
  normalized = TRUE,
  digits = 2
)

graph_closeness(
  object,
  directed = c("all", "out", "in", "total"),
  normalized = TRUE,
  digits = 2
)

graph_betweenness(
  object,
  directed = c("all", "out", "in", "total"),
  normalized = TRUE,
  digits = 2
)

graph_eigenvector(object, digits = 2)
```

Arguments

- `object`  A matrix, igraph graph, or tidygraph object.
**cluster**

**directed**
Character string, “out” for out-degree, “in” for in-degree, and "all" or “total” for 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 nodeset.

**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.

**digits**
whether to round the resulting score, by default 2. Add FALSE to turn all rounding off.

**Value**
A single centralization score if the object was one-mode, and two centralization scores if the object was two-mode. In the case of a two-mode network, to return just the score for the first nodeset (rows), append $nodes1 to the end of the function call or returned object. To return just the score for the second nodeset (cols), append $nodes2 to the end of the function call or returned object.

**References**

**See Also**
Other two-mode measures: centrality, cohesion, node_constraint(), node_smallworld()

**Examples**
```r
graph_degree(southern_women, directed = "in")
graph_closeness(southern_women, directed = "in")
graph_betweenness(southern_women, directed = "in")
graph_eigenvector(mpn_elite_mex)
```

---

**cluster**

*Clustering algorithms*

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

**Usage**
```r
cluster_structural_equivalence(object)
cluster_regular_equivalence(object)
```
Arguments

object A migraph-consistent object.

Examples

ggtree(cluster_structural_equivalence(mpn_elite_mex))
ggtree(cluster_regular_equivalence(mpn_elite_mex))
ggtree(cluster_regular_equivalence(mpn_elite_usa_advice))

coercion Coercion between graph/network/edgelist/matrix object classes

Description

The as_ functions in {migraph} coerce objects between several common classes of social network objects. These include:

- edgelists, as data frames or tibbles
- adjacency and incidence matrices
- {igraph} graph objects
- {tidygraph} tbl_graph objects
- {network} network objects

Usage

as_edgelist(object, weight = FALSE)
as_matrix(object, weight = FALSE)
as_igraph(object, weight = FALSE, twomode = FALSE)
as_tidygraph(object, twomode = FALSE)
as_network(object)

Arguments

object A data frame edgelist, matrix, igraph, tidygraph, or network object.
weight An option to override the heuristics for distinguishing weighted networks. By default FALSE.
twomode An option to override the heuristics for distinguishing incidence from adjacency matrices. By default FALSE.
Details

An effort is made for all of these coercion routines to be as lossless as possible, though some object classes are better at retaining certain kinds of information than others. Note also that there are some reserved column names in one or more object classes, which could otherwise lead to some unexpected results.

Behaviour is a little different depending on the data format.

If the data frame is a 2 column edgelist, the first column will become the rows and the second column will become the columns. If the data frame is a 3 column edgelist, then the third column will be used as the cell values or tie weights.

Incidence matrices are typically inferred from unequal dimensions, but since in rare cases a matrix with equal dimensions may still be an incidence matrix, an additional argument `twomode` can be specified to override this heuristic. This information is usually already embedded in `{igraph}`, `{tidygraph}`, and `{network}` objects.

Value

The currently implemented coercions or translations are:

<table>
<thead>
<tr>
<th>to/from</th>
<th>edgelists</th>
<th>matrices</th>
<th>igraph</th>
<th>tidygraph</th>
<th>network</th>
</tr>
</thead>
<tbody>
<tr>
<td>edgelists (data frames)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>matrices</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>igraph</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>tidygraph</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>network</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

See Also

Other manipulation: `is()`, `project.to`

Examples

```r
test <- data.frame(id1 = c("A","B","B","C","C"),
                   id2 = c("I","G","I","G","H"))
as_matrix(test)
as_igraph(test)
as_tidygraph(test)
as_network(test)
```

Cohesion for one-, two-, and three- mode networks

Description

These functions offer methods for summarising the cohesion in one-, two-, and three-mode networks.
Usage

```r
graph_density(object)
graph_reciprocity(object, method = "default")
graph_transitivity(object)
graph_equivalency(object)
graph_congruency(object, object2)
```

Arguments

- **object**: A one-mode or two-mode matrix, igraph, or tidygraph
- **method**: For reciprocity, either default or ratio. See `igraph::reciprocity`
- **object2**: Optionally, a second (two-mode) matrix, igraph, or tidygraph

Details

For one- and two-mode networks, `graph_density` summarises the ratio of ties to the number of possible ties.

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

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

References


See Also

Other one-mode measures: `node_constraint()`

Other two-mode measures: `centrality`, `centralization`, `node_constraint()`, `node_smallworld()`

Examples

