Package 'manynet'

March 15, 2024

Title Many Ways to Make, Modify, Mark, and Map Myriad Networks

Version 0.4.4

Date 2024-03-15

Description
A set of tools for making, modifying, marking, and mapping many different types of networks.
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.
The package includes functions for importing and exporting, creating and generating networks,
modifying networks and node and tie attributes,
and describing and visualizing networks with sensible defaults.

URL `https://stocnet.github.io/manynet/`

BugReports `https://github.com/stocnet/manynet/issues`

Depends R (>= 3.6.0)

License MIT + file LICENSE

Language en-GB

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Imports dplyr (>= 1.1.0), ggplot2, ggraph, igraph (>= 1.6.0), network,
pillar, tidygraph

Suggests BiocManager, concaveman, gganimate, ggforce, gifski,
graphlayouts, grDevices, knitr, learnr, methods, migraph,
minMSE, multiplex, patchwork, png, readxl, rmarkdown, roxygen2,
RSiena, testthat (>= 3.0.0), xml2, future, furrr

Enhances Rgraphviz

Config/testthat/parallel true

Config/testthat/edition 3

Config/testthat/start-first mark_is

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>)

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

Repository CRAN

Date/Publication 2024-03-15 19:20:10 UTC

R topics documented:

`add_nodes` *Modifying node data*

Description

These functions allow users to add and delete nodes and their attributes:

- `add_nodes()` adds an additional number of nodes to network data.
- `delete_nodes()` deletes nodes from network data.
- `add_node_attribute()`, `mutate()`, or `mutate_nodes()` offer ways to add a vector of values to a network as a nodal attribute.
- `rename_nodes()` and `rename()` rename nodal attributes.
- `bind_node_attributes()` appends all nodal attributes from one network to another, and `join_nodes()` merges all nodal attributes from one network to another.
- `filter_nodes()` subsets nodes based on some nodal attribute-related logical statement.

Note that while `add_*()`/`delete_*()` functions operate similarly as comparable `{igraph}` functions, `mutate*()`, `bind*()`, etc work like `{tidyverse}` or `{dplyr}`-style functions.

Usage

```
add_nodes(.data, nodes, attribute = NULL)

delete_nodes(.data, nodes)

add_node_attribute(.data, attr_name, vector)

mutate_nodes(.data, ...)
```

```
mutate(.data, ...)

bind_node_attributes(.data, object2)

join_nodes(
  .data,
  object2,
  .by = NULL,
  join_type = c("full", "left", "right", "inner")
)

rename_nodes(.data, ...)

rename(.data, ...)

filter_nodes(.data, ..., .by)
```

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
`nodes`	The number of nodes to be added.
`attribute`	A named list to be added as tie or node attributes.
`attr_name`	Name of the new attribute in the resulting object.
`vector`	A vector of values for the new attribute.
`...`	Additional arguments.
`object2`	A second object to copy nodes or ties from.
`.by`	An attribute name to join objects by. By default, NULL.
`join_type`	A type of join to be used. Options are "full","left", "right", "inner".

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	igraph	network	tbl_graph
add_nodes	1	1	1
delete_nodes	1	0	0

Value

A data object of the same class as the function was given.

See Also

Other modifications: `add_ties()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
other <- create_filled(4) %>% mutate(name = c("A", "B", "C", "D"))
add_nodes(other, 4, list(name = c("Matthew", "Mark", "Luke", "Tim")))
other <- create_filled(4) %>% mutate(name = c("A", "B", "C", "D"))
another <- create_filled(3) %>% mutate(name = c("E", "F", "G"))
join_nodes(another, other)
```

`add_ties` *Modifying tie data*

Description

These functions allow users to add and delete ties and their attributes:

- `add_ties()` adds additional ties to network data
- `delete_ties()` deletes ties from network data
- `add_tie_attribute()` and `mutate_ties()` offer ways to add a vector of values to a network as a tie attribute.
- `rename_ties()` renames tie attributes.
- `bind_ties()` appends the tie data from two networks and `join_ties()` merges ties from two networks, adding a tie attribute identifying the newly added ties.
- `filter_ties()` subsets ties based on some tie attribute-related logical statement.

Note that while `add_*()`/`delete_*()` functions operate similarly as comparable `{igraph}` functions, `mutate*()`, `bind*()`, etc work like `{tidyverse}` or `{dplyr}`-style functions.

Usage

```
add_ties(.data, ties, attribute = NULL)

delete_ties(.data, ties)

add_tie_attribute(.data, attr_name, vector)

mutate_ties(.data, ...)

rename_ties(.data, ...)
```

```
bind_ties(.data, ...)

join_ties(.data, object2, attr_name)

filter_ties(.data, ...)

select_ties(.data, ...)

summarise_ties(.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
`ties`	The number of ties to be added or an even list of ties.
`attribute`	A named list to be added as tie or node attributes.
`attr_name`	Name of the new attribute in the resulting object.
`vector`	A vector of values for the new attribute.
`...`	Additional arguments.
`object2`	A second object to copy nodes or ties from.

Value

A tidygraph (`tbl_graph`) data object.

See Also

Other modifications: `add_nodes()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
other <- create_filled(4) %>% mutate(name = c("A", "B", "C", "D"))
mutate_ties(other, form = 1:6) %>% filter_ties(form < 4)
add_tie_attribute(other, "weight", c(1, 2, 2, 2, 1, 2))
```

as *Modifying network classes*

Description

The as_ functions in {manynet} coerce objects of any of the following common classes of social network objects in R into the declared class:

- as_edgelist() coerces the object into an edgelist, as data frames or tibbles.
- as_matrix() coerces the object into an adjacency (one-mode/unipartite) or incidence (two-mode/bipartite) matrix.
- as_igraph() coerces the object into an {igraph} graph object.
- as_tidygraph() coerces the object into a {tidygraph} tbl_graph objects.
- as_network() coerces the object into a {network} network objects.
- as_siena() coerces the (igraph/tidygraph) object into a SIENA dependent variable.
- as_graphAM() coerces the object into a graph adjacency matrix.
- as_diffusion() coerces a table of diffusion events into a diff_model object similar to the output of play_diffusion().

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.

Usage

```
as_edgelist(.data, twomode = FALSE)

as_matrix(.data, twomode = NULL)

as_igraph(.data, twomode = FALSE)

as_tidygraph(.data, twomode = FALSE)

as_network(.data, twomode = FALSE)

as_siena(.data, twomode = FALSE)

as_graphAM(.data, twomode = NULL)

as_diffusion(events, .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
`twomode`	Logical option used to override heuristics for distinguishing incidence (two-mode/bipartite) from adjacency (one-mode/unipartite) networks. By default FALSE.
`events`	A table (data frame or tibble) of diffusion events with columns `t` indicating the time (typically an integer) of the event, `nodes` indicating the number or name of the node involved in the event, and `event`, which can take on the values "I" for an infection event, "E" for an exposure event, or "R" for a recovery event.

Details

Edgelists are expected to be held in data.frame or tibble class objects. The first two columns of such an object are expected to be the senders and receivers of a tie, respectively, and are typically named "from" and "to" (even in the case of an undirected network). These columns can contain integers to identify nodes or character strings/factors if the network is labelled. If the sets of senders and receivers overlap, a one-mode network is inferred. If the sets contain no overlap, a two-mode network is inferred. If a third, numeric column is present, a weighted network will be created.

Matrices can be either adjacency (one-mode) or incidence (two-mode) matrices. 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:

	data.frame	diff_model	igraph	list	matrix	network	network.goldfish	siena	tbl_graph
as_edgelist	1	0	1	0	1	1	1	1	1
as_graphAM	1	0	1	0	1	1	1	1	1
as_igraph	1	0	1	0	1	1	1	1	1
as_matrix	1	0	1	0	1	1	1	1	1
as_network	1	0	1	0	1	1	1	1	1
as_siena	0	0	1	0	0	0	0	0	1
as_tidygraph	1	1	1	1	1	1	1	1	1

`as_diffusion()` and `play_diffusion()` return a 'diff_model' object that contains two different tibbles (tables) – a table of diffusion events and a table of the number of nodes in each relevant component (S, E, I, or R) – as well as a copy of the network upon which the diffusion ran. By default, a compact version of the component table is printed (to print all the changes at each time point, use `print(..., verbose = T)`). To retrieve the diffusion events table, use `summary(...)`.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
test <- data.frame(from = c("A","B","B","C","C"), to = c("I","G","I","G","H"))
as_edgelist(test)
as_matrix(test)
as_igraph(test)
as_tidygraph(test)
as_network(test)
  # How to create a diff_model object from (basic) observed data
 events <- data.frame(t = c(0,1,1,2,3), nodes = c(1,2,3,2,4), event = c("I","I","I","R","I"))
  as_diffusion(events, manynet::create_filled(4))
```

`attributes` *Describing attributes of nodes or ties in a network*

Description

These functions extract certain attributes from network data:

- `node_attribute()` returns an attribute's values for the nodes in a network.
- `node_names()` returns the names of the nodes in a network.
- `node_mode()` returns the mode of the nodes in a network.
- `tie_attribute()` returns an attribute's values for the ties in a network.
- `tie_weights()` returns the weights of the ties in a network.
- `tie_signs()` returns the signs of the ties in a network.

These functions are also often used as helpers within other functions. `node_*()` and `tie_*()` always return vectors the same length as the number of nodes or ties in the network, respectively.

Usage

```
node_attribute(.data, attribute)

node_names(.data)

node_mode(.data)

tie_attribute(.data, attribute)

tie_weights(.data)

tie_signs(.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
`attribute`	Character string naming an attribute in the object.

See Also

Other mapping: `autographr()`, `autographs()`, `autographt()`, `configuration_layouts`, `partition_layouts`, `properties`

Examples

```
node_attribute(ison_lotr, "Race")
node_names(ison_southern_women)
node_mode(ison_southern_women)
tie_attribute(ison_algebra, "task_tie")
tie_weights(to_mode1(ison_southern_women))
tie_signs(ison_marvel_relationships)
```

`autographr` *Easily graph networks with sensible defaults*

Description

This function provides users with an easy way to graph (m)any network data for exploration, investigation, and communication.

It builds upon `{ggplot2}` and `{ggraph}` to offer pretty and extensible graphing solutions. However, compared to those solutions, `autographr()` contains various algorithms to provide better looking graphs by default. This means that just passing the function some network data will often be sufficient to return a reasonable-looking graph.

The function also makes it easy to modify many of the most commonly adapted aspects of a graph, including node and edge size, colour, and shape, as arguments rather than additional functions that you need to remember. These can be defined outright, e.g. `node_size = 8`, or in reference to an attribute of the network, e.g. `node_size = "wealth"`.

Lastly, `autographr()` uses ggplot2-related theme information, so it is easy to make colour palette and fonts institution-specific and consistent. See e.g. `theme_iheid()` for more.

Usage

```
autographr(
  .data,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  node_group,
  edge_color,
  edge_size,
  ...
)

graphr(
  .data,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  node_group,
  edge_color,
  edge_size,
  ...
)
```

Arguments

`.data`	A manynet-consistent object.
`layout`	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress" for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.
`labels`	Logical, whether to print node names as labels if present.
`node_color`	Node variable to be used for coloring the nodes. It is easiest if this is added

	as a node attribute to the graph before plotting. Nodes can also be colored by declaring a color instead.
`node_shape`	Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.
`node_size`	Node variable to be used for sizing 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. Nodes can also be sized by declaring a numeric size or vector instead.
`node_group`	Node variable to be used for grouping the nodes. It is easiest if this is added as a hull over groups before plotting. Group variables should have a minimum of 3 nodes, if less, number groups will be reduced by merging categories with lower counts into one called "other".
`edge_color`	Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.
`edge_size`	Tie variable to be used for sizing the edges. 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 edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.
`...`	Extra arguments to pass on to the layout algorithm, if necessary.

Value

A `ggplot2::ggplot()` object. The last plot can be saved to the file system using `ggplot2::ggsave()`.

See Also

Other mapping: `attributes()`, `autographs()`, `autographt()`, `configuration_layouts`, `partition_layouts`, `properties`

Examples

```
autographr(ison_adolescents)
ison_adolescents |>
  mutate(color = rep(c("extrovert", "introvert"), times = 4),
         size = ifelse(node_is_cutpoint(ison_adolescents), 6, 3)) |>
  mutate_ties(ecolor = rep(c("friends", "aquaintances"), times = 5)) |>
  autographr(node_color = "color", node_size = "size",
             edge_size = 1.5, edge_color = "ecolor")
#autographr(ison_lotr, node_color = Race,
#           node_size = migraph::node_degree(ison_lotr)*2,
#           edge_color = "darkgreen",
#           edge_size = migraph::tie_degree(ison_lotr))
#autographr(ison_karateka, node_group = allegiance,
#           edge_size = migraph::tie_closeness(ison_karateka))
```

`autographs`	*Easily graph a set of networks with sensible defaults*

Description

This function provides users with an easy way to graph lists of network data for comparison.

It builds upon this package's `autographr()` function, and inherits all the same features and arguments. See `autographr()` for more. However, it uses the `{patchwork}` package to plot the graphs side by side and, if necessary, in successive rows. This is useful for lists of networks that represent, for example, ego or component subgraphs of a network, or a list of a network's different types of tie or across time. By default just the first and last network will be plotted, but this can be overridden by the "waves" parameter.

Where the graphs are of the same network (same nodes), the graphs may share a layout to facilitate comparison. By default, successive graphs will use the layout calculated for the "first" network, but other options include the "last" layout, or a mix, "both", of them.

Usage

```
autographs(netlist, waves, based_on = c("first", "last", "both"), ...)

graphs(netlist, waves, based_on = c("first", "last", "both"), ...)
```

Arguments

`netlist`	A list of manynet-compatible networks.
`waves`	Numeric, the number of plots to be displayed side-by-side. If missing, the number of plots will be reduced to the first and last when there are more than four plots. This argument can also be passed a vector selecting the waves to plot.
`based_on`	Whether the layout of the joint plots should be based on the "first" or the "last" network, or "both".
`...`	Additional arguments passed to `autographr()`.

