Package ‘manynet’

November 2, 2023

Title Many Ways to Make, Manipulate, and Map Myriad Networks
Version 0.2.8
Date 2023-11-02

Description A set of tools for making, manipulating, 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, molding and manipulating networks and node and tie attributes, and describing and visualizing networks with sensible defaults.

URL https://snlab-ch.github.io/manynet/

BugReports https://github.com/snlab-ch/manynet/issues

Depends R (>= 3.6.0)
License MIT + file LICENSE
Language en-GB
Encoding UTF-8
LazyData true

RoxygenNote 7.2.3

Imports dplyr (>= 1.1.0), ggplot2, ggraph, igraph (>= 1.5.0.1), network, tidygraph

Suggests gganimate, methods, readxl, roxygen2, RSiena, testthat, xml2, patchwork, rmarkdown, learner, grDevices, BiocManager, minMSE, graphlayouts, ggforce, multiplex, concaveman, migraph, knitr

Enhances Rgraphviz

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 2023-11-02 10:10:08 UTC
R topics documented:

add ...................................................... 2
as ....................................................... 4
attributes ............................................. 6
autographing .......................................... 7
configuration_layouts ............................... 10
create .................................................. 11
from ..................................................... 13
generate ............................................... 14
iheidPalette ........................................... 16
is .......................................................... 17
ison_adolescents ...................................... 20
ison_algebra ........................................... 21
ison_brandes ........................................... 22
ison_friends ............................................ 22
ison_karateka ......................................... 23
ison_konigsberg ....................................... 24
ison_laterals .......................................... 25
ison_lawfirm .......................................... 27
ison_lotr ............................................... 28
ison_marvel ............................................ 29
ison_networkers ....................................... 31
ison_southern_women ................................ 32
ison_starwars ......................................... 33
miss ..................................................... 37
partition_layouts ..................................... 38
properties ............................................. 40
read ..................................................... 41
reformat .............................................. 43
scales .................................................. 46
split ................................................... 47
themes ................................................ 49
tidy ..................................................... 49
transform ............................................. 51
tutorials ............................................. 54

Index 56

Description

These functions allow users to add nodes, ties, or attributes to the nodes or ties of a network. The add_*() functions operate similarly to in \{igraph\}. Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:
### add

<table>
<thead>
<tr>
<th></th>
<th>igraph</th>
<th>network</th>
<th>tbl_graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_nodes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>add_ties</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>delete_nodes</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Usage

- `add_nodes(.data, nodes, attribute = NULL)`
- `delete_nodes(.data, nodes)`
- `add_ties(.data, ties, attribute = NULL)`
- `add_node_attribute(.data, attr_name, vector)`
- `add_tie_attribute(.data, attr_name, vector)`

#### 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.

- **ties**
  The number of ties to be added or an even list of ties.

- **attr_name**
  Name of the new attribute in the resulting object.

- **vector**
  A vector of values for the new attribute.

#### Value

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

#### Functions

- `add_nodes()`: Add additional nodes to a network
- `delete_nodes()`: Delete nodes in a network
- `add_ties()`: Add additional ties to a network
- `add_node_attribute()`: Add a vector of values to a network as a nodal attribute.
- `add_tie_attribute()`: Add a vector of values to a network as a tie attribute.
See Also

Other manipulations: from, miss, reformat, split(), tidy, transform()

Examples

other <- create_filled(4) %>% mutate(name = c("A", "B", "C", "D"))
add_nodes(other, 4, list(name = c("Matthew", "Mark", "Luke", "Tim")))
add_tie_attribute(other, "weight", c(1, 2, 2, 1, 2))

Description

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

- edgelists, as data frames or tibbles
- adjacency (one-mode/unipartite) and incidence (two-mode/bipartite) matrices
- {igraph} graph objects
- {tidygraph} tbl_graph objects
- {network} network objects

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)
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.

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:

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

Functions

• as_edgelist(): Coercing various network objects into an edgelist

See Also

Other makes: create, generate, read
**Examples**

```r
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)
```

---

**attributes**

*Describing attributes of nodes or ties in a network*

---

**Description**

These functions extract certain attributes from given network data. They are also useful as helpers within other functions.