```r
graph_density(mpn_elite_mex)
graph_density(mpn_elite_usa_advice)
graph_reciprocity(southern_women)
```
graph_transitivity(southern_women)

graph_equivalency(southern_women)

create networks with particular structures

**Description**

These functions create a variety of different network objects. Despite the common function names and syntax with existing packages, the common `n` argument can not only be passed a single integer to return a one-mode network, but also a vector of two integers to return a two-mode network.

**Usage**

- `create_empty(n)`
- `create_complete(n)`
- `create_ring(n, width = 1, directed = FALSE, ...)`
- `create_components(n, components = 2)`
- `create_star(n, directed = c("undirected", "in", "out"))`
- `create_tree(n, directed = c("undirected", "in", "out"), branches = 2)`
- `create_lattice(n, directed = c("undirected", "in", "out"))`

**Arguments**

- `n` Number of nodes. If a single integer is given, e.g. `n = 10`, the function will create a one-mode network. If a vector of two integers is given, e.g. `n = c(5, 10)`, the function will create a two-mode network.
- `width` The width or breadth of the ring. This is typically double the degree.
- `directed` One of the following options: "in", "out", or "undirected" (DEFAULT).
- `...` Additional arguments passed on to igraph.
- `components` Number of components to create.
- `branches` How many branches at each level

**Details**

- `create_empty()` creates an empty graph of the given dimensions.
- `create_complete()` creates a filled graph of the given dimensions.
- `create_ring()` creates a ring or chord graph of the given dimensions that loops around is of a certain width or thickness.
create_components() creates a graph in which the nodes are clustered into separate components.
create_star() creates a graph of the given dimensions that has a maximally central node
create_tree() creates a graph of the given dimensions with successive branches
create_lattice() creates a graph of the given dimensions with ties to all neighbouring nodes

Value

By default an igraph object will be returned, but this can be coerced into other types of objects using as_matrix(), as_tidygraph(), or as_network().

See Also

as_matrix as_tidygraph as_network

Other creation: generate

Examples

g <- create_empty(c(8,6))
autographr(g)
g <- create_complete(c(8,6))
autographr(g)
g <- create_ring(c(8,6), width = 2)
autographr(g)
autographr(create_components(c(10,12), components = 3))
autographr(create_star(c(12,1), "in"))
tr1 <- autographr(create_tree(12))
tr2 <- autographr(create_tree(12), "tree")
grid.arrange(tr1, tr2, ncol = 2)
cl1 <- autographr(create_lattice(5))
cl2 <- autographr(create_lattice(c(5,5)))
cl3 <- autographr(create_lattice(c(5,5,5)))
grid.arrange(cl1, cl2, cl3, ncol = 3)

---

<table>
<thead>
<tr>
<th>diversity</th>
<th>Measures of network diversity</th>
</tr>
</thead>
</table>

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.

Usage

graph_blau_index(object, attribute, clusters = NULL)

graph_ei_index(object, attribute)
Arguments

- **object**: A data frame edgelist, matrix, igraph, tidygraph, or network object.
- **attribute**: The name of a vertex attribute to measure the diversity of.
- **clusters**: A nodal cluster membership vector or name of a vertex attribute.

Examples

```r
marvel_friends <- to_unsigned(ison_marvel_relationships, "positive")
graph_blau_index(marvel_friends, "Gender")
graph_blau_index(marvel_friends, "Attractive")
graph_blau_index(marvel_friends, "Gender", "Rich")
graph_ei_index(marvel_friends, "Gender")
graph_ei_index(marvel_friends, "Attractive")
```

---

**edge**

Identifying edges by certain properties

Description

Identifying edges by certain properties

Usage

- `edge_mutual(object)`
- `edge_multiple(object)`
- `edge_loop(object)`

Arguments

- **object**: A migraph-consistent class object (matrix, edgelist, igraph, network, tidygraph)

Functions

- `edge_mutual`: Identify edges that are mutual/reciprocated
- `edge_multiple`: Identify edges that are multiples
- `edge_loop`: Identify edges that are loops

Examples

```r
define_edges <- ison_m182
define_edges <- define_edges
```

---

```r
define_edges <- ison_m182
define_edges <- define_edges
```
Generate networks from particular probabilities

Usage

```r
generate_random(n, p, m, directed = FALSE)
generate_smallworld(n, p = 0.05)
generate_scalefree(n, p = 1)
generate_permutation(object)
```

Arguments

- `n`: Integer of length 1 or 2. If passed a migraph-consistent object, a random network of the same dimensions and density as the original network will be returned.
- `p`: Number of edges in the network over the number of edges possible
- `m`: Number of edges in the network
- `directed`: Whether to generate network as directed. By default FALSE.
- `object`: a migraph-consistent object

Details

Creates a random network. If `length(n) == 1`, then a one-mode network will be returned, equivalent to an Erdős-Renyi graph. If `length(n) == 1`, then a two-mode network will be returned. The first number is the number of nodes in the first nodeset (rows), and the second number becomes the number of nodes in the second nodeset (columns).

See Also

Other creation: `create`

Examples