Value

Multiple `ggplot2::ggplot()` objects displayed side-by-side.

See Also

Other mapping: `attributes()`, `autographr()`, `autographt()`, `configuration_layouts`, `partition_layouts`, `properties`

Examples

```
#autographs(to_egos(ison_adolescents))
#autographs(to_egos(ison_adolescents), waves = 8)
#autographs(to_egos(ison_adolescents), waves = c(2, 4, 6))
#autographs(play_diffusion(ison_adolescents))
```

autographt *Easily animate dynamic networks with sensible defaults*

Description

This function provides users with an easy way to graph dynamic network data for exploration and presentation.

It builds upon this package's `autographr()` function, and inherits all the same features and arguments. See `autographr()` for more. However, it uses the `{gganimate}` package to animate the changes between successive iterations of a network. This is useful for networks in which the ties and/or the node or tie attributes are changing.

A progress bar is shown if it takes some time to encoding all the .png files into a .gif.

Usage

```
autographt(
  tlist,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  edge_color,
  edge_size,
  keep_isolates = TRUE,
  ...
)

autographd(
  tlist,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  edge_color,
  edge_size,
  keep_isolates = TRUE,
  ...
)

grapht(
  tlist,
  layout,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  edge_color,
  edge_size,
  keep_isolates = TRUE,
  ...
)
```

Arguments

`tlist`	The same migraph-compatible network listed according to a time attribute, waves, or slices.
`layout`	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress" for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.
`labels`	Logical, whether to print node names as labels if present.
`node_color`	Node variable to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be colored by declaring a color instead.
`node_shape`	Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.
`node_size`	Node variable to be used for sizing 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. Nodes can also be sized by declaring a numeric size or vector instead.
`edge_color`	Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.
`edge_size`	Tie variable to be used for sizing the edges. 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 edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.

`keep_isolates`	Logical, whether to keep isolate nodes in the graph. TRUE by default. If FALSE, removes nodes from each frame they are isolated in.
`...`	Extra arguments to pass on to the layout algorithm, if necessary.

Value

Shows a .gif image. Assigning the result of the function saves the gif to a temporary folder and the object holds the path to this file.

Source

https://blog.schochastics.net/posts/2021-09-15_animating-network-evolutions-with-gganimate/

See Also

Other mapping: `attributes()`, `autographr()`, `autographs()`, `configuration_layouts`, `partition_layouts`, `properties`

Examples

```
#ison_adolescents %>%
#  mutate_ties(year = sample(1995:1998, 10, replace = TRUE)) %>%
#  to_waves(attribute = "year", cumulative = TRUE) %>%
#  autographd()
#ison_adolescents %>%
#  mutate(gender = rep(c("male", "female"), times = 4),
#         hair = rep(c("black", "brown"), times = 4),
#         age = sample(11:16, 8, replace = TRUE)) %>%
#  mutate_ties(year = sample(1995:1998, 10, replace = TRUE),
#               links = sample(c("friends", "not_friends"), 10, replace = TRUE),
#               weekly_meetings = sample(c(3, 5, 7), 10, replace = TRUE)) %>%
#  to_waves(attribute = "year") %>%
#  autographd(layout = "concentric", membership = "gender",
#             node_shape = "gender", node_color = "hair",
#             node_size =  "age", edge_color = "links",
#             edge_size = "weekly_meetings")
#autographd(play_diffusion(ison_adolescents, seeds = 5))
```

`configuration_layouts` *Layout algorithms based on configurational positions*

Description

Configurational layouts locate nodes at symmetric coordinates to help illustrate the particular layouts. Currently "triad" and "quad" layouts are available. The "configuration" layout will choose the appropriate configurational layout automatically.

Usage

```
layout_tbl_graph_configuration(.data, circular = FALSE, times = 1000)

layout_tbl_graph_triad(.data, circular = FALSE, times = 1000)

layout_tbl_graph_quad(.data, circular = FALSE, times = 1000)
```

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
`circular`	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
`times`	Maximum number of iterations, where appropriate

See Also

Other mapping: `attributes()`, `autographr()`, `autographs()`, `autographt()`, `partition_layouts`, `properties`

`create` *Making networks with defined structures*

Description

These functions create networks with particular structural properties.

- `create_empty()` creates an empty network without any ties.
- `create_filled()` creates a filled network with every possible tie realised.
- `create_ring()` creates a ring or chord network where each nodes' neighbours form a clique.
- `create_star()` creates a network with a maximally central node.
- `create_tree()` creates a network with successive branches.
- `create_lattice()` creates a network that forms a regular tiling.
- `create_components()` creates a network that clusters nodes into separate components.
- `create_core()` creates a network in which a certain proportion of 'core' nodes are densely tied to each other, and the rest peripheral, tied only to the core.
- `create_explicit()` creates a network based on explicitly named nodes and ties between them.

These functions can create either one-mode or two-mode networks. To create a one-mode network, pass the main argument `n` a single integer, indicating the number of nodes in the network. To create a two-mode network, pass `n` a vector of *two* integers, where the first integer indicates the number of nodes in the first mode, and the second integer indicates the number of nodes in the second mode. As an alternative, an existing network can be provided to `n` and the number of modes, nodes, and directedness will be inferred.

Usage

```
create_empty(n, directed = FALSE)

create_filled(n, directed = FALSE)

create_ring(n, directed = FALSE, width = 1, ...)

create_star(n, directed = FALSE)

create_tree(n, directed = FALSE, width = 2)

create_lattice(n, directed = FALSE, width = 8)

create_components(n, directed = FALSE, membership = NULL)

create_core(n, directed = FALSE, membership = NULL)

create_explicit(...)
```

Arguments

`n`	Given: • A single integer, e.g. `n = 10`, a one-mode network will be created. • A vector of two integers, e.g. `n = c(5,10)`, a two-mode network will be created. • A manynet-compatible object, a network of the same dimensions will be created.
`directed`	Logical whether the graph should be directed. By default `directed = FALSE`. If the opposite direction is desired, use `to_redirected()` on the output of these functions.
`width`	Integer specifying the width of the ring, breadth of the branches, or maximum extent of the neighbourbood.
`...`	Additional arguments passed on to `{igraph}`.
`membership`	A vector of partition membership as integers. If left as `NULL` (the default), nodes in each mode will be assigned to two, equally sized partitions.

Value

By default a `tbl_graph` object is returned, but this can be coerced into other types of objects using `as_edgelist()`, `as_matrix()`, `as_tidygraph()`, or `as_network()`.

By default, all networks are created as undirected. This can be overruled with the argument `directed = TRUE`. This will return a directed network in which the arcs are out-facing or equivalent. This direction can be swapped using `to_redirected()`. In two-mode networks, the directed argument is ignored.

Lattice graphs

`create_lattice()` creates both two-dimensional grid and triangular lattices with as even dimensions as possible. When the `width` parameter is set to 4, nodes cannot have (in or out) degrees larger than 4. This creates regular square grid lattices where possible. Such a network is bipartite, that is partitionable into two types that are not adjacent to any of their own type. If the number of nodes is a prime number, it will only return a chain (a single dimensional lattice).

A `width` parameter of 8 creates a network where the maximum degree of any nodes is 8. This can create a triangular mesh lattice or a Queen's move lattice, depending on the dimensions. A `width` parameter of 12 creates a network where the maximum degree of any nodes is 12. Prime numbers of nodes will return a chain.

See Also

as

`igraph::graph_from_literal()` which `create_explicit()` mostly just wraps. `create_explicit()` will also accept character input and not just a formula though, and will never simplify the result.

Other makes: `generate`, `learning`, `play`, `read`, `write()`

Examples

```
create_empty(10)
create_filled(10)
create_ring(8, width = 2)
create_star(12)
create_tree(c(7,8))
create_lattice(12, width = 4)
create_components(10, membership = c(1,1,1,2,2,2,3,3,3,3))
create_core(6)
  create_explicit(A -+ B, B -+ C, A +-+ C, D)
```

`features` *Marking networks features*

Description

These functions implement logical tests for various network features.

- `is_connected()` tests whether network is strongly connected, or weakly connected if undirected.
- `is_perfect_matching()` tests whether there is a matching for a network that covers every node in the network.

- `is_eulerian()` tests whether there is a Eulerian path for a network where that path passes through every tie exactly once.
- `is_acyclic()` tests whether network is a directed acyclic graph.
- `is_aperiodic()` tests whether network is aperiodic.

Usage

```
is_connected(.data)

is_perfect_matching(.data, mark = "type")

is_eulerian(.data)

is_acyclic(.data)

is_aperiodic(.data, max_path_length = 4)
```

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
`mark`	A logical vector marking two types or modes. By default "type".
`max_path_length`	Maximum path length considered. If negative, paths of all lengths are considered. By default 4, to avoid potentially very long computation times.

Value

TRUE if the condition is met, or FALSE otherwise.

is_connected

To test weak connection on a directed network, please see `to_undirected()`.

is_perfect_matching

For two-mode or bipartite networks, `to_matching()` is used to identify whether a perfect matching is possible. For one-mode networks, we use the Tutte theorem. Note that currently only subgraphs with cutpoints removed are tested, and not all possible subgraphs. This is to avoid computationally expensive combinatorial operations, but may come at the cost of some edge cases where a one-mode network cannot perfectly match as suggested.

Source

https://stackoverflow.com/questions/55091438/r-igraph-find-all-cycles

References

Tutte, W. T. (1950). "The factorization of locally finite graphs". *Canadian Journal of Mathematics*. 2: 44–49. doi:10.4153/cjm19500052

See Also

Other marking: `is()`, `is_format`

Examples

```
is_connected(ison_southern_women)
is_perfect_matching(ison_southern_women)
is_eulerian(ison_brandes)
is_acyclic(ison_algebra)
is_aperiodic(ison_algebra)
```

`from` *Joining lists of networks, graphs, and matrices*

Description

These functions offer tools for joining lists of manynet-consistent objects (matrices, igraph, tidygraph, or network objects) into a single object.

- `from_subgraphs()` modifies a list of subgraphs into a single tidygraph.
- `from_egos()` modifies a list of ego networks into a whole tidygraph
- `from_waves()` modifies a list of network waves into a longitudinal tidygraph.
- `from_slices()` modifies a list of time slices of a network into a dynamic tidygraph.
- `from_ties()` modifies a list of different ties into a multiplex tidygraph

Usage

```
from_subgraphs(netlist)

from_egos(netlist)

from_waves(netlist)

from_slices(netlist, remove.duplicates = FALSE)

from_ties(netlist, netnames)
```

Arguments

`netlist`	A list of network, igraph, tidygraph, matrix, or edgelist objects.
`remove.duplicates`	Should duplicates be removed? By default FALSE. If TRUE, duplicated edges are removed.
`netnames`	A character vector of names for the different network objects, if not already named within the list.

Value

A tidygraph object combining the list of network data.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
ison_adolescents %>%
  mutate(unicorn = sample(c("yes", "no"), 8, replace = TRUE)) %>%
  to_subgraphs(attribute = "unicorn") %>%
  from_subgraphs()
ison_adolescents %>%
  to_egos() %>%
  from_egos()
ison_adolescents %>%
  mutate_ties(wave = sample(1:4, 10, replace = TRUE)) %>%
  to_waves(attribute = "wave") %>%
  from_waves()
ison_adolescents %>%
  mutate_ties(time = 1:10, increment = 1) %>%
  add_ties(c(1,2), list(time = 3, increment = -1)) %>%
  to_slices(slice = c(5,7)) %>%
  from_slices()
```

`generate` *Making networks with a stochastic element*

Description

These functions are similar to the `create_*` functions, but include some element of randomisation. They are particularly useful for creating a distribution of networks for exploring or testing network properties.

- `generate_random()` generates a random network with ties appearing at some probability.
- `generate_smallworld()` generates a small-world structure via ring rewiring at some probability.

- `generate_scalefree()` generates a scale-free structure via preferential attachment at some probability.
- `generate_permutation()` generates a permutation of the network using a Fisher-Yates shuffle on both the rows and columns (for a one-mode network) or on each of the rows and columns (for a two-mode network).
- `generate_utilities()` generates a random utility matrix.

These functions can create either one-mode or two-mode networks. To create a one-mode network, pass the main argument `n` a single integer, indicating the number of nodes in the network. To create a two-mode network, pass `n` a vector of *two* integers, where the first integer indicates the number of nodes in the first mode, and the second integer indicates the number of nodes in the second mode. As an alternative, an existing network can be provided to `n` and the number of modes, nodes, and directedness will be inferred.

Usage

```
generate_random(n, p = 0.5, directed = FALSE, with_attr = TRUE)

generate_smallworld(n, p = 0.05, directed = FALSE, width = 2)

generate_scalefree(n, p = 1, directed = FALSE)

generate_permutation(.data, with_attr = TRUE)

generate_utilities(n, steps = 1, volatility = 0, threshold = 0)
```

Arguments

`n`	Given: • A single integer, e.g. `n = 10`, a one-mode network will be created. • A vector of two integers, e.g. `n = c(5,10)`, a two-mode network will be created. • A manynet-compatible object, a network of the same dimensions will be created.
`p`	Power of the preferential attachment, default is 1.
`directed`	Whether to generate network as directed. By default FALSE.
`with_attr`	Logical whether any attributes of the object should be retained. By default TRUE.
`width`	Integer specifying the width of the ring, breadth of the branches, or maximum extent of the neighbourbood.
`.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

`steps`	Number of simulation steps to run. By default 1: a single, one-shot simulation. If more than 1, further iterations will update the utilities depending on the values of the volatility and threshold parameters.
`volatility`	How much change there is between steps. Only if volatility is more than 1 do further simulation steps make sense. This is passed on to `stats::rnorm` as the `sd` or standard deviation parameter.
`threshold`	This parameter can be used to mute or disregard stepwise changes in utility that are minor. The default 0 will recognise all changes in utility, but raising the threshold will mute any changes less than this threshold.

Value

By default a `tbl_graph` object is returned, but this can be coerced into other types of objects using `as_edgelist()`, `as_matrix()`, `as_tidygraph()`, or `as_network()`.

By default, all networks are created as undirected. This can be overruled with the argument `directed = TRUE`. This will return a directed network in which the arcs are out-facing or equivalent. This direction can be swapped using `to_redirected()`. In two-mode networks, the directed argument is ignored.

References

Erdos, Paul, and Alfred Renyi. (1959). "On Random Graphs I" *Publicationes Mathematicae*. 6: 290–297.

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

Barabasi, Albert-Laszlo, and Reka Albert. 1999. “Emergence of Scaling in Random Networks.” *Science* 286(5439):509–12. doi:10.1126/science.286.5439.509.

See Also

Other makes: `create`, `learning`, `play`, `read`, `write()`

Examples