**Usage**

- `node_names(.data)`
- `node_mode(.data)`
- `node_attribute(.data, attribute)`
- `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.

**Value**

`node_*()` and `tie_*()` always return vectors the same length as the number of nodes or ties in the network, respectively.
Functions

- `node_names()`: Extracts the names of the nodes in a network.
- `node_mode()`: Extracts the mode of the nodes in a network.
- `node_attribute()`: Extracts an attribute’s values for the nodes in a network.
- `tie_attribute()`: Extracts an attribute’s values for the edges in a network.
- `tie_weights()`: Extracts the weights of the edges in a network.
- `tie_signs()`: Extracts the signs of the edges in a network.

See Also

Other mapping: `autographing`, `configuration_layouts`, `is()`, `partition_layouts`, `properties`

Examples

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

Description

The aim of this function is to provide users with a quick and easy graphing function that makes best use of the data, whatever its composition. Users can also tailor the plot according to their preferences regarding node size, colour, and shape. The function also supports visualisation of network measures such as centrality.

Usage

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

autographd(
  tlist,
  keep_isolates = TRUE,
  layout = "stress",
  labels = TRUE,
  node_color = NULL,
  node_shape = NULL,
  node_size = NULL,
  edge_color = NULL
)

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 "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.

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".

tie_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.

tie_size Edge 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 layout.

Extra arguments to pass on to layout.

data = NULL
netlist = A list of manynet-compatible networks.
tlist = The same migraph-compatible network listed according to a time attribute, waves,
or slices.

keep_isolates = Would you like to remove vertices that do not have any adjacent edges in each
frame? TRUE by default. If FALSE, deletes isolated vertices in each frame.

Value

A ggplot2::ggplot() object.

Functions

• autographr(): Graphs a network with sensible defaults
• autographs(): Graphs a list of networks with sensible defaults
• autographd(): Graphs an dynamic (animated) network with sensible defaults

Source

http://blog.schochastics.net/post/animating-network-evolutions-with-gganimate/

See Also

Other mapping: attributes(), configuration_layouts, is(), partition_layouts, properties

Examples

#autographr(ison_adolescents)
#autographr(ison_algebra, layout = "circle",
#    node_size = 8, node_color = "orange", node_shape = "square",
#    edge_color = "blue", edge_size = 2)
#autographr(ison_algebra, edge_color = "friends",
#    node_size = migraph::node_betweenness(ison_algebra)*100)
#ison_adolescents |> |
#  mutate(cut = migraph::node_is_cutpoint(ison_adolescents)) |> |
#  autographr(node_color = "cut", node_shape = "cut")
#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 = obc,
#    edge_size = migraph::tie_closeness(ison_karateka))
#autographr(ison_southern_women, layout = "concentric",
#    node_color = "type", membership = "type")
#autographr(ison_lotr, layout = "multilevel",


configuration_layouts

Layout algorithms based on configurational positions

Description

Layout algorithms based on configurational positions

Usage

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(), autographing.is(), partition_layouts.properties
create | Make networks with defined structures

**Description**

These functions create networks with particular structural properties. They 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.

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.

**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.
create

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().

Functions

- create_empty(): Creates an empty graph of the given dimensions.
- create_filled(): Creates a filled graph of the given dimensions, with every possible tie realised.
- create_ring(): Creates a ring or chord graph of the given dimensions that loops around is of a certain width or thickness.
- create_star(): Creates a graph of the given dimensions that has a maximally central node.
- create_tree(): Creates a graph of the given dimensions with successive branches.
- create_lattice(): Creates a lattice graph of the given dimensions with ties to all neighbouring nodes.
- create_components(): Creates a graph in which the nodes are clustered into separate components.
- create_core(): Creates a graph with a certain proportion of nodes being core nodes, densely tied to each other and peripheral nodes, and the rest peripheral, tied only to the core.
- create_explicit(): Creates a network based on explicitly named nodes and ties between them.