```r
er1 <- autographr(generate_random(12, 0.4))
er2 <- autographr(generate_random(c(6, 6), 0.4))
grid.arrange(er1, er2, ncol = 2)
sw1 <- autographr(generate_smallworld(12, 0.025))
sw2 <- autographr(generate_smallworld(12, 0.25))
grid.arrange(sw1, sw2, ncol = 2)
sf1 <- autographr(generate_scalefree(12, 0.25))
sf2 <- autographr(generate_scalefree(12, 1.25))
```
grid.arrange(sf1, sf2, ncol = 2)
em1 <- autographr(mpn_elite_usa_advice)
em2 <- autographr(generate_permutation(mpn_elite_usa_advice))
ggrid.arrange(em1, em2, ncol = 2)

---

### gatyear

**Plotting network at particular timepoint (year)**

**Description**

Plotting network at particular timepoint (year)

**Usage**

```r
gatyear(edgelist, year)
```

**Arguments**

- `edgelist`: a manyverse edgelist, expecting `Beg` and `End` variables, among others
- `year`: numeric year, gets expanded to first of January that year

**Examples**

```r
## Not run:
gatyear(membs, 1900)
## End(Not run)
```

---

### ggevolution

**Plot the evolution of a network**

**Description**

This function offers a method to plot a network at two or more timepoints for quick and easy comparison. The function is currently limited to two networks and only the layout given by the first or last network, but further extensions expected.

**Usage**

```r
ggevolution(..., layout = "kk", based_on = c("first", "last", "both"))
```

**Arguments**

- `...`: two or more networks
- `layout`: an igraph layout. Default is Kamada-Kawai ("kk")
- `based_on`: whether the layout of the joint plots should be based on the "first" or the "last" network.
Examples

```r
mpn_elite_mex2 <- mpn_elite_mex %>%
  tidygraph::activate(edges) %>%
  tidygraph::reroute(from = sample.int(11, 44, replace = TRUE),
  to = sample.int(11, 44, replace = TRUE))
ggevolution(mpn_elite_mex, mpn_elite_mex2)
ggevolution(mpn_elite_mex, mpn_elite_mex2, based_on = "last")
ggevolution(mpn_elite_mex, mpn_elite_mex2, based_on = "both")
```

Description

Lineage implies a direct descent from an ancestor; ancestry or pedigree. That is, how observation
derives and is connected to previous observations. The function plots a lineage graph of citations,
amendments, and more, for example.

Usage

```r
gglineage(object, labels = TRUE)
```

Arguments

- `object`: A migraph-consistent network/graph.
- `labels`: Whether to plot node labels or not. Default: TRUE.

Examples

```r
gglineage(cites)
```
ggtools

Visualising graphs and identifying nodes with maximum values of the specified measure.

Description

Visualising graphs and identifying nodes with maximum values of the specified measure.

Usage

```r
ggidentify(object, node_measure, identify_function = max)

ggdistrib(object, node_measure)
```

Arguments

- `object` a migraph-consistent object
- `node_measure` some arbitrary function that runs on the object and returns a numeric vector that can be used to scale the nodes
- `identify_function` a function for the identification of a single node, e.g. max, min, mean, etc.

Examples

```r
ggidentify(brandes, node_degree)
ggidentify(brandes, node_betweenness)