```
autographr(generate_random(12, 0.4))
# autographr(generate_random(c(6, 6), 0.4))
autographr(generate_smallworld(12, 0.025))
autographr(generate_smallworld(12, 0.25))
autographr(generate_scalefree(12, 0.25))
autographr(generate_scalefree(12, 1.25))
autographr(ison_adolescents)
autographr(generate_permutation(ison_adolescents))
```

`is` *Marking networks classes*

Description

These functions implement logical tests for networks' classes.

- `is_manynet()` marks a network TRUE if it is compatible with `{manynet}` functions.
- `is_edgelist()` marks a network TRUE if it is an edgelist.
- `is_graph()` marks a network TRUE if it contains graph-level information.
- `is_list()` marks a network TRUE if it is a (non-igraph) list of networks, for example a set of ego networks or a dynamic or longitudinal set of networks.
- `is_longitudinal()` marks a network TRUE if it contains longitudinal, panel data.
- `is_dynamic()` marks a network TRUE if it contains dynamic, time-stamped data

All `is_*()` functions return a logical scalar (TRUE or FALSE).

Usage

```
is_manynet(.data)

is_graph(.data)

is_edgelist(.data)

is_list(.data)

is_longitudinal(.data)

is_dynamic(.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

Value

TRUE if the condition is met, or FALSE otherwise.

See Also

Other marking: features, is_format

Examples

```
is_manynet(create_filled(2))
is_graph(create_star(2))
is_edgelist(matrix(c(2,2), 1, 2))
is_edgelist(as_edgelist(matrix(c(2,2), 1, 2)))
is_longitudinal(create_tree(5, 3))
is_dynamic(create_tree(3))
```

`ison_adolescents` *One-mode subset of the adolescent society network (Coleman 1961)*

Description

One-mode subset of Coleman's adolescent society network (Coleman 1961), as used in Feld's (1991) "Why your friends have more friends than you do". Coleman collected data on friendships among students in 12 U.S. high schools. Feld explored a subset of 8 girls from one of these schools, "Marketville", and gave them fictitious names, which are retained here.

Usage

```
data(ison_adolescents)
```

Format

```
#> # A labelled, undirected network with 8 nodes and 10 ties
#> # A tibble: 8 x 1
#>   name
#>   <chr>
#> 1 Betty
#> 2 Sue
#> 3 Alice
#> 4 Jane
#> 5 Dale
#> 6 Pam
#> # i 2 more rows
#> # A tibble: 10 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     2     3
#> 3     3     4
#> 4     2     5
#> 5     3     5
#> 6     4     5
#> # i 4 more rows
```

References

Coleman, James S. 1961. *The Adolescent Society*. New York: Free Press.

Feld, Scott. 1991. "Why your friends have more friends than you do" *American Journal of Sociology* 96(6): 1464-1477. doi:10.1086/229693.

`ison_algebra` *Multiplex graph object of friends, social, and task ties (McFarland 2001)*

Description

Multiplex graph object of friends, social, and task ties between 16 anonymous students. M182 was an honors algebra class where researchers collected friendship, social, and task ties between 16 students. The edge attribute `friends` contains friendship ties, where `2` = best friends, `1` = friend, and `0` is not a friend. `social` consists of social interactions per hour, and `tasks` consists of task interactions per hour.

Usage

```
data(ison_algebra)
```

Format

```
#> # A multiplex, weighted, directed network with 16 nodes and 279 arcs
#> # A tibble: 279 x 4
#>    from    to type   weight
#>   <int> <int> <chr>   <dbl>
#> 1     1     5 social   1.2
#> 2     1     5 tasks    0.3
#> 3     1     8 social   0.15
#> 4     1     9 social   2.85
#> 5     1     9 tasks    0.3
#> 6     1    10 social   6.45
#> # i 273 more rows
```

Source

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

References

McFarland, Daniel A. (2001) "Student Resistance." *American Journal of Sociology* 107(3): 612-78. doi:10.1086/338779.

`ison_brandes` *One-mode and two-mode centrality demonstration networks*

Description

This network should solely be used for demonstration purposes as it does not describe a real network. To convert into the two-mode version, assign `ison_brandes %>% rename(type = twomode_type)`.

Usage

```
data(ison_brandes)
```

Format

```
#> # A undirected network with 11 nodes and 12 ties
#> # A tibble: 11 x 1
#>   twomode_type
#>   <lgl>
#> 1 FALSE
#> 2 FALSE
#> 3 TRUE
#> 4 FALSE
#> 5 TRUE
#> 6 TRUE
#> # i 5 more rows
#> # A tibble: 12 x 2
#>    from    to
#>   <int> <int>
#> 1     1     3
#> 2     2     3
#> 3     3     4
#> 4     4     5
#> 5     4     6
#> 6     5     7
#> # i 6 more rows
```

`ison_friends` *One-mode Friends character connections (McNulty, 2020)*

Description

One-mode network collected by McNulty (2020) on the connections between the Friends TV series characters from Seasons 1 to 10. The `ison_friends` is a directed network containing connections between characters organised by season number, which is reflected in the tie attribute 'season'. The network contains 650 nodes Each tie represents the connection between a character pair (appear in the same scene), and the weight of the tie is the number of scenes the character pair appears in together. For all networks, characters are named (eg. Phoebe, Ross, Rachel).

Usage

```
data(ison_friends)
```

Format

```
#> # A labelled, multiplex, weighted, directed network with 650 nodes and 3959 arcs
#> # A tibble: 650 x 1
#>   name
#>   <chr>
#> 1 Actor
#> 2 Alan
#> 3 Andrea
#> 4 Angela
#> 5 Aunt Iris
#> 6 Aunt Lillian
#> # i 644 more rows
#> # A tibble: 3,959 x 4
#>    from    to season weight
#>   <int> <int>  <int>  <int>
#> 1     1    44      1      1
#> 2     2    14      1      1
#> 3     2    44      1      1
#> 4     2    58      1      2
#> 5     2    72      1      1
#> 6     2    75      1      1
#> # i 3,953 more rows
```

Details

The data contains both networks but each may be used separately.

References

McNulty, K. (2020). *Network analysis of Friends scripts.*.

`ison_hightech` *One-mode multiplex, directed network of managers of a high-tech company (Krackhardt 1987)*

Description

21 managers of a company of just over 100 employees manufactured high-tech equipment on the west coast of the United States. Three types of ties were collected:

- *friends*: managers' answers to the question "Who is your friend?"
- *advice*: managers' answers to the question "To whom do you go to for advice?"
- *reports*: "To whom do you report?" based on company reports

The data is anonymised, but four nodal attributes are included:

- *age*: the manager's age in years
- *tenure*: the manager's length of service
- *level*: the manager's level in the corporate hierarchy, where 3 = CEO, 2 = Vice President, and 1 = manager
- *dept*: one of four departments, B, C, D, E, with the CEO alone in A

Usage

```
data(ison_hightech)
```

Format

```
#> # A multiplex, directed network with 21 nodes and 312 arcs
#> # A tibble: 21 x 4
#>     age tenure level dept
#>   <dbl>  <dbl> <dbl> <chr>
#> 1    33      9     1 E
#> 2    42     20     2 E
#> 3    40     13     1 C
#> 4    33      8     1 E
#> 5    32      3     1 C
#> 6    59     28     1 B
#> # i 15 more rows
#> # A tibble: 312 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     2 friends
#> 2     1     2 advice
#> 3     1     2 reports
#> 4     1     4 friends
#> 5     1     4 advice
#> 6     1     8 friends
#> # i 306 more rows
```

References

Krackhardt, David. 1987. "Cognitive social structures". *Social Networks* 9: 104-134.

ison_karateka *One-mode karateka network (Zachary 1977)*

Description

The network was observed in a university Karate club in 1977. The network describes association patterns among 34 members and maps out allegiance patterns between members and either Mr. Hi, the instructor, or the John A. the club president after an argument about hiking the price for lessons. The allegiance of each node is listed in the `obc` argument which takes the value 1 if the individual sided with Mr. Hi after the fight and 2 if the individual sided with John A.

Usage

```
data(ison_karateka)
```

Format

```
#> # A labelled, weighted, undirected network with 34 nodes and 78 ties
#> # A tibble: 34 x 2
#>   name  allegiance
#>   <chr>      <dbl>
#> 1 Mr Hi          1
#> 2 2              1
#> 3 3              1
#> 4 4              1
#> 5 5              1
#> 6 6              1
#> # i 28 more rows
#> # A tibble: 78 x 3
#>    from    to weight
#>   <int> <int>  <dbl>
#> 1     1     2      4
#> 2     1     3      5
#> 3     2     3      6
#> 4     1     4      3
#> 5     2     4      3
#> 6     3     4      3
#> # i 72 more rows
```

References

Zachary, Wayne W. 1977. "An Information Flow Model for Conflict and Fission in Small Groups." *Journal of Anthropological Research* 33(4):452–73. doi:10.1086/jar.33.4.3629752.

`ison_koenigsberg` *One-mode Seven Bridges of Koenigsberg network (Euler 1741)*

Description

The Seven Bridges of Koenigsberg is a notable historical problem in mathematics and laid the foundations of graph theory. The city of Koenigsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges. A weekend diversion for inhabitants was to find a walk through the city that would cross each bridge once and only once. The islands could not be reached by any route other than the bridges, and every bridge must have been crossed completely every time (one could not walk half way onto the bridge and then turn around and later cross the other half from the other side). In 1735, Leonard Euler proved that the problem has no solution.

Usage

```
data(ison_koenigsberg)
```

Format

```
#> # A labelled, multiplex, undirected network with 4 nodes and 7 ties
#> # A tibble: 4 x 3
#>   name       lat   lon
#>   <chr>    <dbl> <dbl>
#> 1 Altstadt  54.7  20.5
#> 2 Kneiphof  54.7  20.5
#> 3 Lomse     54.7  20.5
#> 4 Vorstadt  54.7  20.5
#> # A tibble: 7 x 3
#>    from    to name
#>   <int> <int> <chr>
#> 1     1     2 Kraemer Bruecke
#> 2     1     2 Schmiedebruecke
#> 3     1     3 Holzbruecke
#> 4     2     3 Honigbruecke
#> 5     2     4 Gruene Bruecke
#> 6     2     4 Koettelbruecke
#> # i 1 more row
```

Source

{igraphdata}

References

Euler, Leonard. 1741. “Solutio problematis ad geometriam situs pertinentis.” *Commentarii academiae scientiarum Petropolitanae*.

`ison_laterals` *Two-mode projection examples (Hollway 2021)*

Description

These networks are for demonstration purposes and do not describe any real world network. All examples contain named nodes. The networks are gathered together as a list and can be retrieved simply by plucking the desired network.

Usage

```
data(ison_laterals)
```

Format

```
#> $ison_bb
#> # A labelled, two-mode network with 10 nodes and 12 ties
#> # A tibble: 10 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 U     TRUE
#> 3 B     FALSE
#> 4 V     TRUE
#> 5 C     FALSE
#> 6 W     TRUE
#> # i 4 more rows
#> # A tibble: 12 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     1     4
#> 3     3     2
#> 4     3     6
#> 5     3     7
#> 6     3     8
#> # i 6 more rows
#>
#> $ison_bm
#> # A labelled, two-mode network with 8 nodes and 9 ties
#> # A tibble: 8 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 U     TRUE
#> 3 B     FALSE
#> 4 V     TRUE
#> 5 C     FALSE
#> 6 W     TRUE
#> # i 2 more rows
#> # A tibble: 9 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     1     4
#> 3     3     2
#> 4     3     6
#> 5     3     7
#> 6     5     4
#> # i 3 more rows
#>
#> $ison_mb
#> # A labelled, two-mode network with 8 nodes and 9 ties
#> # A tibble: 8 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 M     TRUE
#> 3 B     FALSE
#> 4 C     FALSE
#> 5 X     TRUE
#> 6 Y     TRUE
#> # i 2 more rows
#> # A tibble: 9 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     3     2
#> 3     3     5
#> 4     3     6
#> 5     4     2
#> 6     4     5
#> # i 3 more rows
#>
#> $ison_mm
#> # A labelled, two-mode network with 6 nodes and 6 ties
#> # A tibble: 6 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 M     TRUE
#> 3 B     FALSE
#> 4 C     FALSE
#> 5 N     TRUE
#> 6 D     FALSE
#> # A tibble: 6 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     3     2
#> 3     3     5
#> 4     4     2
#> 5     4     5
#> 6     6     5
```

`ison_lawfirm` *One-mode lawfirm (Lazega 2001)*

Description

One-mode network dataset collected by Lazega (2001) on the relations between partners in a corporate law firm called SG&R in New England 1988-1991. This particular subset includes the 36 partners among the 71 attorneys of this firm. Nodal attributes include seniority, formal status, office in which they work, gender, lawschool they attended, their age, and how many years they had been at the firm.