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: as(), generate, read

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)

Description

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

Usage

from_subgraphs(.data)
from_egos(.data)
from_waves(.data)
from_slices(.data, remove.duplicates = 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
.remove.duplicates Should duplicates be removed? By default FALSE. If TRUE, duplicated edges are removed.
Value

A tidygraph object combining the list of network data.

Functions

- `from_subgraphs()`: Returns a single network object from a list of subgraphs.
- `from_egos()`: Returns a single network object from a list of egos.
- `from_waves()`: Returns a single network object from a list of waves.
- `from_slices()`: Returns a single network object from a list of slices.

See Also

Other manipulations: `add, miss, reformat, split, tidy, transform()`

Examples

```r
ison_adolescents %>%
mutate(unicorn = sample(c("yes", "no"), 8, replace = TRUE)) %>%
to_subgraphs(attribute = "unicorn") %>%
from_subgraphs()
ison_adolescents %>%
activate(edges) %>%
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()
```

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. Note that passing the first argument an empirical network will prompt these functions to generate a network with the same dimensions.
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 neighbourhood.

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

- `generate_random()`: Generates a random network with a particular probability.
- `generate_smallworld()`: Generates a small-world structure following the lattice rewiring model.
- `generate_scalefree()`: Generates a scale-free structure following the preferential attachment model.
- `generate_permutation()`: Generates a permutation of the original 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 utility matrix

References


See Also

Other makes: `as()`, `create`, `read`

Examples

```r
autographr(generate_random(12, 0.4))
# autographr(generate_random(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))
```

```
iheid_palette  An IHEID palette generator
```

Description

These are a few color palettes useful for members of the Geneva Graduate Institute. This function calls palettes for the Institute, for the Centres, and for the SDGs.

Usage

```r
iheid_palette(palette, n, type = c("discrete", "continuous"))
```
is

Arguments

- **palette**: Name of desired palette. Current choices are: IHEID, Centres, and SDGs.
- **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 vector of colours.

Source


Examples

```r
# iheid_palette("IHEID")
```

---

is  

Describing network formats

Description

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

Usage

- `is_manynet(.data)`
- `is_graph(.data)`
- `is_edgelist(.data)`
- `is_twomode(.data)`
- `is_weighted(.data)`
- `is_directed(.data)`
- `is_labelled(.data)`
- `is_signed(.data)`
- `is_complex(.data)`
is_multiplex(.data)
is_uniplex(.data)
is_longitudinal(.data)
is_dynamic(.data)
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.
TRUE if the condition is met, or FALSE otherwise.

Functions
• is_manynet(): Tests whether network is manynet-compatible
• is_graph(): Tests whether network contains graph-level information
• is_edgelist(): Tests whether data frame is an edgelist
• is_twomode(): Tests whether network is a two-mode network
• is_weighted(): Tests whether network is weighted
• is_directed(): Tests whether network is directed
• is_labelled(): Tests whether network includes names for the nodes
is

- is_signed(): Tests whether network is signed positive/negative
- is_complex(): Tests whether network contains any loops
- is_multiplex(): Tests whether network is multiplex, either from multiple rows with the same sender and receiver, or multiple columns to the edgelist.
- is_uniplex(): Tests whether network is simple (both uniplex and simplex)
- is_longitudinal(): Tests whether network is longitudinal, panel data
- is_dynamic(): Tests whether network is dynamic, time-stamped data
- is_connected(): Tests whether network is weakly connected if the network is undirected or strongly connected if directed. To test weak connection on a directed network, please see to_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 @importFrom igraph has_eulerian_path
- is_acyclic(): Tests whether network is a directed acyclic graph
- is_aperiodic(): Tests whether network is aperiodic

Source

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

See Also

Other mapping: attributes(), autographing, configuration_layouts, partition_layouts, properties

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_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))
is_longitudinal(create_tree(5, 3))
is_dynamic(create_tree(3))
is_connected(ison_southern_women)
is_perfect_matching(ison_southern_women)
is_eulerian(ison_brandes)
is_acyclic(ison_algebra)
is_aperiodic(ison_algebra)
**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**

```r
data(ison_adolescents)
```