gdistrib(brandes, node_degree)
```

grab

Helpers to grab various attributes from nodes or edges in a graph

Description

Helpers to grab various attributes from nodes or edges in a graph
graph_balance

Usage

node_names(object)

node_attribute(object, attribute)

edge_attribute(object, attribute)

edge_weights(object)

graph_nodes(object)

graph_edges(object)

graph_dims(object)

graph_node_attributes(object)

graph_edge_attributes(object)

Arguments

object A data frame edgelist, matrix, igraph, tidygraph, or network object.
attribute An attribute name.

Value

"triangles" returns the proportion of balanced triangles, ranging between 0 if all triangles are imbalanced and 1 if all triangles are balanced.

Source

{signnet} by David Schoch
graph_census

Censuses for the whole graph

Description

Censuses for the whole graph

Usage

graph_mixed_census(object, object2)

graph_dyad_census(object)

graph_triad_census(object)

Arguments

object     A migraph-consistent object.
object2    A second, two-mode migraph-consistent object.

Source

Alejandro Espinosa 'netmem'

References


Examples

marvel_friends <- to_unsigned(ison_marvel_relationships, "positive")
(mixed_cen <- graph_mixed_census(marvel_friends, ison_marvel_teams))
graph_dyad_census(ison_coleman)
graph_triad_census(ison_coleman)
graph_components  Number of components in the network

Description
Number of components in the network

Usage
graph_components(object, method = c("weak", "strong"))

Arguments
- object: a migraph-consistent object
- method: For directed networks, either weak if edge direction is irrelevant, or strong if edge direction is salient. Ignored if network undirected.

is  Tests of network properties

Description
These functions implement tests for various network properties.

Usage
- is_migraph(object)
- is_graph(object)
- is_twomode(object)
- is_weighted(object)
- is_directed(object)
- is_labelled(object)
- is_signed(object)
- is_connected(object, method = c("weak", "strong"))
- is_complex(object)
- is_multiplex(object)
is_uniplex(object)

is_acyclic(object)

Arguments

object A migraph-consistent class object (matrix, edgelist, igraph, network, tidygraph)

method Whether to identify components if only "weak"ly connected or also "strong"ly connected.

Value

TRUE if the condition is met, or FALSE otherwise.

Functions

- is_migraph: Tests whether network is migraph-compatible
- is_graph: Tests whether network contains graph-level information
- is_twomode: Tests whether network is a two-mode network
- is_weighted: Tests whether network is weighted
- is_directed: Tests whether network is directed
- is_labelled: Tests whether network includes names for the nodes
- is_signed: Tests whether network is signed positive/negative
- is_connected: Tests whether network is (weakly/strongly) connected
- is_complex: Tests whether network contains any loops
- is_multiplex: Tests whether network is multiplex
- is_uniplex: Tests whether network is simple (both uniplex and simplex)
- is_acyclic: Tests whether network is a directed acyclic graph

See Also

Other manipulation: coercion, project, to

Examples

is_twomode(southern_women)
is_weighted(southern_women)
is_directed(southern_women)
is_labelled(southern_women)
is_signed(southern_women)
is_connected(southern_women)
is_complex(southern_women)
is_uniplex(ison_m182)
is_acyclic(ison_m182)
### ison_coleman

*One-mode subset of adolescent society dataset*

**Description**

One-mode subset of adolescent society dataset

**Usage**

`data(ison_coleman)`

**Format**

tidygraph graph object

**References**


### ison_community

*Zachary's kareteka network*

**Description**

Zachary's kareteka network

**Usage**

`data(ison_karateka)`

**Format**

Undirected one-mode igraph with 34 nodes and 78 edges
ison_eies

1DW 32(2) 440(1) EIES messages

Description

1DW 32(2) 440(1) EIES messages

Usage

data(ison_eies)

Format

tidygraph graph object

Source

networkdata package

References


ison_marvel

Multilevel two-mode affiliation, signed one-mode networks of Marvel comic book characters

Description

Multilevel two-mode affiliation, signed one-mode networks of Marvel comic book characters

Usage

data(ison_marvel_teams)

data(ison_marvel_relationships)

Format

Two-mode igraph of 53 Marvel comic book characters and 141 team-ups, with 683 team affiliations between them

One-mode igraph of 53 Marvel comic book characters and 558 signed (1 = friends, -1 = enemies) undirected ties
Details

This package includes two datasets related to the Marvel comic book universe. The first, ison_marvel_teams, is a two-mode affiliation network of 53 Marvel comic book characters and their affiliations to 141 different teams. This network includes only information about nodes' names and nodeset, but additional nodal data can be taken from the other Marvel dataset here.

The second network, ison_marvel_relationships, is a one-mode signed network of friendships and enmities between the 53 Marvel comic book characters. Friendships are indicated by a positive sign in the edge sign attribute, whereas enmities are indicated by a negative sign in this edge attribute. Additional nodal variables have been coded and included by Dr Umut Yüksel:

- **Gender**: binary character, 43 "Male" and 10 "Female"
- **PowerOrigin**: binary character, 2 "Alien", 1 "Cyborg", 5 "God/Eternal", 22 "Human", 1 "Infection", 16 "Mutant", 5 "Radiation", 1 "Robot"
- **Appearances**: integer, in how many comic book issues they appeared in
- **Attractive**: binary integer, 41 1 (yes) and 12 0 (no)
- **Rich**: binary integer, 11 1 (yes) and 42 0 (no)
- **Intellect**: binary integer, 39 1 (yes) and 14 0 (no)
- **Omnilingual**: binary integer, 8 1 (yes) and 45 0 (no)
- **UnarmedCombat**: binary integer, 51 1 (yes) and 2 0 (no)
- **ArmedCombat**: binary integer, 25 1 (yes) and 28 0 (no)

Source

Umut Yüksel, 31 March 2017

---

ison_projection  Two-mode projection examples

Description

Two-mode projection examples

Usage

data(ison_mm)
data(ison_bm)
data(ison_mb)
data(ison_bb)
**Format**

Directed two-mode igraph with 6 nodes and 6 edges  
Directed two-mode igraph with 8 nodes and 9 edges  
Directed two-mode igraph with 8 nodes and 9 edges  
Directed two-mode igraph with 10 nodes and 12 edges

**Description**

Layouts for one- and two-mode networks

**Usage**

```r
layout_tbl_graph_frgrid(object, circular = FALSE, maxiter = 1000)
layout_tbl_graph_kkgrid(object, circular = FALSE, maxiter = 1000)
layout_tbl_graph_gogrid(object, circular = FALSE, maxiter = 1000)
```

**Arguments**

- `object` A migraph-consistent network/graph  
- `circular` Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE  
- `maxiter` maximum number of iterations, where appropriate

**Details**

The function uses approximate pattern matching to redistribute the coarse layouts on the square grid points, while preserving the topological relationships among the nodes (see Inoue et al. 2012).

**References**


**Examples**