Usage

```
data(ison_lawfirm)
```

Format

```
#> # A multiplex, directed network with 71 nodes and 2571 arcs
#> # A tibble: 71 x 7
#>   status  gender office   seniority   age practice   school
#>   <chr>   <chr>  <chr>        <dbl> <dbl> <chr>      <chr>
#> 1 partner man    Boston          31    64 litigation Harvard/Yale
#> 2 partner man    Boston          32    62 corporate  Harvard/Yale
#> 3 partner man    Hartford        13    67 litigation Harvard/Yale
#> 4 partner man    Boston          31    59 corporate  Other
#> 5 partner man    Hartford        31    59 litigation UConn
#> 6 partner man    Hartford        29    55 litigation Harvard/Yale
#> # i 65 more rows
#> # A tibble: 2,571 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     2 friends
#> 2     1     2 advice
#> 3     1     4 friends
#> 4     1     8 friends
#> 5     1    17 friends
#> 6     1    17 advice
#> # i 2,565 more rows
```

Details

The larger data from which this subset comes includes also individual performance measurements (hours worked, fees brought in) and attitudes concerning various management policy options (see also {sand}), their strong-coworker network, advice network, friendship network, and indirect control network.

Source

{networkdata}

References

Lazega, Emmanuel. 2001. *The Collegial Phenomenon: The Social Mechanisms of Cooperation Among Peers in a Corporate Law Partnership*. Oxford: Oxford University Press.

`ison_lotr` *One-mode network of Lord of the Rings character interactions*

Description

A network of 36 Lord of the Rings book characters and 66 interactional relationships. The ties are unweighted and concern only interaction. Interaction can be cooperative or conflictual.

Usage

```
data(ison_lotr)
```

Format

```
#> # A labelled, complex, undirected network with 36 nodes and 66 ties
#> # A tibble: 36 x 2
#>   name     Race
#>   <chr>    <chr>
#> 1 Aragorn  Human
#> 2 Beregond Human
#> 3 Bilbo    Hobbit
#> 4 Celeborn Elf
#> 5 Denethor Human
#> 6 Elladan  Elf
#> # i 30 more rows
#> # A tibble: 66 x 2
#>    from    to
#>   <int> <int>
#> 1     1     7
#> 2     1     8
#> 3     5     9
#> 4     1    10
#> 5     3    10
#> 6     9    10
#> # i 60 more rows
```

---

ison_marvel *Multilevel two-mode affiliation, signed one-mode networks of Marvel comic book characters (Yuksel 2017)*

Description

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 tie `sign` attribute, whereas enmities are indicated by a negative sign in this edge attribute.

Usage

```
data(ison_marvel_teams)

data(ison_marvel_relationships)
```

Format

```
#> # A labelled, two-mode network with 194 nodes and 683 ties
#> # A tibble: 194 x 2
#>   type  name
#>   <lgl> <chr>
#> 1 FALSE Abomination
#> 2 FALSE Ant-Man
#> 3 FALSE Apocalypse
#> 4 FALSE Beast
#> 5 FALSE Black Panther
#> 6 FALSE Black Widow
#> # i 188 more rows
#> # A tibble: 683 x 2
#>    from    to
#>   <int> <int>
#> 1     1   120
#> 2     1   152
#> 3     1   160
#> 4     1   162
#> 5     1   179
#> 6     2    56
#> # i 677 more rows
#> # A labelled, complex, multiplex, signed, undirected network with 53 nodes and 558 ties
#> # A tibble: 53 x 10
#>   name     Gender Appearances Attractive  Rich Intellect Omnilingual PowerOrigin
#>   <chr>    <chr>        <int>      <int> <int>     <int>       <int> <chr>
#> 1 Abomina~ Male           427          0     0         1           1 Radiation
#> 2 Ant-Man  Male           589          1     0         1           0 Human
#> 3 Apocaly~ Male          1207          0     0         1           1 Mutant
#> 4 Beast    Male          7609          1     0         1           0 Mutant
#> 5 Black P~ Male          2189          1     1         1           0 Human
#> 6 Black W~ Female        2907          1     0         1           0 Human
#> # i 47 more rows
#> # i 2 more variables: UnarmedCombat <int>, ArmedCombat <int>
#> # A tibble: 558 x 3
#>    from    to  sign
#>   <int> <int> <dbl>
#> 1     1     4    -1
#> 2     1    11    -1
#> 3     1    12    -1
#> 4     1    23    -1
#> 5     1    24    -1
#> 6     1    25    -1
#> # i 552 more rows
```

Details

Additional nodal variables have been coded and included by Dr Umut Yuksel:

- **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 Yuksel, 31 March 2017

`ison_monastery` *Three one-mode signed, weighted networks and a three-wave longitudinal network of monks (Sampson 1969)*

Description

The data were collected for an ethnographic study of community structure in a New England monastery. Various sociometric data was collected of the novices attending the minor seminary of 'Cloisterville' preparing to join the monastic order:

- `ison_monastery_like` records whom novices said they liked most at three time points/waves
- `ison_monastery_esteem` records whom novices said they held in esteem (sign > 0) and disesteem (sign < 0)
- `ison_monastery_praise` records whom novices said they praised (sign > 0) and blamed (sign < 0)
- `ison_monastery_influence` records whom novices said were a positive influence (sign > 0) and negative influence (sign < 0)

All networks are weighted. Novices' first choices are weighted 3, the second 2, and third choices 1. Some subjects offered tied ranks for their top four choices.

In addition to node names, a 'groups' variable records the four groups that Sampson observed during his time there:

- The *Loyal* Opposition consists of novices who entered the monastery first and defended existing practices
- The *Young Turks* arrived later during a period of change and questioned practices in the monastery
- The *Interstitial* did not take sides in the debate
- The *Outcasts* were novices that were not accepted in the group

Information about senior monks was not included. While `ison_monastery_like` is observed over three waves, the rest of the data was recorded retrospectively from the end of the study, after the network fragmented. The waves in which the novitiates were expelled (1), voluntarily departed (2 and 3), or remained (4) are given in the nodal attribute "left".

Usage

```
data(ison_monastery_like)

data(ison_monastery_esteem)

data(ison_monastery_influence)

data(ison_monastery_praise)
```

Format

```
#> # A longitudinal, labelled, multiplex, weighted, directed network with 18 nodes and 168 arcs
#> # A tibble: 18 x 3
#>   name        groups        left
#>   <chr>       <chr>        <dbl>
#> 1 Romuald     Interstitial     3
#> 2 Bonaventure Loyal            4
#> 3 Ambrose     Loyal            4
#> 4 Berthold    Loyal            4
#> 5 Peter       Loyal            3
#> 6 Louis       Loyal            4
#> # i 12 more rows
#> # A tibble: 168 x 4
#>    from    to weight  wave
#>   <int> <int>  <dbl> <dbl>
#> 1     1     5      3     1
#> 2     1     7      1     1
#> 3     1    11      2     1
#> 4     2     5      3     1
#> 5     2     6      2     1
#> 6     2    15      1     1
#> # i 162 more rows

#> # A labelled, signed, directed network with 18 nodes and 112 arcs
#> # A tibble: 18 x 3
#>   name        groups        left
#>   <chr>       <chr>        <dbl>
#> 1 Romuald     Interstitial     3
#> 2 Bonaventure Loyal            4
#> 3 Ambrose     Loyal            4
#> 4 Berthold    Loyal            4
#> 5 Peter       Loyal            3
#> 6 Louis       Loyal            4
#> # i 12 more rows
#> # A tibble: 112 x 3
#>    from    to  sign
#>   <int> <int> <dbl>
#> 1     2     3     1
#> 2     2     5     3
#> 3     2     6     2
#> 4     2    16    -1
#> 5     2    17    -3
#> 6     2    18    -2
#> # i 106 more rows

#> # A labelled, signed, directed network with 18 nodes and 103 arcs
#> # A tibble: 18 x 3
#>   name        groups        left
#>   <chr>       <chr>        <dbl>
#> 1 Romuald     Interstitial     3
#> 2 Bonaventure Loyal            4
#> 3 Ambrose     Loyal            4
#> 4 Berthold    Loyal            4
#> 5 Peter       Loyal            3
#> 6 Louis       Loyal            4
#> # i 12 more rows
#> # A tibble: 103 x 3
#>    from    to  sign
#>   <int> <int> <dbl>
#> 1     2     5     3
#> 2     2     6     2
#> 3     2    10     1
#> 4     2    16    -1
#> 5     2    17    -3
#> 6     2    18    -2
#> # i 97 more rows

#> # A labelled, signed, directed network with 18 nodes and 80 arcs
#> # A tibble: 18 x 3
#>   name        groups        left
#>   <chr>       <chr>        <dbl>
#> 1 Romuald     Interstitial     3
#> 2 Bonaventure Loyal            4
#> 3 Ambrose     Loyal            4
#> 4 Berthold    Loyal            4
#> 5 Peter       Loyal            3
#> 6 Louis       Loyal            4
#> # i 12 more rows
#> # A tibble: 80 x 3
#>    from    to  sign
#>   <int> <int> <dbl>
#> 1     4     3     1
#> 2     4     5     3
#> 3     4     6     2
#> 4     4    13    -3
#> 5     4    15    -1
#> 6     4    17    -2
#> # i 74 more rows
```

References

Sampson, Samuel F. 1969. *Crisis in a cloister*. Unpublished doctoral dissertation, Cornell University.

Breiger R., Boorman S. and Arabie P. 1975. "An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling". *Journal of Mathematical Psychology*, 12: 328-383.

ison_networkers *One-mode EIES dataset (Freeman and Freeman 1979)*

Description

A directed, simple, named, weighted graph with 32 nodes and 440 edges. Nodes are academics and edges illustrate the communication patterns on an Electronic Information Exchange System among them. Node attributes include the number of citations (`Citations`) and the discipline of the researchers (`Discipline`). Edge weights illustrate the number of emails sent from one academic to another over the studied time period.

Usage

```
data(ison_networkers)
```

Format

```
#> # A labelled, weighted, directed network with 32 nodes and 440 arcs
#> # A tibble: 32 x 3
#>   name               Discipline   Citations
#>   <chr>              <chr>            <dbl>
#> 1 Lin Freeman        Sociology           19
#> 2 Doug White         Anthropology         3
#> 3 Ev Rogers          Other              170
#> 4 Richard Alba       Sociology           23
#> 5 Phipps Arabie      Other               16
#> 6 Carol Barner-Barry Other                6
#> # i 26 more rows
#> # A tibble: 440 x 3
#>    from    to weight
#>   <int> <int>  <dbl>
#> 1     1     2    488
#> 2     1     3     28
#> 3     1     4     65
#> 4     1     5     20
#> 5     1     6     65
#> 6     1     7     45
#> # i 434 more rows
```

Source

networkdata package

References

Freeman, Sue C. and Linton C. Freeman. 1979. *The networkers network: A study of the impact of a new communications medium on sociometric structure*. Social Science Research Reports No 46. Irvine CA, University of California.

Wasserman Stanley and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge.

`ison_physicians` *Four multiplex one-mode physician diffusion data (Coleman, Katz, and Menzel, 1966)*

Description

Ron Burt prepared this data from Coleman, Katz and Menzel's 1966 study on medical innovation. They had collected data from physicians in four towns in Illinois: Peoria, Bloomington, Quincy and Galesburg. These four networks are held as separate networks in a list.

Coleman, Katz and Menzel were concerned with the impact of network ties on the physicians' adoption of a new drug, tetracycline. Data on three types of ties were collected in response to three questions:

- advice: "When you need information or advice about questions of therapy where do you usually turn?"
- discussion: "And who are the three or four physicians with whom you most often find yourself discussing cases or therapy in the course of an ordinary week – last week for instance?"
- friendship: "Would you tell me the first names of your three friends whom you see most often socially?"

Additional questions and records of prescriptions provided additional information:

- recorded date of tetracycline `adoption` date
- years in `practice` (note that these are `{messydates}`-compatible dates)
- `conferences` attended (those that attended "Specialty" conferences presumably also attended "General" conferences)
- regular subscriptions to medical `journals`
- `free_time` spent associating with doctors
- `discussions` on medical matters when with other doctors sociallyy
- memberships in `clubs` with other doctores
- number of top 3 `friends` that are doctors
- time practicing in current `community`
- `patients` load (ordinal)
- physical `proximity` to other physicians (in building/sharing office)
- medical `specialty` (GP/Internist/Pediatrician/Other)

Usage

```
data(ison_physicians)
```

Format

```
#> $Peoria
#> # A multiplex, directed network with 117 nodes and 543 arcs
#> # A tibble: 117 x 12
#>   adoption specialty    conferences journals practice   community patients
#>      <dbl> <chr>        <chr>          <dbl> <chr>      <chr>     <chr>
#> 1        1 Pediatrician Specialty          9 1920..1929 20+yrs    101-150
#> 2       12 GP           None               5 1945..     -1yr      76-100
#> 3        8 Internist    General            7 1935..1939 10-20yrs  76-100
#> 4        9 GP           General            6 1940..1944 5-10yrs   51-75
#> 5        9 GP           General            4 1935..1939 10-20yrs  51-75
#> 6       10 Internist    None               7 1930..1934 10-20yrs  101-150
#> # i 111 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#> # A tibble: 543 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     8 friendship
#> 2     1    58 friendship
#> 3     1    87 advice
#> 4     1    90 advice
#> 5     1   110 advice
#> 6     1   112 friendship
#> # i 537 more rows
#>
#> $Bloomington
#> # A multiplex, directed network with 50 nodes and 211 arcs
#> # A tibble: 50 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1       98 Internist Specialty          8 1930..1934 10-20yrs  101-150
#> 2        1 GP        General            3 1945..     5-10yrs   76-100
#> 3       98 GP        Specialty          4 1930..1934 10-20yrs  101-150
#> 4        7 Internist None               3 1945..     -1yr      26-50
#> 5        6 Internist General            9 1935..1939 5-10yrs   76-100
#> 6        1 GP        Specialty          5 1935..1939 10-20yrs  101-150
#> # i 44 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#> # A tibble: 211 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     3 friendship
#> 2     1    10 discussion
#> 3     1    24 advice
#> 4     1    44 advice
#> 5     2     4 advice
#> 6     2     6 advice
#> # i 205 more rows
#>
#> $Quincy
#> # A multiplex, directed network with 44 nodes and 174 arcs
#> # A tibble: 44 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1        2 Internist None               6 1935..1939 10-20yrs  151+
#> 2       18 GP        General            3 1920..1929 20+yrs    151+
#> 3       18 Internist None               5 1945..     -1yr      -25
#> 4        4 GP        General            3 1930..1934 20+yrs    151+
#> 5       18 GP        Specialty          4 1935..1939 10-20yrs  151+
#> 6        5 Internist General            5 ..1919     20+yrs    51-75
#> # i 38 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#> # A tibble: 174 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     8 advice
#> 2     1     9 advice
#> 3     1    10 discussion
#> 4     1    13 friendship
#> 5     1    15 advice
#> 6     1    22 discussion
#> # i 168 more rows
#>
#> $Galesburg
#> # A multiplex, directed network with 35 nodes and 171 arcs
#> # A tibble: 35 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1       18 GP        General            4 1935..1939 5-10yrs   101-150
#> 2       18 GP        None               4 1935..1939 -1yr      151+
#> 3        4 GP        General            6 1945..     2-5yrs    51-75
#> 4        5 GP        None               4 1935..1939 10-20yrs  101-150
#> 5        8 Internist General            6 1935..1939 5-10yrs   151+
#> 6        4 Internist Specialty          8 ..1919     20+yrs    76-100
#> # i 29 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#> # A tibble: 171 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     5 advice
#> 2     1     6 advice
#> 3     1    20 discussion
#> 4     1    23 discussion
#> 5     1    30 friendship
#> 6     1    31 friendship
#> # i 165 more rows
```

Source

{networkdata}

References

Coleman, James, Elihu Katz, and Herbert Menzel. 1966. *Medical innovation: A diffusion study*. Indianapolis: The Bobbs-Merrill Company.