**Format**

```r
#> # 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**


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

```r
#> A multiplex, directed network with 16 nodes and 144 arcs
#> A tibble: 144 x 5
#> from to friends social tasks
#> <int> <int> <dbl> <dbl> <dbl>
#> 1 1 5 0 1.2 0.3
#> 2 1 8 0 0.15 0
#> 3 1 9 0 2.85 0.3
#> 4 1 10 0 6.45 0.3
#> 5 1 10 0 6.45 0.3
#> 6 1 12 0 1.95 0.15
#> # i 138 more rows
```

Source

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

References

ison_friends

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

```r
#> # 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


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

```r
#> # A labelled, undirected network with 34 nodes and 78 ties
#> # A tibble: 34 x 2
#> name obc
#> <chr> <dbl>
#> 1 "Mr Hi" 1
#> 2 "" 1
#> 3 "" 1
#> 4 "" 1
#> 5 "" 1
#> 6 "" 1
#> # i 28 more rows
#> # A tibble: 78 x 2
#> from to
#> <int> <int>
#> 1 1 2
#> 2 1 3
#> 3 1 4
#> 4 1 5
#> 5 1 6
#> 6 1 7
#> # i 72 more rows
```

References

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 2
#> from to
#> <int> <int>
#> 1 1  2
#> 2 1  2
#> 3 1  3
#> 4 2  3
#> 5 2  4
#> 6 2  4
#> # i 1 more row

References


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
#> # 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
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, law school they attended, their age, and how many years they had been at the firm.
Usage
data(ison_lawfirm)

Format

```r
#> # A undirected network with 36 nodes and 115 ties
#> # A tibble: 36 x 8
#> Seniority Status Gender Office Years Age Practice School
#> <int> <chr> <chr> <chr> <int> <int> <chr> <chr>
#> 1 1 Partner Man Boston 31 64 Litigation Harvard/Yale
#> 2 2 Partner Man Boston 32 62 Corporate Harvard/Yale
#> 3 3 Partner Man Hartford 13 67 Litigation Harvard/Yale
#> 4 4 Partner Man Boston 31 59 Corporate Other
#> 5 5 Partner Man Hartford 31 59 Litigation Connecticut
#> 6 6 Partner Man Hartford 29 55 Litigation Harvard/Yale
#> i 30 more rows
#> # A tibble: 115 x 2
#> from to
#> <int> <int>
#> 1 1 17
#> 2 2 7
#> 3 2 16
#> 4 2 17
#> 5 2 22
#> 6 2 26
#> i 109 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.

References


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_marvel)

data(ison_marvel_teams)

data(ison_marvel_relationships)

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

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.
# Format

```r
#> # 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
```

```r
#> # 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 Power Origin
#> <chr> <chr> <int> <int> <int> <int> <int> <chr>       <chr>
#> 1 Abomination Male 427 0 0 1 1 Radiation
#> 2 Ant-Man Male 589 1 0 1 0 Human
#> 3 Apocalypse Male 1207 0 0 1 1 Mutant
#> 4 Beast Male 7609 1 0 1 0 Human
#> 5 Black Panther Male 2189 1 1 1 0 Human
#> 6 Black Widow 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 Yüksel:

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

Source

Umut Yüksel, 31 March 2017

---

ison_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

```r
data(ison_networkers)
```

Format

```r
# 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 Arabeie   Other        16
# 6 Carol Barner-Barry Other       6
```
ison_southern_women

```r
#> # 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
#> # i 434 more rows
```

Source

networkdata package

References


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

```r
#> # 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
```
# ison_starwars

```r
#> 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


## 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 (e.g., 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. Weighted ties represent the number of times characters speak within the same scene of the film.

## Usage