```r
autographr(mpn_elite_mex, "frgrid")
autographr(mpn_ryanair, "frgrid")
autographr(mpn_elite_mex, "kkgrid")
autographr(mpn_ryanair, "kkgrid")
autographr(mpn_elite_mex, "gogrid")
autographr(mpn_ryanair, "gogrid")
```
m182  
*Multiplex igraph of friends, social, and task ties between 16 anonymous students*

**Description**

M182 was an honors algebra class and friendship, social, and task ties were collected/observed.

**Usage**

`data(ison_m182)`

**Format**

Multiplex tidygraph of friends, social, and task ties between 16 anonymous students. The edge attribute `friend_ties` contains friendship ties, where 2 = best friends, 1 = friend, and 0 is not a friend. `social_ties` consists of social interactions per hour, and `task_ties` consists of task interactions per hour.

**Source**

See also `data(studentnets.M182, package = "NetData")`. Larger comprehensive data set publicly available, contact Daniel A. McFarland for details.

mpn_bristol  
*Multimodal (3) Bristol protest events, 1990-2002*

**Description**

A multimodal (3) matrix containing individuals affiliations to civic organizations in Bristol and their participation in major protest and civic events between 1990-2002, and the involvement of the organizations in these events.

**Usage**

`data(mpn_bristol)`

**Format**

A matrix with 264 rows and columns. Node IDs are prefaced with a type identifier:

1. 150 Individuals, anonymised
2. 97 Bristol Civic Organizations
3. 17 Major Protest and Civic Events in Bristol, 1990-2002
mpn_elite_mex

Source

mpn_elite_mex  
One-mode Mexican power elite database

Description
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). 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).

Usage
```r
data(mpn_elite_mex)
```

Format
Matrix with 11 rows/columns

Source

mpn_elite_usa_advice  
Two-mode American power elite database

Description
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.

Usage
```r
data(mpn_elite_usa_advice)
```
Format

Matrix with 14 rows and 20 columns

References


---

**mpn_elite_usa_money**

Three-mode American power elite database

---

Description

This data 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_money)

Format

Matrix with 26 rows and 6+6 columns

References


Two-mode European Values Survey, 1990 and 2008

Description


Usage

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

Matrices 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 10.

Source


mpn_ryanair

One-mode EU policy influence network, June 2004

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

Matrix with 20 rows/columns

Source


Two-mode 112th Congress Senate Voting

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.

Usage
```r
data(mpn_DemSxP)
data(mpn_RepSxP)
data(mpn_OverSxP)
```

Format
- Matrix of 51 rows (Senators) and 63 columns (PACS)
- Matrix of 62 rows (Senators) and 72 columns (PACS)
- Matrix of 20 rows (Senators) and 32 columns (PACS)

Details
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).

Source
node_components  Identifying nodes’ component membership

Description
Identifying nodes’ component membership

Usage
node_components(object, method = c("weak", "strong"))

Arguments
object  a migraph-consistent object
method  For directed networks, either weak if edge direction is irrelevant, or strong if edge direction is salient. Ignored if network undirected.

node_constraint  Constraint for one- and two-mode networks

Description
This function measures constraint for both one-mode and two-mode networks. For one-mode networks, the function wraps the implementation of Ron Burt’s measure in {igraph}. For two-mode networks, the function employs the extension outlined in Hollway et al. (2020).

Usage
node_constraint(object, nodes = V(object), weights = NULL)

Arguments
object  A matrix, igraph graph, or tidygraph object.
nodes  The vertices for which the constraint will be calculated. Defaults to all vertices.
weights  The weights of the edges. If this is NULL and there is a weight edge attribute this is used. If there is no such edge attribute all edges will have the same weight.

Value
A named vector (one-mode) or a list of two named vectors ($nodes1, $nodes2).

References
See Also

Other one-mode measures: cohesion
Other two-mode measures: centrality, centralization, cohesion, node_smallworld()
Other node-level measures: centrality, node_smallworld()

Examples

node_constraint(southern_women)

---

Description

Calculates small-world metrics for two-mode networks

Usage

node_smallworld(object, niter = 100)

Arguments

object A matrix, igraph graph, or tidygraph object
niter Number of simulations

Details

The first column of the returned table is simply the number of the second-mode column. The next three columns report the observed and expected clustering, and the ratio of the former to the latter. The next three columns report the observed and expected path-length, and the ratio of the former to the latter. The last column reports the ratio of the observed/expected clustering ratio to the observed/expected path-length ratio, which is known as a small-world metric. Expected clustering and paths is the mean of twomode_clustering and mean_distance over 100 random simulations with the same row and column sums.

Value

Returns a table of small-world related metrics for each second-mode node.

See Also

graph_transitivity and graph_equivalency for how clustering is calculated
Other two-mode measures: centrality, centralization, cohesion, node_constraint()
Other node-level measures: centrality, node_constraint()

Examples

node_smallworld(southern_women)
**Description**

These functions ‘project’ or convert a two-mode object in any format – tidygraph, igraph, or matrix – into a corresponding one-mode object.

**Usage**

```r
project_rows(object)
```  

```r
project_cols(object)
```  

**Arguments**

- `object`: A matrix, igraph graph or tidygraph tbl_graph object.

**Details**

- `project_rows()` results in a weighted one-mode object that retains the row nodes from a two-mode object, and weights the ties between them on the basis of their joint ties to nodes in the second mode (columns).

- `project_cols()` results in a weighted one-mode object that retains the column nodes from a two-mode object, and weights the ties between them on the basis of their joint ties to nodes in the first mode (rows).

**See Also**

Other manipulation: `coercion`, `is()`, `to`

**Examples**