---

ison_potter *Six complex one-mode support data in Harry Potter books (Bossaert and Meidert 2013)*

Description

Goele Bossaert and Nadine Meidert coded peer support ties among 64 characters in the Harry Potter books. Each author coded four of seven books using NVivo, with the seventh book coded by both and serving to assess inter-rater reliability. The first six books concentrated on adolescent interactions, were studied in their paper, and are made available here. The peer support ties mean voluntary emotional, instrumental, or informational support, or praise from one living, adolescent character to another within the book's pages. In addition, nodal attributes name, schoolyear (which doubles as their age), gender, and their house assigned by the sorting hat are included.

Usage

```
data(ison_potter)
```

Format

```
#> $book1
#> # A labelled, complex, directed network with 64 nodes and 47 arcs
#> # A tibble: 64 x 4
#>   name              schoolyear gender house
#>   <chr>                  <int> <chr>  <chr>
#> 1 Adrian Pucey            1989 male   Slytherin
#> 2 Alicia Spinnet          1989 female Gryffindor
#> 3 Angelina Johnson        1989 female Gryffindor
#> 4 Anthony Goldstein       1991 male   Ravenclaw
#> 5 Blaise Zabini           1991 male   Slytherin
#> 6 C. Warrington           1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 47 x 2
#>    from    to
#>   <int> <int>
#> 1    11    11
#> 2    11    25
#> 3    11    26
#> 4    11    44
#> 5    11    56
#> 6    11    58
#> # i 41 more rows
#>
#> $book2
#> # A labelled, complex, directed network with 64 nodes and 110 arcs
#> # A tibble: 64 x 4
#>   name              schoolyear gender house
#>   <chr>                  <int> <chr>  <chr>
#> 1 Adrian Pucey            1989 male   Slytherin
#> 2 Alicia Spinnet          1989 female Gryffindor
#> 3 Angelina Johnson        1989 female Gryffindor
#> 4 Anthony Goldstein       1991 male   Ravenclaw
#> 5 Blaise Zabini           1991 male   Slytherin
#> 6 C. Warrington           1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 110 x 2
#>    from    to
#>   <int> <int>
#> 1     2     2
#> 2     2     3
#> 3     2    19
#> 4     2    20
#> 5     2    25
#> 6     2    26
#> # i 104 more rows
#>
#> $book3
#> # A labelled, complex, directed network with 64 nodes and 104 arcs
#> # A tibble: 64 x 4
#>   name              schoolyear gender house
#>   <chr>                  <int> <chr>  <chr>
#> 1 Adrian Pucey            1989 male   Slytherin
#> 2 Alicia Spinnet          1989 female Gryffindor
#> 3 Angelina Johnson        1989 female Gryffindor
#> 4 Anthony Goldstein       1991 male   Ravenclaw
#> 5 Blaise Zabini           1991 male   Slytherin
#> 6 C. Warrington           1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 104 x 2
#>    from    to
#>   <int> <int>
#> 1     2     2
#> 2     2     3
#> 3     2    19
#> 4     2    20
#> 5     2    25
#> 6     2    26
#> # i 98 more rows
#>
#> $book4
#> # A labelled, complex, directed network with 64 nodes and 49 arcs
#> # A tibble: 64 x 4
#>   name              schoolyear gender house
#>   <chr>                  <int> <chr>  <chr>
#> 1 Adrian Pucey            1989 male   Slytherin
#> 2 Alicia Spinnet          1989 female Gryffindor
#> 3 Angelina Johnson        1989 female Gryffindor
#> 4 Anthony Goldstein       1991 male   Ravenclaw
#> 5 Blaise Zabini           1991 male   Slytherin
#> 6 C. Warrington           1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 49 x 2
#>    from    to
#>   <int> <int>
#> 1     7     7
#> 2     7     8
#> 3     7    25
#> 4     8     8
#> 5     8    25
#> 6     9     9
#> # i 43 more rows
#>
#> $book5
#> # A labelled, complex, directed network with 64 nodes and 160 arcs
#> # A tibble: 64 x 4
#>   name              schoolyear gender house
#>   <chr>                  <int> <chr>  <chr>
#> 1 Adrian Pucey            1989 male   Slytherin
#> 2 Alicia Spinnet          1989 female Gryffindor
#> 3 Angelina Johnson        1989 female Gryffindor
#> 4 Anthony Goldstein       1991 male   Ravenclaw
#> 5 Blaise Zabini           1991 male   Slytherin
#> 6 C. Warrington           1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 160 x 2
#>    from    to
#>   <int> <int>
#> 1     2     2
#> 2     2     3
#> 3     2    19
#> 4     2    20
#> 5     2    25
#> 6     2    29
#> # i 154 more rows
#>
#> $book6
#> # A labelled, complex, directed network with 64 nodes and 74 arcs
#> # A tibble: 64 x 4
#>   name                 schoolyear gender house
#>   <chr>                     <int> <chr>  <chr>
#> 1 Adrian Pucey               1989 male   Slytherin
#> 2 Alicia Spinnet             1989 female Gryffindor
#> 3 Angelina Johnson           1989 female Gryffindor
#> 4 Anthony Goldstein          1991 male   Ravenclaw
#> 5 Blaise Zabini              1991 male   Slytherin
#> 6 C. Warrington              1989 male   Slytherin
#> # i 58 more rows
#> # A tibble: 74 x 2
#>    from    to
#>   <int> <int>
#> 1    11    11
#> 2    11    25
#> 3    11    56
#> 4    11    58
#> 5    12    12
#> 6    14    14
#> # i 68 more rows
```

References

Bossaert, Goele and Nadine Meidert (2013). "'We are only as strong as we are united, as weak as we are divided'. A dynamic analysis of the peer support networks in the Harry Potter books." *Open Journal of Applied Sciences*, 3(2): 174-185. doi:10.4236/ojapps.2013.32024

`ison_southern_women` *Two-mode southern women (Davis, Gardner and Gardner 1941)*

Description

Two-mode network dataset collected by Davis, Gardner and Gardner (1941) about the attendance pattern of women at informal social events during a 9 month period. Events and women are named.

Usage

```
data(ison_southern_women)
```

Format

```
#> # A labelled, two-mode network with 32 nodes and 93 ties
#> # A tibble: 32 x 2
#>   type  name
#>   <lgl> <chr>
#> 1 FALSE Evelyn
#> 2 FALSE Laura
#> 3 FALSE Theresa
#> 4 FALSE Brenda
#> 5 FALSE Charlotte
#> 6 FALSE Frances
#> # i 26 more rows
#> # A tibble: 93 x 2
#>    from    to
#>   <int> <int>
#> 1     1    19
#> 2     1    20
#> 3     1    21
#> 4     1    22
#> 5     1    23
#> 6     1    24
#> # i 87 more rows
```

References

Davis, Allison, Burleigh B. Gardner, and Mary R. Gardner. 1941. *Deep South*. Chicago: University of Chicago Press.

`ison_starwars` *Seven one-mode Star Wars character interactions (Gabasova 2016)*

Description

One-mode network dataset collected by Gabasova (2016) on the interactions between Star Wars characters in each movie from Episode 1 (The Phantom Menace) to Episode 7 (The Force Awakens). There is a separate network for each episode, and the data is listed in order from episode 1 to 7. The network for each episode varies in the number of nodes and ties. For all networks, characters are named (eg. R2-D2, Anakin, Chewbacca) and the following node attributes are provided where available: height, mass, hair color, skin color, eye color, birth year, sex, homeworld, and species. The node attribute 'faction' has also been added, denoting the faction (eg. Jedi, Rebel Alliance, etc) that Star Wars characters belong to in each episode (coding completed with help of Yichen Shen and Tiphaine Aeby). Weighted ties represent the number of times characters speak within the same scene of the film.

Usage

```
data(ison_starwars)
```

Format

```
#> $`Episode I`
#> # A labelled, weighted, undirected network with 38 nodes and 135 ties
#> # A tibble: 38 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2      96    32 <NA>       white, bl~ red               33 none  Naboo
#> 2 QUI-G~    193    89 brown      fair       blue              92 male  <NA>
#> 3 NUTE ~    191    90 none      mottled g~ red              NA male  Cato Nei~
#> 4 PK-4       NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 5 TC-14      NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 6 OBI-W~    182    77 auburn, w~ fair      blue-gray         57 male  Stewjon
#> # i 32 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 135 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1    16     11
#> 2     1     2     14
#> 3     1    19     16
#> 4     1    18      3
#> 5     1    23      2
#> 6     1    25      2
#> # i 129 more rows
#>
#> $`Episode II`
#> # A labelled, weighted, undirected network with 33 nodes and 101 ties
#> # A tibble: 33 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2      96    32 <NA>       white, bl~ red               33 none  Naboo
#> 2 CAPTA~    185    85 black      dark       brown             NA male  Naboo
#> 3 EMPER~    170    75 grey       pale       yellow            82 male  Naboo
#> 4 SENAT~     NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 5 ORN F~     NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 6 MACE ~    188    84 none      dark      brown             72 male  Haruun K~
#> # i 27 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 101 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1    13      7
#> 2     1    12      7
#> 3     1    24      3
#> 4     3     4      2
#> 5     3     5      2
#> 6     4     5      1
#> # i 95 more rows
```

```
#>
#> $`Episode III`
#> # A labelled, weighted, undirected network with 24 nodes and 65 ties
#> # A tibble: 24 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>    <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2       96    32 <NA>       white, bl~ red             33   none  Naboo
#> 2 ANAKIN    188    84 blond      fair       blue           41.9 male  Tatooine
#> 3 OBI-W~    182    77 auburn, w~ fair       blue-gray       57   male  Stewjon
#> 4 ODD B~      NA    NA <NA>       <NA>       <NA>            NA   <NA>  <NA>
#> 5 GENER~    216   159 none       brown, wh~ green, y~       NA   male  Kalee
#> 6 EMPER~    170    75 grey       pale       yellow          82   male  Naboo
#> # i 18 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 65 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1     6      2
#> 2     1     3     12
#> 3     1     2      9
#> 4     1     9      5
#> 5     1     8      4
#> 6     1    10      4
#> # i 59 more rows
#>
#> $`Episode IV`
#> # A labelled, weighted, undirected network with 21 nodes and 60 ties
#> # A tibble: 21 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>    <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2       96    32 <NA>       white, bl~ red             33   none  Naboo
#> 2 CHEWB~    228   112 brown      unknown    blue           200   male  Kashyyyk
#> 3 C-3PO     167    75 <NA>       gold       yellow         112   none  Tatooine
#> 4 LUKE      172    77 blond      fair       blue            19   male  Tatooine
#> 5 DARTH~    202   136 none       white      yellow          41.9 male  Tatooine
#> 6 CAMIE      NA    NA <NA>       <NA>       <NA>            NA   <NA>  <NA>
#> # i 15 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 60 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1     2      3
#> 2     1     3     17
#> 3     1     9      1
#> 4     1     4     14
#> 5     1    10      1
#> 6     1    11      4
#> # i 54 more rows
```

```
#>
#> $`Episode V`
#> # A labelled, weighted, undirected network with 21 nodes and 55 ties
#> # A tibble: 21 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2      96    32 <NA>       white, bl~ red               33 none  Naboo
#> 2 CHEWB~    228   112 brown      unknown    blue             200 male  Kashyyyk
#> 3 LUKE      172    77 blond      fair       blue              19 male  Tatooine
#> 4 HAN       180    80 brown      fair       brown             29 male  Corellia
#> 5 RIEEK~     NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 6 LEIA      150    49 brown      light      brown             19 fema~ Alderaan
#> # i 15 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 55 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1     2      5
#> 2     1     7     10
#> 3     1     3      7
#> 4     1     4      4
#> 5     1     6      5
#> 6     1    21      1
#> # i 49 more rows
#>
#> $`Episode VI`
#> # A labelled, weighted, undirected network with 20 nodes and 55 ties
#> # A tibble: 20 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 R2-D2      96    32 <NA>       white, bl~ red             33   none  Naboo
#> 2 CHEWB~    228   112 brown      unknown    blue           200   male  Kashyyyk
#> 3 JERJE~     NA    NA <NA>       <NA>       <NA>            NA   <NA>  <NA>
#> 4 DARTH~    202   136 none       white      yellow          41.9 male  Tatooine
#> 5 C-3PO     167    75 <NA>       gold       yellow         112   none  Tatooine
#> 6 BIB F~    180    NA none       pale       pink            NA   male  Ryloth
#> # i 14 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 55 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1     2      8
#> 2     1     5     14
#> 3     1    12      2
#> 4     1     7      2
#> 5     1     8      8
#> 6     1    10      9
#> # i 49 more rows
```

```
#>
#> $`Episode VII`
#> # A labelled, weighted, undirected network with 27 nodes and 92 ties
#> # A tibble: 27 x 11
#>   name   height  mass hair_color skin_color eye_color birth_year sex   homeworld
#>   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 LUKE      172    77 blond      fair       blue              19 male  Tatooine
#> 2 R2-D2      96    32 <NA>       white, bl~ red               33 none  Naboo
#> 3 CHEWB~    228   112 brown      unknown    blue             200 male  Kashyyyk
#> 4 BB-8       NA    NA none       none       black             NA none  <NA>
#> 5 LOR S~     NA    NA <NA>       <NA>       <NA>              NA <NA>  <NA>
#> 6 POE        NA    NA brown      light      brown             NA male  <NA>
#> # i 21 more rows
#> # i 2 more variables: species <chr>, faction <chr>
#> # A tibble: 92 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     2     3      1
#> 2     2     4      2
#> 3     3     4      9
#> 4     2    18      2
#> 5     3     9     20
#> 6     3    11     13
#> # i 86 more rows
```

Details

The network for each episode may be extracted and used separately, eg. `ison_starwars[[1]]` or `ison_starwars$Episode I` for Episode 1.