```r
data(ison_starwars)
```

## Format

```r
# 'Episode I'
# A labelled, weighted, undirected network with 38 nodes and 135 ties
# A tibble: 38 x 10
# 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 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>
# 5 TC-14  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 1 more variable: species <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 10
> 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 white, bl~ red 33 none Naboo
> 2 CAPTAIN 185 85 black dark brown NA male Naboo
> 3 EMPEROR 170 75 grey pale yellow 82 male Naboo
> 4 SENATOR NA NA NA <NA> <NA> <NA> <NA>
> 5 ORN FEATHERNA NA NA <NA> <NA> <NA> <NA>
> 6 MACE WINDU 188 84 none dark brown 72 male Haruun K-1
> # i 27 more rows
> # i 1 more variable: species <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 10
> 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 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>
> 5 GENER~ NA NA <NA> <NA> <NA> <NA>
> 6 MACE ~ 188 84 none dark brown 72 male Haruun K~
```r
# 6 EMPEROR 170 75 grey pale yellow 82 male Naboo
# i 18 more rows
# i 1 more variable: species <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 10
# 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 white, bl~ red 33 none Naboo
# 2 CHEWBACCA 228 112 brown unknown blue 200 male Kashyyyk
# 3 C-3PO 167 75 gold yellow 112 none Tatooine
# 4 LUKE 172 77 blond fair blue 19 male Tatooine
# 5 DARTH VADER 202 136 none white yellow 41.9 male Tatooine
# 6 CAMIE NA NA <NA> <NA> <NA> NA <NA> <NA>
# i 15 more rows
# i 1 more variable: species <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 10
# 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 white, bl~ red 33 none Naboo
# 2 CHEWBACCA 228 112 brown unknown blue 200 male Kashyyyk
# 3 C-3PO 167 75 gold yellow 112 none Tatooine
# 4 LUKE 172 77 blond fair blue 19 male Tatooine
# 5 HAN SOLO 180 80 brown fair brown 29 male Corellia
# 6 RIeka NA NA <NA> <NA> <NA> NA <NA> <NA>
```
```r
#> 6 LEIA 150 49 brown light brown 19 female Alderaan
#> # i 15 more rows
#> # i 1 more variable: species <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
#>
#> `$Episode VI`
#> # A labelled, weighted, undirected network with 20 nodes and 55 ties
#> # A tibble: 20 x 10
#> name height mass hair_color skin_color eye_color birth_year sex homeworld
#> <chr> <int> <dbl> <chr> <chr> <chr> <dbl> <chr> <chr>
#> 1 R2-D2 96 32 <NA> white, bl~ red 33 none Naboo
#> 2 CHEWB~ 228 112 brown unknown blue 200 male Kashyyhk
#> 3 JERJE~ NA NA <NA> <NA> <NA> NA <NA> <NA>
#> 4 DARTH~ 202 136 none white yellow 41.9 male Tatooine
#> 5 C-3PO 167 75 <NA> gold yellow 112 none Tatooine
#> 6 BIB F~ 180 NA none pale pink NA male Ryloth
#> # i 14 more rows
#> # i 1 more variable: species <chr>
#> # A tibble: 55 x 3
#> from to weight
#> <int> <int> <int>
#> 1 1 2 8
#> 2 1 5 14
#> 3 1 12 2
#> 4 1 7 2
#> 5 1 8 8
#> 6 1 10 9
#> # i 49 more rows
#>
#> `$Episode VII`
#> # A labelled, weighted, undirected network with 27 nodes and 92 ties
#> # A tibble: 27 x 10
#> 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 Kashyyhk
#> 4 BB-8 NA NA none none black NA none <NA>
#> 5 LOR S~ NA NA <NA> <NA> <NA> NA <NA> <NA>
```
Details

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

References


Description

These functions offer tools for imputing missing tie data. Currently two options are available: replacing the missing values with zeros, which are the modal value in sparse social networks, and replacing the missing values with the average non-missing value for that vector.

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.

Functions

- `na_to_zero()`: Impute missing tie data as zero, the modal value in sparse social networks.
- `na_to_mean()`: Impute missing tie data as the mean value in the network.