```r
project_rows(southern_women)
```  

```r
project_cols(southern_women)
```  

---

**Description**

Users regularly need to work with a variety of external data formats. The following functions offers ways to import from some common external file formats into objects that `{migraph}` and other graph/network packages in R can work with. Note that these functions are not as actively maintained as others in the package, so please let us know if any are not currently working for you by raising an issue on Github.
Usage

read_edgelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)
write_edgelist(object, filename, name, ...)
read_nodelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)
write_nodelist(object, filename, name, ...)
read_pajek(file = file.choose(), ...)
write_pajek(object, filename, ...)
read_ucinet(file = file.choose(), ...)
write_ucinet(object, filename, name)

Arguments

file A character string with the system path to the file to import. If left unspeci-
       fied, an OS-specific file picker is opened to help users select it. Note that in
read_ucinet() the file path should be to the header file (.##h), if it exists.
sv Allows users to specify whether their csv file is "comma" (English) or "semi-colon" (European) separated.
... Additional parameters passed to the read/write function.
object A migraph-consistent object to be exported.
filename UCINET filename (without ## extension). By default the files will have the
       same name as the object and be saved to the working directory.
name name of matrix to be known in UCINET. By default the name will be the same as the object.

Details

There are a number of repositories for network data that hold various datasets in different formats. See for example:

- UCINET data
- Pajek data

See also:

- networkdata
- GML datasets
- UCIrvine Network Data Repository
- KONECT project
- SNAP Stanford Large Network Dataset Collection
Please let us know if you identify any further repositories of social or political networks and we would be happy to add them here.

The _ucinet functions only work with relatively recent UCINET file formats, e.g. type 6406 files. To import earlier UCINET file types, you will need to update them first. To import multiple matrices packed into a single UCINET file, you will need to unpack them and convert them one by one.

Value

The `read_edgelist()` and `read_nodelist()` functions will import into edgelist (tibble) format which can then be coerced or combined into different graph objects from there.

The `read_pajek()` and `read_ucinet()` functions will import into a tidygraph format, since they already contain both edge and attribute data. Note that all graphs can be easily coerced into other formats with `{migraph}`'s as methods.

The `write_*` functions export to different file formats, depending on the function.

A pair of UCINET files in V6404 file format (.##h, .##d)

Functions

- `read_edgelist`: Reading edgelists from Excel/csv files
- `write_edgelist`: Writing edgelists to csv files
- `read_nodelist`: Reading nodelists from Excel/csv files
- `write_nodelist`: Writing nodelists to csv files
- `read_pajek`: Reading pajek (.net/.paj) files
- `write_pajek`: Writing pajek .net files
- `read_ucinet`: Reading UCINET files
- `write_ucinet`: Writing UCINET files

Author(s)

Christian Steglich, 18 June 2015

See Also

coercion

Examples

```r
## Not run:
# import Roethlisberger & Dickson's horseplay game data set:
horseplay <- read_ucinet("WIRING-RDGAM.##h")

## End(Not run)
## Not run:
# export it again to UCINET under a different name:
write_ucinet(horseplay,"R&D-horseplay")

## End(Not run)
```
**Description**

This function extends the multiple regression quadratic assignment procedure (MRQAP) of network linear model to two mode networks. It also works with combined graph/network objects by constructing the various dependent and independent matrices for the user. Lastly, because it relies on an object that contains all this information it can offer a more informative formula-based system for specifying the model.

**Usage**

```r
network_reg(formula, data, ...)
```

## S3 method for class 'netlm'

```r
summary(object, reps = 1000, ...)
```

## S3 method for class 'summary.netlm'

```r
print(x, digits = max(3, getOption("digits") - 3), signif.stars = getOption("show.signif.stars"), ...)
```

**Arguments**

- **formula**: A formula describing the relationship being tested.
- **data**: A named list of matrices, graphs, or a tidygraph object.
- **...**: Arguments passed on to `lm()`.
- **object**: an object of class "netlm", usually as a result of a call to `network_reg()`.
- **reps**: Integer indicating the number of draws to use 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, reps=1000.
- **x**: an object of class "summary.netlm", usually, a result of a call to `summary.netlm()`.
- **digits**: the number of significant digits to use when printing.
- **signif.stars**: logical. If TRUE, ‘significance stars’ are printed for each coefficient.

**Examples**