References

Gabasova, E. (2016). *Star Wars social network.*. doi:10.5281/zenodo.1411479

`ison_usstates` *One-mode undirected network of US state contiguity (Meghanathan 2017)*

Description

This network is of contiguity between US states. States that share a border are connected by a tie in the network. The data is a network of 107 ties among 50 US states (nodes). States are named by their two-letter ISO-3166 code. This data includes also the names of the capitol cities of each state, which are listed in the node attribute 'capitol'.

Usage

```
data(ison_usstates)
```

Format

```
#> # A labelled, undirected network with 50 nodes and 107 ties
#> # A tibble: 50 x 2
#>   name  capitol
#>   <chr> <chr>
#> 1 AK    Juneau
#> 2 AL    Montgomery
#> 3 AR    Little Rock
#> 4 AZ    Phoenix
#> 5 CA    Sacramento
#> 6 CO    Denver
#> # i 44 more rows
#> # A tibble: 107 x 2
#>    from    to
#>   <int> <int>
#> 1     2     9
#> 2     2    10
#> 3     2    25
#> 4     2    42
#> 5     3    18
#> 6     3    24
#> # i 101 more rows
```

References

Meghanathan, Natarajan. 2017. "Complex network analysis of the contiguous United States graph." *Computer and Information Science*, 10(1): 54-76. doi:10.5539/cis.v10n1p54

`is_format` *Marking networks formats*

Description

These functions implement logical tests for various network properties. All `is_*()` functions return a logical scalar (TRUE or FALSE).

- `is_twomode()` marks networks TRUE if they contain two sets of nodes.
- `is_weighted()` marks networks TRUE if they contain tie weights.
- `is_directed()` marks networks TRUE if the ties specify which node is the sender and which the receiver.
- `is_labelled()` marks networks TRUE if there is a 'names' attribute for the nodes.
- `is_signed()` marks networks TRUE if the ties can be either positive or negative.
- `is_complex()` marks networks TRUE if any ties are loops, with the sender and receiver being the same node.
- `is_multiplex()` marks networks TRUE if it contains multiple types of ties, such that there can be multiple ties between the same sender and receiver.
- `is_uniplex()` marks networks TRUE if it is neither complex nor multiplex.

Usage

```
is_twomode(.data)

is_weighted(.data)

is_directed(.data)

is_labelled(.data)

is_signed(.data)

is_complex(.data)

is_multiplex(.data)

is_uniplex(.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

See Also

Other marking: `features`, `is()`

Examples

```
is_twomode(create_filled(c(2,2)))
is_weighted(create_tree(3))
is_directed(create_tree(2))
is_directed(create_tree(2, directed = TRUE))
is_labelled(create_empty(3))
is_signed(create_lattice(3))
is_complex(create_lattice(4))
is_multiplex(create_filled(c(3,3)))
is_uniplex(create_star(3))
```

learning — *Making learning models on networks*

Description

These functions allow learning games to be played upon networks.

- `play_learning()` plays a DeGroot learning model upon a network.
- `play_segregation()` plays a Schelling segregation model upon a network.

Usage

```
play_learning(.data, beliefs, steps, epsilon = 5e-04)

play_segregation(
  .data,
  attribute,
  heterophily = 0,
  who_moves = c("ordered", "random", "most_dissatisfied"),
  choice_function = c("satisficing", "optimising", "minimising"),
  steps
)
```

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
`beliefs`	A vector indicating the probabilities nodes put on some outcome being 'true'.
`steps`	The number of steps forward in learning. By default the number of nodes in the network.
`epsilon`	The maximum difference in beliefs accepted for convergence to a consensus.
`attribute`	A string naming some nodal attribute in the network. Currently only tested for binary attributes.
`heterophily`	A score ranging between -1 and 1 as a threshold for how heterophilous nodes will accept their neighbours to be. A single proportion means this threshold is shared by all nodes, but it can also be a vector the same length of the nodes in the network for issuing different thresholds to different nodes. By default this is 0, meaning nodes will be dissatisfied if more than half of their neighbours differ on the given attribute.

who_moves	One of the following options: "ordered" (the default) checks each node in turn for whether they are dissatisfied and there is an available space that they can move to, "random" will check a node at random, and "most_dissatisfied" will check (one of) the most dissatisfied nodes first.
choice_function	One of the following options: "satisficing" (the default) will move the node to any coordinates that satisfy their heterophily threshold, "optimising" will move the node to coordinates that are most homophilous, and "minimising" distance will move the node to the next nearest unoccupied coordinates.

See Also

Other makes: create, generate, play, read, write()

Other models: play

Examples

```
play_learning(ison_networkers,
    rbinom(manynet::network_nodes(ison_networkers),1,prob = 0.25))
startValues <- rbinom(100,1,prob = 0.5)
startValues[sample(seq_len(100), round(100*0.2))] <- NA
latticeEg <- create_lattice(100)
latticeEg <- add_node_attribute(latticeEg, "startValues", startValues)
latticeEg
play_segregation(latticeEg, "startValues", 0.5)
# autographr(latticeEg, node_color = "startValues", node_size = 5) +
# autographr(play_segregation(latticeEg, "startValues", 0.2),
#            node_color = "startValues", node_size = 5)
```

many_palettes *Many palettes generator*

Description

Many palettes generator

Usage

```
many_palettes(palette, n, type = c("discrete", "continuous"))
```

Arguments

palette	Name of desired palette. Current choices are: IHEID, Centres, SDGs, ETHZ, RUG, and UZH.
n	Number of colors desired. If omitted, uses all colours.
type	Either "continuous" or "discrete". Use continuous if you want to automatically interpolate between colours.

Value

A graphic display of colours in palette.

Source

Adapted from https://github.com/karthik/wesanderson/blob/master/R/colors.R

Examples

```
many_palettes()
#many_palettes("IHEID")
```

`mark_diff` *Marking nodes based on diffusion properties*

Description

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

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

Usage

```
node_is_latent(diff_model, time = 0)

node_is_infected(diff_model, time = 0)

node_is_recovered(diff_model, time = 0)

node_is_exposed(.data, mark)
```

Arguments

`diff_model`	A diff_model object, created either by `play_diffusion()` or `as_diffusion()`.
`time`	A time step at which nodes are identified.
`.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
`mark`	vector denoting which nodes are infected

Exposed

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

See Also

Other marks: `mark_nodes`, `mark_select`, `mark_tie_select`, `mark_ties`

Examples

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

---

mark_nodes *Marking nodes based on structural properties*

Description

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

- `node_is_cutpoint()` marks nodes that cut or act as articulation points in a network, increasing the number of connected components when removed.
- `node_is_isolate()` marks nodes that are isolates, with neither incoming nor outgoing ties.
- `node_is_core()` marks nodes that are members of the network's core.
- `node_is_fold()` marks nodes that are in a structural fold between two or more triangles that are only connected by that node.
- `node_is_mentor()` marks a proportion of high indegree nodes as 'mentors' (see details)

Usage

```
node_is_isolate(.data)

node_is_cutpoint(.data)

node_is_core(.data)

node_is_fold(.data)

node_is_mentor(.data, elites = 0.1)
```

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
`elites`	The proportion of nodes to be selected as mentors. By default this is set at 0.1. This means that the top 10% of nodes in terms of degree, or those equal to the highest rank degree in the network, whichever is the higher, will be used to select the mentors. Note that if nodes are equidistant from two mentors, they will choose one at random. If a node is without a path to a mentor, for example because they are an isolate, a tie to themselves (a loop) will be created instead. Note that this is a different default behaviour than that described in Valente and Davis (1999).

References

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

See Also

Other marks: `mark_diff`, `mark_select`, `mark_tie_select`, `mark_ties`

Examples

```
node_is_isolate(ison_brandes)
node_is_cutpoint(ison_brandes)
node_is_core(ison_brandes)
node_is_fold(create_explicit(A-B, B-C, A-C, C-D, C-E, D-E))
```

`mark_select` *Marking nodes for selection based on measures*

Description

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

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

Usage

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

node_is_max(node_measure, ranks = 1)

node_is_min(node_measure, ranks = 1)
```

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

See Also

Other marks: `mark_diff`, `mark_nodes`, `mark_tie_select`, `mark_ties`

Examples

```
node_is_random(ison_brandes, 2)
#node_is_max(migraph::node_degree(ison_brandes))
#node_is_min(migraph::node_degree(ison_brandes))
```

`mark_ties` *Marking ties based on structural properties*

Description

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

- `tie_is_multiple()` marks ties that are multiples.
- `tie_is_loop()` marks ties that are loops.
- `tie_is_reciprocated()` marks ties that are mutual/reciprocated.
- `tie_is_feedback()` marks ties that are feedback arcs causing the network to not be acyclic.
- `tie_is_bridge()` marks ties that cut or act as articulation points in a network.

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

Usage

```
tie_is_multiple(.data)

tie_is_loop(.data)

tie_is_reciprocated(.data)

tie_is_feedback(.data)

tie_is_bridge(.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

See Also

Other marks: `mark_diff`, `mark_nodes`, `mark_select`, `mark_tie_select`

Examples

```
tie_is_multiple(ison_marvel_relationships)
tie_is_loop(ison_marvel_relationships)
tie_is_reciprocated(ison_algebra)
tie_is_feedback(ison_algebra)
tie_is_bridge(ison_brandes)
```

`mark_tie_select` *Marking ties for selection based on measures*

Description

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

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

Usage

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

tie_is_max(tie_measure)

tie_is_min(tie_measure)
```

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
`size`	The number of nodes to select (as TRUE).
`tie_measure`	An object created by a `tie_` measure.

See Also

Other marks: `mark_diff`, `mark_nodes`, `mark_select`, `mark_ties`

Examples

```
# tie_is_max(migraph::tie_betweenness(ison_brandes))
#tie_is_min(migraph::tie_betweenness(ison_brandes))
```

`miss` *Modifying missing tie data*

Description

These functions offer tools for imputing missing tie data. Currently two options are available:

- `na_to_zero()` replaces any missing values with zeros, which are the modal value in sparse social networks.
- `na_to_mean()` replaces missing values with the average non-missing value.

Usage

```
na_to_zero(.data)

na_to_mean(.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

Value

A data object of the same class as the function was given.

References

Krause, Robert, Mark Huisman, Christian Steglich, and Tom A.B. Snijders. 2020. "Missing data in cross-sectional networks–An extensive comparison of missing data treatment methods". *Social Networks*, 62, 99-112.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
missTest <- ison_adolescents %>%
   add_tie_attribute("weight", c(1,NA,NA,1,1,1,NA,NA,1,1)) %>%
   as_matrix
missTest
na_to_zero(missTest)
na_to_mean(missTest)
```

`partition_layouts` *Layout algorithms based on bi- or other partitions*

Description

These algorithms layout networks based on two or more partitions, and are recommended for use with `autographr()` or `{ggraph}`. Note that these layout algorithms use `{Rgraphviz}`, a package that is only available on Bioconductor. It will first need to be downloaded using `BiocManager::install("Rgraphviz")`. If it has not already been installed, there is a prompt the first time these functions are used though.

The "hierarchy" layout layers the first node set along the bottom, and the second node set along the top, sequenced and spaced as necessary to minimise edge overlap. The "alluvial" layout is similar to "hierarchy", but places successive layers horizontally rather than vertically. The "railway" layout is similar to "hierarchy", but nodes are aligned across the layers. The "ladder" layout is similar to "railway", but places successive layers horizontally rather than vertically. The "concentric" layout places a "hierarchy" layout around a circle, with successive layers appearing as concentric circles. The "multilevel" layout places successive layers as multiple levels. The "lineage" layout ranks nodes in Y axis according to values.

Usage

```
layout_tbl_graph_hierarchy(
  .data,
  center = NULL,
  circular = FALSE,
  times = 1000
)

layout_tbl_graph_alluvial(.data, circular = FALSE, times = 1000)

layout_tbl_graph_railway(.data, circular = FALSE, times = 1000)

layout_tbl_graph_ladder(.data, circular = FALSE, times = 1000)

layout_tbl_graph_concentric(
  .data,
  membership,
  radius = NULL,
  order.by = NULL,
  circular = FALSE,
  times = 1000
)

layout_tbl_graph_multilevel(.data, level, circular = FALSE)

layout_tbl_graph_lineage(.data, rank, circular = FALSE)
```

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
`center`	Further split "hierarchical" layouts by declaring the "center" argument as the "events", "actors", or by declaring a node name in hierarchy layout. Defaults to NULL.
`circular`	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
`times`	Maximum number of iterations, where appropriate
`membership`	A node attribute or a vector to draw concentric circles for "concentric" layout.