References


See Also

Other manipulations: `add`, `from`, `reformat`, `split()`, `tidy`, `transform()`

Examples

```r
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 `autopapr()` 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.
partition_layouts

Usage

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

layout_tbl_graph_multilevel(.data, level = NULL, 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

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 vector of partition memberships.

radius                 A vector of radii at which the concentric circles should be located. 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. 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.

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(), autographing, configuration_layouts, is(), properties

properties Describing network properties

Description

These functions extract certain attributes from given network data. They are also useful 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.

Functions

• network_nodes(): Returns the total number of nodes (of any mode) in a network.
• network_ties(): Returns the number of edges 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 edge attributes in a network
See Also

Other mapping: `attributes()`, `autographing`, `configuration_layouts`, `is()`, `partition_layouts`

Examples

```r
network_nodes(ison_southern_women)
network_ties(ison_southern_women)
network_dims(ison_southern_women)
network_dims(to_model(ison_southern_women))
network_node_attributes(ison_lotr)
network_tie_attributes(ison_algebra)
```

read  Make networks from/to external formats

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.

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.

Usage

```r
read_matrix(file = file.choose(), sv = c("comma", "semi-colon"), ...)
write_matrix(.data, filename, name, ...)
read_edgelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)
write_edgelist(.data, filename, name, ...)
read_nodelist(file = file.choose(), sv = c("comma", "semi-colon"), ...)
write_nodelist(.data, filename, name, ...)
read_pajek(file = file.choose(), ties = NULL, ...)
write_pajek(.data, filename, ...)
read_ucinet(file = file.choose())
write_ucinet(.data, filename, name)
read_dynetml(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.

**.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**
UCINET filename (without ## extension). By default the files will have the same name as the object and be saved to the working directory.

**name**
name of matrix to be known in UCINET. By default the name will be the same as the object.

**ties**
Where there are

Details

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

- UCINET data
- Pajek data

See also:

- networkdata
- GML datasets
- UCIrvine Network Data Repository
- KONECT project
- SNAP Stanford Large Network Dataset Collection

Please let us know if you identify any further repositories of social or political networks and we would be happy to add them here.

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

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.

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

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

Functions

- read_matrix(): Reading adjacency matrices from Excel/csv files
- write_matrix(): Writing matrix to csv files
- read_edgelist(): Reading edgelists from Excel/csv files
- write_edgelist(): Writing edgelists to csv files
- read_nodelist(): Reading nodelists from Excel/csv files
- write_nodelist(): Writing nodelists to csv files
- read_pajek(): Reading pajek (.net/.paj) files
- write_pajek(): Writing pajek .net files
- read_ucinet(): Reading UCINET files
- write_ucinet(): Writing UCINET files
- read_dynetml(): Reading DynetML files

Source

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

See Also

as
Other makes: as(), create, generate

Description

These functions offer tools for reformatting mgigraph-consistent objects (matrices, igraph, tidygraph, or network objects). Unlike the as_*() group of functions, these functions always return the same object type as they are given, only transforming these objects’ properties.

Not all functions have methods available for all object classes. Below are the currently implemented S3 methods:
to_uniplex(.data, edge)
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_onemode(.data)
to_multilevel(.data)
to_twomode(.data, mark)
Arguments

Arguments of a \{manynet\}-consistent class:

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

edge

Character string naming an edge 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.

mark

A logical vector marking two types or modes. By default "type".

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

- \texttt{to_uniplex()}: Returns an object that includes only a single type of tie.
- \texttt{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\}.
- \texttt{to_directed()}: Returns a directed object. Note that ties’ direction will be randomly assigned. To flip the direction, use \texttt{to_redirected()}. To match the direction, use \texttt{to_reciprocated()}. 
- \texttt{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.
- \texttt{to_reciprocated()}: Returns an object where all ties are reciprocated.
- \texttt{to_acyclic()}: Returns an object where all ties are acyclic.
- \texttt{to_unweighted()}: Returns an object that has all edge weights removed.
- \texttt{to_unsigned()}: Returns a network with either just the "positive" ties or just the "negative" ties.
- \texttt{to_unnamed()}: Returns an object with all vertex names removed.
- \texttt{to_named()}: Returns an object that has random vertex names added.
- \texttt{to_simplex()}: Returns an object that has all loops or self-ties removed.
- \texttt{to_onemode()}: Returns an object that has any type/mode attributes removed, but otherwise includes all the same nodes and ties. Note that this is not the same as \texttt{to_mode1()} or \texttt{to_mode2()}, which return only some of the nodes and new ties established by coincidence.
- \texttt{to_multilevel()}: Returns a network that is not divided into two mode types but embeds two or more modes into a multimodal network structure.
- \texttt{to_twomode()}: Returns a network that divides the nodes into two mode types.
See Also