```r
messages <- mutate_edges(ison_eies, generate_random(ison_eies), attr_name = "random")
model1 <- network_reg(weight ~ random + same(Discipline) + same(Citations), messages)
summary(model1, reps = 200) # increase reps for publication
```
southern_women  Two-mode southern women dataset

Description
Two-mode network dataset collected by Davis, Gardner and Gardner (1941) about women and social events.

Usage
data(southern_women)

Format
igraph graph object

References

tests  Conditional uniform graph and permutation tests

Description
These functions conduct conditional uniform graph (CUG) or permutation (QAP) tests of any graph-level statistic.

Usage
test_random(object, FUN, ..., nSim = 1000)
test_permutation(object, FUN, ..., nSim = 1000)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>A data frame edgelist, matrix, igraph, tidygraph, or network object.</td>
</tr>
<tr>
<td>FUN</td>
<td>A graph-level statistic function to test.</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments to be passed on to FUN, e.g. the name of the attribute.</td>
</tr>
<tr>
<td>nSim</td>
<td>The number of Monte Carlo simulations to perform.</td>
</tr>
</tbody>
</table>
Examples

marvel_friends <- to_unsigned(ison_marvel_relationships)
marvel_friends <- to_main_component(marvel_friends) %>%
    filter(PowerOrigin == "Human")
(cugtest <- test_random(marvel_friends, graph_ei_index, attribute = "Attractive",
    nSim = 200))
plot(cugtest)
(qaptest <- test_permutation(marvel_friends, graph_ei_index, attribute = "Attractive",
    nSim = 200))
plot(qaptest)

Tools for reformatting networks, graphs, and matrices

Description

These functions offer tools for transforming certain properties of migraph-consistent objects (that is, matrices, igraph, tidygraph, or network objects).

Usage

to_unweighted(object, threshold = 1)
to_unnamed(object)
to_undirected(object)
to_onemode(object)
to_main_component(object)
to_uniplex(object, edge)
to_unsigned(object, keep = c("positive", "negative"))
to_simplex(object)
to_named(object)
to_multilevel(object)

Arguments

object A matrix, {igraph} graph, {tidygraph} tbl_graph, or {network} object.
threshold For a matrix, the threshold to binarise/dichotomise at.
edge the name of an edge attribute to retain from a graph
keep in the case of a signed network, whether to retain the "positive" or "negative" ties
Details

Since some modifications are easier to implement for some objects than others, here are the currently implemented modifications:

<table>
<thead>
<tr>
<th>to_</th>
<th>edgelists</th>
<th>matrices</th>
<th>igraph</th>
<th>tidygraph</th>
<th>network</th>
</tr>
</thead>
<tbody>
<tr>
<td>unweighted</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>undirected</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>unsigned</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniplex</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unnamed</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>named</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>simplex</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main_component</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>onemode</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multilevel</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Value

All to_ functions return an object of the same class as that provided. So passing it an igraph object will return an igraph object and passing it a network object will return a network object, with certain modifications as outlined below:

- to_unweighted() returns an object that has all edge weights removed
- to_unnamed() returns an object that has all vertex names removed
- to_named() returns an object that has random vertex names added
- to_undirected() returns an object that has any edge direction removed
- to_onemode() returns an object that has any type/mode attributes removed, but otherwise includes all the same nodes and ties. Note that this is not the same as project_rows() or project_cols(), which return only some of the nodes and new ties established by coincidence.
- to_main_component() returns an object that includes only the main component and not any smaller components or isolates
- to_uniplex() returns an object that includes only a single type of tie
- to_simplex() returns an object that has all loops or self-ties removed
- to_unsigned() returns an object that has

See Also

Other manipulation: coercion, is(), project

Examples

to_unweighted(project_rows(southern_women))
to_unnamed(project_rows(southern_women))
to_undirected(ison_coleman)
to_onemode(ison_marvel_teams)
to_uniplex(ison_m182, "friend_tie")
to_unsigned(ison_marvel_relationships, "positive")
to_unsigned(ison_marvel_relationships, "negative")
to_simplex(ison_m182)
to_named(ison_m182)
to_multilevel(mpn_elite_usa_advice)
Index

* creation
  create, 15
  generate, 18
* datasets
  brandes, 7
  ison_coleman, 26
  ison_community, 26
  ison_eies, 27
  ison_marvel, 27