`radius`	A vector of radii at which the concentric circles should be located for "concentric" layout. By default this is equal placement around an empty centre, unless one (the core) is a single node, in which case this node occupies the centre of the graph.
`order.by`	An attribute label indicating the (decreasing) order for the nodes around the circles for "concentric" layout. By default ordering is given by a bipartite placement that reduces the number of edge crossings.
`level`	A node attribute or a vector to hierarchically order levels for "multilevel" layout.
`rank`	A numerical node attribute to place nodes in Y axis according to values for "lineage" layout.

Source

Diego Diez, Andrew P. Hutchins and Diego Miranda-Saavedra. 2014. "Systematic identification of transcriptional regulatory modules from protein-protein interaction networks". *Nucleic Acids Research*, 42 (1) e6.

See Also

Other mapping: `attributes()`, `autographr()`, `autographs()`, `autographt()`, `configuration_layouts`, `properties`

Examples

```
#autographr(ison_southern_women, layout = "hierarchy", center = "events",
#           node_color = "type", node_size = 3)
#autographr(ison_southern_women, layout = "alluvial")
#autographr(ison_southern_women, layout = "concentric", membership = "type",
#           node_color = "type", node_size = 3)
#autographr(ison_lotr, layout = "multilevel",
#           node_color = "Race", level = "Race", node_size = 3)
# ison_adolescents %>%
#   mutate(year = rep(c(1985, 1990, 1995, 2000), times = 2),
#          cut = node_is_cutpoint(ison_adolescents)) %>%
#   autographr(layout = "lineage", rank = "year", node_color = "cut",
#              node_size = migraph::node_degree(ison_adolescents)*10)
```

`play` *Making diffusion models on networks*

Description

These functions simulate diffusion or compartment models upon a network.

- `play_diffusion()` runs a single simulation of a compartment model, allowing the results to be visualised and examined.
- `play_diffusions()` runs multiple simulations of a compartment model for more robust inference.

These functions allow both a full range of compartment models, as well as simplex and complex diffusion to be simulated upon a network.

Usage

```
play_diffusion(
  .data,
  seeds = 1,
  thresholds = 1,
  transmissibility = 1,
  latency = 0,
  recovery = 0,
  waning = 0,
  immune = NULL,
  steps
)

play_diffusions(
  .data,
  seeds = 1,
  thresholds = 1,
  transmissibility = 1,
  latency = 0,
  recovery = 0,
  waning = 0,
  immune = NULL,
  steps,
  times = 5,
  strategy = "sequential",
  verbose = FALSE
)
```

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
`seeds`	A valid mark vector the length of the number of nodes in the network.
`thresholds`	A numeric vector indicating the thresholds each node has. By default 1. A single number means a generic threshold; for thresholds that vary among the population please use a vector the length of the number of nodes in the network. If 1 or larger, the threshold is interpreted as a simple count of the number of contacts/exposures sufficient for infection. If less than 1, the threshold is interpreted as complex, where the threshold concerns the proportion of contacts.

`transmissibility`	The transmission rate probability, β. By default 1, which means any node for which the threshold is met or exceeded will become infected. Anything lower means a correspondingly lower probability of adoption, even when the threshold is met or exceeded.
`latency`	The inverse probability those who have been exposed become infectious (infected), σ or κ. For example, if exposed individuals take, on average, four days to become infectious, then $\sigma = 0.75$ (1/1-0.75 = 1/0.25 = 4). By default 0, which means those exposed become immediately infectious (i.e. an SI model). Anything higher results in e.g. a SEI model.
`recovery`	The probability those who are infected recover, γ. For example, if infected individuals take, on average, four days to recover, then $\gamma = 0.25$. By default 0, which means there is no recovery (i.e. an SI model). Anything higher results in an SIR model.
`waning`	The probability those who are recovered become susceptible again, ξ. For example, if recovered individuals take, on average, four days to lose their immunity, then $\xi = 0.25$. By default 0, which means any recovered individuals retain lifelong immunity (i.e. an SIR model). Anything higher results in e.g. a SIRS model. $\xi = 1$ would mean there is no period of immunity, e.g. an SIS model.
`immune`	A logical or numeric vector identifying nodes that begin the diffusion process as already recovered. This could be interpreted as those who are vaccinated or equivalent. Note however that a waning parameter will affect these nodes too. By default NULL, indicating that no nodes begin immune.
`steps`	The number of steps forward in the diffusion to play. By default the number of nodes in the network. If `steps = Inf` then the diffusion process will continue until there are no new infections or all nodes are infected.
`times`	Integer indicating number of simulations. By default `times=5`, but 1,000 - 10,000 simulations recommended for publication-ready results.
`strategy`	If `{furrr}` is installed, then multiple cores can be used to accelerate the simulations. 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.

Simple and complex diffusion

By default, the function will simulate a simple diffusion process in which some infectious disease or idea diffuses from seeds through contacts at some constant rate (`transmissibility`).

These `seeds` can be specified by a vector index (the number of the position of each node in the network that should serve as a seed) or as a logical vector where TRUE is interpreted as already infected.

`thresholds` can be set such that adoption/infection requires more than one (the default) contact already being infected. This parameter also accepts a vector so that thresholds can vary.

Complex diffusion is where the `thresholds` are defined less than one. In this case, the thresholds are interpreted as proportional. That is, the threshold to adoption/infection is defined by the proportion of the node's contacts infected.

Nodes that cannot be infected can be indicated as `immune` with a logical vector or index, similar to how `seeds` are identified. Note that `immune` nodes are interpreted internally as Recovered (R) and are thus subject to `waning` (see below).

Compartment models

Compartment models are flexible models of diffusion or contagion, where nodes are compartmentalised into one of two or more categories.

The most basic model is the SI model. The SI model is the default in `play_diffusion()`/`play_diffusions()`, where nodes can only move from the Susceptible (S) category to the Infected (I) category. Whether nodes move from S to I depends on whether they are exposed to the infection, for instance through a contact, the `transmissibility` of the disease, and their `thresholds` to the disease.

Another common model is the SIR model. Here nodes move from S to I, as above, but additionally they can move from I to a Recovered (R) status. The probability that an infected node recovers at a timepoint is controlled by the `recovery` parameter.

The next most common models are the SIS and SIRS models. Here nodes move from S to I or additionally to R, as above, but additionally they can move from I or R back to a Susceptible (S) state. This probability is governed by the `waning` parameter. Where `recover > 0` and `waning = 1`, the Recovery (R) state will be skipped and the node will return immediately to the Susceptible (S) compartment.

Lastly, these functions also offer the possibility of specifying a latency period in which nodes have been infected but are not yet infectious. Where `latency > 0`, an additional Exposed (E) compartment is introduced that governs the probability that a node moves from this E compartment to infectiousness (I). This can be used in in SEI, SEIS, SEIR, and SEIRS models.

See Also

Other makes: `create`, `generate`, `learning`, `read`, `write()`

Other models: `learning`

Examples

```
smeg <- generate_smallworld(15, 0.025)
plot(play_diffusion(smeg, recovery = 0.4))
#autographr(play_diffusion(ison_karateka))
plot(play_diffusions(smeg, times = 10))
```

`properties` *Describing network properties*

Description

These functions extract certain attributes from given network data:

- `network_nodes()` returns the total number of nodes (of any mode) in a network.
- `network_ties()` returns the number of ties in a network.

- `network_dims()` returns the dimensions of a network in a vector as long as the number of modes in the network.
- `network_node_attributes()` returns a vector of nodal attributes in a network.
- `network_tie_attributes()` returns a vector of tie attributes in a network.

These functions are also often used as helpers within other functions.

Usage

```
network_nodes(.data)

network_ties(.data)

network_dims(.data)

network_node_attributes(.data)

network_tie_attributes(.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

Value

`network_*()` functions always relate to the overall graph or network, usually returning a scalar. `network_dims()` returns an integer of the number of nodes in a one-mode network, or two integers representing the number of nodes in each nodeset in the case of a two-mode network. `network_*_attributes()` returns a string vector with the names of all node or tie attributes in the network.

See Also

Other mapping: `attributes()`, `autographr()`, `autographs()`, `autographt()`, `configuration_layouts`, `partition_layouts`

Examples

```
network_nodes(ison_southern_women)
network_ties(ison_southern_women)
network_dims(ison_southern_women)
network_dims(to_mode1(ison_southern_women))
  network_node_attributes(ison_lotr)
  network_tie_attributes(ison_algebra)
```

`read` *Making networks from external files*

Description

Researchers regularly need to work with a variety of external data formats. The following functions offer ways to import from some common external file formats into objects that `{manynet}` and other graph/network packages in R can work with:

- `read_matrix()` imports adjacency matrices from Excel/csv files.
- `read_edgelist()` imports edgelists from Excel/csv files.
- `read_nodelist()` imports nodelists from Excel/csv files.
- `read_pajek()` imports Pajek (.net or .paj) files.
- `read_ucinet()` imports UCINET files from the header (.##h).
- `read_dynetml()` imports DyNetML interchange format for rich social network data.
- `read_graphml()` imports GraphML files.

Usage

```
read_matrix(file = file.choose(), sv = c("comma", "semi-colon"), ...)

read_edgelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)

read_nodelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)

read_pajek(file = file.choose(), ties = NULL, ...)

read_ucinet(file = file.choose())

read_dynetml(file = file.choose())

read_graphml(file = file.choose())
```

Arguments

`file`	A character string with the system path to the file to import. If left unspecified, 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 and that it is currently not possible to import multiple networks from a single UCINET file. Please convert these one by one.
`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.
`ties`	A character string indicating the ties/network, where the data contains several.

Details

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 or if there are missing import routines by raising an issue on Github.

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

- UCINET data
- Pajek data
- 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

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

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

Source

`read_ucinet()` kindly supplied by Christian Steglich, constructed on 18 June 2015.

See Also

as

Other makes: `create`, `generate`, `learning`, `play`, `write()`

`reformat` *Modifying network formats*

Description

These functions reformat manynet-consistent data.

- `to_uniplex()` reformats multiplex network data to a single type of tie.
- `to_undirected()` reformats directed network data to an undirected network.
- `to_directed()` reformats undirected network data to a directed network.
- `to_redirected()` reformats the direction of directed network data, flipping any existing direction.
- `to_reciprocated()` reformats directed network data such that every directed tie is reciprocated.
- `to_acyclic()` reformats network data to an acyclic graph.
- `to_unweighted()` reformats weighted network data to unweighted network data.
- `to_unsigned()` reformats signed network data to unsigned network data.
- `to_unnamed()` reformats labelled network data to unlabelled network data.
- `to_named()` reformats unlabelled network data to labelled network data.
- `to_simplex()` reformats complex network data, containing loops, to simplex network data, without any loops.
- `to_anti()` reformats network data into its complement, where only ties *not* present in the original network are included in the new network.

If the format condition is not met, for example `to_undirected()` is used on a network that is already undirected, the network data is returned unaltered. No warning is given so that these functions can be used to ensure conformance.

Unlike the `as_*()` group of functions, these functions always return the same class as they are given, only transforming these objects' properties.

Usage

```
to_uniplex(.data, tie)

to_undirected(.data)

to_directed(.data)

to_redirected(.data)

to_reciprocated(.data)

to_acyclic(.data)
```

```
to_unweighted(.data, threshold = 1)

to_unsigned(.data, keep = c("positive", "negative"))

to_unnamed(.data)

to_named(.data, names = NULL)

to_simplex(.data)

to_anti(.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
`tie`	Character string naming a tie attribute to retain from a graph.
`threshold`	For a matrix, the threshold to binarise/dichotomise at.
`keep`	In the case of a signed network, whether to retain the "positive" or "negative" ties.
`names`	Character vector of the node names. NULL by default.

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	data.frame	igraph	matrix	network	tbl_graph
to_acyclic	1	1	1	1	1
to_directed	1	1	1	1	1
to_named	1	1	1	1	1
to_reciprocated	1	1	1	1	1
to_redirected	1	1	1	1	1
to_simplex	0	1	1	0	1
to_undirected	1	1	1	1	1
to_uniplex	1	1	1	1	1
to_unnamed	1	1	1	1	1
to_unsigned	1	1	1	1	1
to_unweighted	1	1	1	1	1

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 for each function.