  ison_projection, 28
  m182, 30
  mpn_bristol, 30
  mpn_elite_mex, 31
  mpn_elite_usa_advice, 31
  mpn_elite_usa_money, 32
  mpn_evs, 33
  mpn_ryanair, 34
  mpn_senate112, 35
  southern_women, 42
* graph-level measures
  centralization, 10
* manipulation
  coercion, 12
  is, 24
  project, 38
  to, 43
* node-level measures
  centrality, 8
  node_constraint, 36
  node_smallworld, 37
* one-mode measures
  cohesion, 13
  node_constraint, 36
* three-mode measures
  cohesion, 13
* two-mode measures
  centrality, 8
  centralization, 10
  cohesion, 13

node_constraint, 36
node_smallworld, 37
add, 3
add_edge_attributes (add), 3
add_node_attributes (add), 3
as_edgelist (coercion), 12
as_igraph (coercion), 12
as_matrix (coercion), 12
as_network (coercion), 12
as_tidygraph (coercion), 12
autographr, 4
blockmodel, 5
blockmodel_concor (blockmodel), 5
blockmodel_vis, 6
brandes, 7
census, 7
centrality, 8, 11, 14, 37
centralization, 9, 10, 14, 37
cluster, 11
cluster_regular_equivalence (cluster), 11
cluster_structural_equivalence (cluster), 11
coefficient, 12, 25, 38, 40, 44
cohesion, 9, 11, 13, 37
copy_node_attributes (add), 3
create, 15, 18
create_complete (create), 15
create_components (create), 15
create_empty (create), 15
create_lattice (create), 15
create_ring (create), 15
create_star (create), 15
create_tree (create), 15
diversity, 16
diversity, 16
diversity, 16
diversity, 16
diversity, 16
edge, 17
edge, 17
edge, 17
edge_attribute (grab), 21
edge_loop (edge), 17
edge_multiple (edge), 17
edge_mutual (edge), 17
edge_weights (grab), 21
generate, 16, 18
generate_permutation (generate), 18
generate_random (generate), 18
generate_scalefree (generate), 18
generate_smallworld (generate), 18
ggatyear, 19
ggdistrib (ggtools), 21
ggevolution, 19
ggidentify (ggtools), 21
ggidentify_clusters (blockmodel_vis), 6
gglineage, 20
ggtools, 21
ggtree (blockmodel_vis), 6
grab, 21
graph_balance, 22
graph_betweenness (centralization), 10
graph_blau_index (diversity), 16
graph_census, 23
graph_closeness (centralization), 10
graph_components, 24
graph_congruency (cohesion), 13
graph_degree (centralization), 10
graph_density (cohesion), 13
graph_dims (grab), 21
graph_dyad_census (graph_census), 23
graph_edge_attributes (grab), 21
graph_edges (grab), 21
graph_ei_index (diversity), 16
graph_eigenvector (centralization), 10
graph_equivalency, 37
graph_equivalency (cohesion), 13
graph_mixed_census (graph_census), 23
graph_node_attributes (grab), 21
graph_nodes (grab), 21
graph_reciprocity (cohesion), 13
graph_transitivity, 37
graph_transitivity (cohesion), 13
graph_triad_census (graph_census), 23
group_tie_census (census), 7

is, 13, 24, 38, 44
is_acyclic (is), 24
is_complex (is), 24
is_connected (is), 24
is_directed (is), 24
is_graph (is), 24
is_labelled (is), 24
is_migraph (is), 24
is_multiplex (is), 24
is_signed (is), 24
is_twomode (is), 24
is_uniplex (is), 24
is_weighted (is), 24
ison_bb (ison_projection), 28
ison_bm (ison_projection), 28
ison_coleman, 26
ison_community, 26
ison_eies, 27
ison_karateka (ison_community), 26
ison_m182 (m182), 30
ison_marvel, 27
ison_marvel_relationships (ison_marvel), 27
ison_marvel_teams (ison_marvel), 27
ison_mb (ison_projection), 28
ison_mm (ison_projection), 28
ison_projection, 28
layout_tbl_graph_fgrid (layouts), 29
layout_tbl_graph_gogrid (layouts), 29
layout_tbl_graph_kkgrid (layouts), 29
layouts, 29
m182, 30
mpn_bristol, 30
mpn_DE_1990 (mpn_evs), 33
mpn_DE_2008 (mpn_evs), 33
mpn_DemSxP (mpn_senate112), 35
mpn_elite_mex, 31
mpn_elite_usa_advice, 31
mpn_elite_usa_money, 32
mpn_evs, 33
mpn_IT_1990 (mpn_evs), 33
mpn_IT_2008 (mpn_evs), 33
mpn_OverSxP (mpn_senate112), 35
mpn_RepSxP (mpn_senate112), 35
mpn_ryanair, 34
mpn_senate112, 35
mpn_UK_1990 (mpn_evs), 33
mpn_UK_2008 (mpn_evs), 33
mutate_edges (add), 3
INDEX

network_reg (regression), 41
node_attribute (grab), 21
node_betweenness (centrality), 8
node_closeness (centrality), 8
node_components, 36
node_constraint, 9, 11, 14, 36, 37
node_degree (centrality), 8
node_eigenvector (centrality), 8
node_names (grab), 21
node_quad_census (census), 7
node_smallworld, 9, 11, 14, 37, 37
node_tie_census (census), 7
node_triad_census (census), 7

plot.blockmodel (blockmodel_vis), 6
print.blockmodel (blockmodel), 5
print.summary.netlm (regression), 41
project, 13, 25, 38, 44
project_cols (project), 38
project_rows (project), 38

read, 38
read_edgelist (read), 38
read_nodelist (read), 38
read_pajek (read), 38
read_ucinet (read), 38
reduce_graph (blockmodel), 5
regression, 41

southern_women, 42
summarise_statistics (blockmodel), 5
summary.netlm (regression), 41

test_permutation (tests), 42
test_random (tests), 42
tests, 42
to, 13, 25, 38, 43
to_main_component (to), 43
to_multilevel (to), 43
to_named (to), 43
to_onemode (to), 43
to_simplex (to), 43
to_undirected (to), 43
to_uniplex (to), 43
to_unnamed (to), 43
to_unsigned (to), 43
to_unweighted (to), 43

write_edgelist (read), 38
write_nodelist (read), 38
write_pajek (read), 38
write_ucinet (read), 38