Functions

- `to_undirected()`: Returns an object that has any edge direction removed, so that any pair of nodes with at least one directed edge will be connected by an undirected edge in the new network. This is equivalent to the "collapse" mode in `{igraph}`.
- `to_redirected()`: Returns an object that has any edge direction transposed, or flipped, so that senders become receivers and receivers become senders. This essentially has no effect on undirected networks or reciprocated ties.
- `to_reciprocated()`: Returns an object where all ties are reciprocated.
- `to_unweighted()`: Returns an object that has all edge weights removed.
- `to_unsigned()`: Returns a network with either just the "positive" ties or just the "negative" ties
- `to_unnamed()`: Returns an object with all vertex names removed
- `to_named()`: Returns an object that has random vertex names added
- `to_simplex()`: Returns an object that has all loops or self-ties removed

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `split()`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
as_tidygraph(create_filled(5)) %>%
  mutate_ties(type = sample(c("friend", "enemy"), 10, replace = TRUE)) %>%
  to_uniplex("friend")
to_anti(ison_southern_women)
#autographr(to_anti(ison_southern_women))
```

`scales` *Many scales*

Description

These functions enable to add color scales to be graphs.

Usage

```
scale_fill_iheid(direction = 1, ...)

scale_colour_iheid(direction = 1, ...)

scale_color_iheid(direction = 1, ...)

scale_edge_colour_iheid(direction = 1, ...)

scale_edge_color_iheid(direction = 1, ...)

scale_fill_centres(direction = 1, ...)

scale_colour_centres(direction = 1, ...)

scale_color_centres(direction = 1, ...)

scale_edge_colour_centres(direction = 1, ...)

scale_edge_color_centres(direction = 1, ...)

scale_fill_sdgs(direction = 1, ...)

scale_colour_sdgs(direction = 1, ...)

scale_color_sdgs(direction = 1, ...)

scale_edge_colour_sdgs(direction = 1, ...)

scale_edge_color_sdgs(direction = 1, ...)

scale_fill_ethz(direction = 1, ...)

scale_colour_ethz(direction = 1, ...)

scale_color_ethz(direction = 1, ...)

scale_edge_colour_ethz(direction = 1, ...)

scale_edge_color_ethz(direction = 1, ...)

scale_fill_uzh(direction = 1, ...)

scale_colour_uzh(direction = 1, ...)

scale_color_uzh(direction = 1, ...)

scale_edge_colour_uzh(direction = 1, ...)
```

```
scale_edge_color_uzh(direction = 1, ...)

scale_fill_rug(direction = 1, ...)

scale_colour_rug(direction = 1, ...)

scale_color_rug(direction = 1, ...)

scale_edge_colour_rug(direction = 1, ...)

scale_edge_color_rug(direction = 1, ...)
```

Arguments

`direction`	Direction for using palette colors.
`...`	Extra arguments passed to `ggplot2::discrete_scale()`.

Examples

```
#ison_brandes %>%
#mutate(core = migraph::node_is_core(ison_brandes)) %>%
#autographr(node_color = "core") +
#scale_color_iheid()
#autographr(ison_physicians[[1]], edge_color = "type") +
#scale_edge_color_ethz()
```

`split` *Splitting networks into lists*

Description

These functions offer tools for splitting manynet-consistent objects (matrices, igraph, tidygraph, or network objects) into lists of networks.

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	data.frame	diff_model	igraph	matrix	network	tbl_graph
to_components	1	0	1	1	1	1
to_egos	1	0	1	1	1	1
to_slices	0	0	1	0	0	1
to_subgraphs	0	0	1	0	1	1
to_waves	1	1	1	0	0	1

Usage

```
to_egos(.data, max_dist = 1, min_dist = 0)

to_subgraphs(.data, attribute)

to_components(.data)

to_waves(.data, attribute = "wave", panels = NULL, cumulative = FALSE)

to_slices(.data, attribute = "time", slice = NULL)
```

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
`max_dist`	The maximum breadth of the neighbourhood. By default 1.
`min_dist`	The minimum breadth of the neighbourhood. By default 0. Increasing this to 1 excludes the ego, and 2 excludes ego's direct alters.
`attribute`	One or two attributes used to slice data.
`panels`	Would you like to select certain waves? NULL by default. That is, a list of networks for every available wave is returned. Users can also list specific waves they want to select.
`cumulative`	Whether to make wave ties cumulative. FALSE by default. That is, each wave is treated isolated.
`slice`	Character string or character list indicating the date(s) or integer(s) range used to slice data (e.g slice = c(1:2, 3:4)).

Value

The returned object will be a list of network objects.

Functions

- `to_egos()`: Returns a list of ego (or focal) networks.
- `to_subgraphs()`: Returns a list of subgraphs on some given node attribute.
- `to_components()`: Returns a list of the components in a network.
- `to_waves()`: Returns a network with some discrete observations over time into a list of those observations.
- `to_slices()`: Returns a list of a network with some continuous time variable at some time slice(s).

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `reformat`, `to_levels`, `to_paths`, `to_project`, `to_scope`

Examples

```
  to_egos(ison_adolescents)
  #autographs(to_egos(ison_adolescents,2))
ison_adolescents %>%
  mutate(unicorn = sample(c("yes", "no"), 8,
                          replace = TRUE)) %>%
  to_subgraphs(attribute = "unicorn")
  to_components(ison_marvel_relationships)
ison_adolescents %>%
  mutate_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
  to_waves(attribute = "wave")
ison_adolescents %>%
  mutate_ties(time = 1:10, increment = 1) %>%
  add_ties(c(1,2), list(time = 3, increment = -1)) %>%
  to_slices(slice = 7)
```

`themes` *Many themes*

Description

These functions enable graphs to be easily and quickly themed, e.g. changing the default colour of the graph's vertices and edges.

Usage

```
theme_iheid(base_size = 12, base_family = "serif")

theme_ethz(base_size = 12, base_family = "sans")

theme_uzh(base_size = 12, base_family = "sans")

theme_rug(base_size = 12, base_family = "mono")
```

Arguments

`base_size`	Font size, by default 12.
`base_family`	Font family, by default "sans".

Examples

```
to_mentoring(ison_brandes) %>%
  mutate(color = c(rep(c(1,2,3), 3), 3)) %>%
  autographr(node_color = "color") +
  labs(title = "Who leads and who follows?") +
  scale_color_iheid() +
  theme_iheid()
```

---

to_levels *Modifying network levels*

Description

These functions reformat the levels in manynet-consistent network data.

- `to_onemode()` reformats two-mode network data into one-mode network data by simply removing the nodeset 'type' information. Note that this is not the same as `to_mode1()` or `to_mode2()`.
- `to_twomode()` reformats one-mode network data into two-mode network data, using a mark to distinguish the two sets of nodes.
- `to_multilevel()` reformats two-mode network data into multimodal network data, which allows for more levels and ties within modes.

If the format condition is not met, for example `to_onemode()` is used on a network that is already one-mode, the network data is returned unaltered. No warning is given so that these functions can be used to ensure conformance.

Unlike the `as_*()` group of functions, these functions always return the same class as they are given, only transforming these objects' properties.

Usage

```
to_onemode(.data)

to_twomode(.data, mark)

to_multilevel(.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
`mark`	A logical vector marking two types or modes. By default "type".

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	igraph	matrix	network	tbl_graph
to_multilevel	1	1	0	1
to_onemode	1	1	0	1
to_twomode	1	0	1	1

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 for each function.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_paths`, `to_project`, `to_scope`

`to_paths` *Modifying networks paths*

Description

These functions return tidygraphs containing only special sets of ties:

- `to_matching()` returns only the matching ties in some network data.
- `to_mentoring()` returns only ties to nodes' closest mentors.
- `to_eulerian()` returns only the Eulerian path within some network data.
- `to_tree()` returns the spanning tree in some network data or, if the data is unconnected, a forest of spanning trees.

Usage

```
to_matching(.data, mark = "type")

to_mentoring(.data, elites = 0.1)

to_eulerian(.data)

to_tree(.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
`mark`	A logical vector marking two types or modes. By default "type".
`elites`	The proportion of nodes to be selected as mentors. By default this is set at 0.1. This means that the top 10% of nodes in terms of degree, or those equal to the highest rank degree in the network, whichever is the higher, will be used to select the mentors. Note that if nodes are equidistant from two mentors, they will choose one at random. If a node is without a path to a mentor, for example because they are an isolate, a tie to themselves (a loop) will be created instead. Note that this is a different default behaviour than that described in Valente and Davis (1999).

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	data.frame	igraph	matrix	network	tbl_graph
to_eulerian	0	1	0	0	1
to_matching	1	1	1	1	1
to_mentoring	0	1	0	0	1

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 for each function.

`to_matching()`

`to_matching()` uses `{igraph}`'s `max_bipartite_match()` to return a network in which each node is only tied to one of its previous ties. The number of these ties left is its *cardinality*, and the algorithm seeks to maximise this such that, where possible, each node will be associated with just one node in the other mode or some other mark. The algorithm used is the push-relabel algorithm with greedy initialization and a global relabelling after every $\frac{n}{2}$ steps, where n is the number of nodes in the network.

References

Goldberg, A V; Tarjan, R E (1986). "A new approach to the maximum flow problem". *Proceedings of the eighteenth annual ACM symposium on Theory of computing – STOC '86*. p. 136.

doi:10.1145/12130.12144

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

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_project`, `to_scope`

Examples

```
to_matching(ison_southern_women)
#autographr(to_matching(ison_southern_women))
autographr(to_mentoring(ison_adolescents))
  to_eulerian(delete_nodes(ison_koenigsberg, "Lomse"))
  #autographr(to_eulerian(delete_nodes(ison_koenigsberg, "Lomse")))
```

`to_project` *Modifying networks projection*

Description

These functions offer tools for projecting manynet-consistent data:

- `to_mode1()` projects a two-mode network to a one-mode network of the first node set's (e.g. rows) joint affiliations to nodes in the second node set (columns).
- `to_mode2()` projects a two-mode network to a one-mode network of the second node set's (e.g. columns) joint affiliations to nodes in the first node set (rows).
- `to_ties()` projects a network to one where the ties become nodes and incident nodes become their ties.
- `to_galois()` projects a network to its Galois derivation.

Usage

```
to_mode1(.data, similarity = c("count", "jaccard", "rand", "pearson", "yule"))

to_mode2(.data, similarity = c("count", "jaccard", "rand", "pearson", "yule"))

to_ties(.data)

to_galois(.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
`similarity`	Method for establishing ties, currently "count" (default), "jaccard", or "rand". "count" calculates the number of coinciding ties, and can be interpreted as indicating the degree of opportunities between nodes. "jaccard" uses this count as the numerator in a proportion, where the denominator consists of any cell where either node has a tie. It can be interpreted as opportunity weighted by participation. "rand", or the Simple Matching Coefficient, is a proportion where the numerator consists of the count of cells where both nodes are present or both are absent, over all possible cells. It can be interpreted as the (weighted) degree of behavioral mirroring between two nodes. "pearson" (Pearson's coefficient) and "yule" (Yule's Q) produce correlations for valued and binary data, respectively. Note that Yule's Q has a straightforward interpretation related to the odds ratio.

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	data.frame	igraph	matrix	network	tbl_graph
to_mode1	1	1	1	1	1
to_mode2	1	1	1	1	1
to_ties	1	1	1	1	1

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 for each function.

Galois lattices

Note that the output from `to_galois()` is very busy at the moment.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_scope`

Examples

```
to_mode1(ison_southern_women)
```

```
to_mode2(ison_southern_women)
#autographr(to_mode1(ison_southern_women))
#autographr(to_mode2(ison_southern_women))
to_ties(ison_adolescents)
#autographr(to_ties(ison_adolescents))
```

`to_scope` *Modifying networks scope*

Description

These functions offer tools for transforming manynet-consistent objects (matrices, igraph, tidygraph, or network objects). Transforming means that the returned object may have different dimensions than the original object.

- `to_giant()` scopes a network into one including only the main component and no smaller components or isolates.
- `to_no_isolates()` scopes a network into one excluding all nodes without ties
- `to_subgraph()` scopes a network into a subgraph by filtering on some node-related logical statement.
- `to_blocks()` reduces a network to ties between a given partition membership vector.

Usage

```
to_giant(.data)

to_no_isolates(.data)

to_subgraph(.data, ...)

to_blocks(.data, membership, FUN = mean)
```

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
`...`	Arguments passed on to dplyr::filter
`membership`	A vector of partition memberships.
`FUN`	A function for summarising block content. By default `mean`. Other recommended options include `median`, `sum`, `min` or `max`.

Details

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:

	data.frame	igraph	list	matrix	network	tbl_graph
to_blocks	1	1	0	1	1	1
to_giant	1	1	0	1	1	1
to_no_isolates	1	1	1	1	1	1
to_subgraph	1	1	0	1	1	1

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 for each function.

`to_blocks()`

Reduced graphs provide summary representations of network structures by collapsing groups of connected nodes into single nodes while preserving the topology of the original structures.

See Also

Other modifications: `add_nodes()`, `add_ties()`, `as()`, `from`, `miss`, `reformat`, `split()`, `to_levels`, `to_paths`, `to_project`

Examples

```
ison_adolescents %>%
  mutate_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
  to_waves(attribute = "wave") %>%
  to_no_isolates()
```

`tutorials` *Open and extract code from tutorials*

Description

These functions make it easy to use the tutorials in the `{manynet}` and `{migraph}` packages:

- `run_tute()` runs a `{learnr}` tutorial from either the `{manynet}` or `{migraph}` packages, wraps `learnr::run_tutorial()` with some convenience.
- `extract_tute()` extracts and opens just the solution code from a `{manynet}` or `{migraph}` tutorial, saving the .R script to the current working directory.
- `pkg_data()` returns a tibble with details of the network datasets included in the packages.

Usage

```
run_tute(tute)

extract_tute(tute)

pkg_data(pkg = "manynet")
```

Arguments

`tute`	String, name of the tutorial (e.g. "tutorial2").
`pkg`	String, name of the package.

Examples

```
#run_tute("tutorial2")
#extract_tute("tutorial2")
#pkg_data()
# to obtain overview of unique datasets:
 #pkg_data() %>%
  #dplyr::distinct(directed, weighted, twomode, signed,
   #               .keep_all = TRUE)
```

`write` *Making networks to external files*

Description

Researchers may want to save or work with networks outside R. The following functions offer ways to export to some common external file formats:

- `write_matrix()` exports an adjacency matrix to a .csv file.
- `write_edgelist()` exports an edgelist to a .csv file.
- `write_nodelist()` exports a nodelist to a .csv file.
- `write_pajek()` exports Pajek .net files.
- `write_ucinet()` exports a pair of UCINET files in V6404 file format (.##h, .##d).
- `write_graphml()` exports GraphML files.

Usage

```
write_matrix(.data, filename, ...)

write_edgelist(.data, filename, ...)

write_nodelist(.data, filename, ...)

write_pajek(.data, filename, ...)
```

```
write_ucinet(.data, filename, name)

write_graphml(.data, filename, ...)
```

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
`filename`	Character string filename. If missing, the files will have the same name as the object and be saved to the working directory. An appropriate extension will be added if not included.
`...`	Additional parameters passed to the write function.
`name`	Character string to name the network internally, e.g. in UCINET. By default the name will be the same as the object.

Details

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 or if there are missing import routines by raising an issue on Github.

Value

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

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

Source

`write_ucinet()` kindly supplied by Christian Steglich, constructed on 18 June 2015.

See Also

as

Other makes: `create`, `generate`, `learning`, `play`, `read`

Index