Other manipulations: add, from, miss, split(), tidy, transform()

Examples

```r
as_tidygraph(create_filled(5)) %>%
  mutate_ties(type = sample(1:2, 10, replace = TRUE)) %>%
to_uniplex("type")
```

### Description

IHEID color scales

Centres color scales

SDGs color scales

### Usage

```r
scale_fill_iheid(palette = "IHEID", direction = 1, ...)
scale_colour_iheid(palette = "IHEID", direction = 1, ...)
scale_color_iheid(palette = "IHEID", direction = 1, ...)
scale_fill_centres(palette = "Centres", direction = 1, ...)
scale_colour_centres(palette = "Centres", direction = 1, ...)
scale_color_centres(palette = "Centres", direction = 1, ...)
scale_fill_sdgs(palette = "SDGs", direction = 1, ...)
scale_colour_sdgs(palette = "SDGs", direction = 1, ...)
scale_color_sdgs(palette = "SDGs", direction = 1, ...)
```

### Arguments

- **palette**: Name of the palette. Current choices are: IHEID, Centres, and SDGs.
- **direction**: Direction for using palette colors.
- **...**: Extra arguments passed to ggplot2::discrete_scale().
split

Tools for splitting networks, graphs, and matrices 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>data.frame</th>
<th>igraph</th>
<th>matrix</th>
<th>network</th>
<th>tbl_graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_components</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>to_egos</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>to_slices</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>to_subgraphs</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>to_waves</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Usage

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

to_subgraphs(.data, attribute)

to_components(.data)

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

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.
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.

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 manipulations: add, from, miss, reformat, tidy, transform()

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 %>%
mutable_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
to_waves(attribute = "wave")
ison_adolescents %>%
mutable_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
to_waves(attribute = "wave", panels = c(1995, 1996))
ison_adolescents %>%
mutable_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 = "sans")

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

Value
Themes the current ggraph to current IHEID guidelines.

Examples
# autographr(to_mentoring(ison_brandes)) +
# labs(title = "Who leads and who follows?") +
# theme_iheid()

 tidy  Tidy manipulations of node or tie data

Description
These functions allow users to add nodes, ties, or attributes to the nodes or ties of a network. The join_*, mutate_*, select_*, filter_*, rename_*(), and summarise_*() functions adapt the {dplyr}-type syntax to work with networks of any type.

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

join_ties(.data, object2, attr_name)
mutate_ties(.data, ...)
select_ties(.data, ...)
filter_ties(.data, ...)
rename_ties(.data, ...)
summarise_ties(.data, ...)
bind_node_attributes(.data, object2)

Arguments

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

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

object2 A second object to copy nodes or edges 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".
attr_name Name of the new attribute in the resulting object.
... Additional arguments.

Value

A tidygraph (tbl_graph) data object.

Functions

• join_nodes(): Copies node attributes from a given graph into specified graph
• join_ties(): Copies ties from another graph to specified graph and adds a tie attribute identifying the ties that were newly added
• mutate_ties(): Tidy way to add vector as tie attributes.
• select_ties(): Tidy way to select tie attributes.
• filter_ties(): Tidy way to filter ties based on a logical statement with relation to some tie attribute.
• rename_ties(): Tidy way to rename tie attributes.
• summarise_ties(): Tidy way to summarise tie attributes.
• bind_node_attributes(): Copying all nodal attributes from one network to another
transform

See Also

Other manipulations: add, from, miss, reformat, split(), transform()

Examples

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)
muse_ties(other, form = 1:6) %>% filter_ties(form < 4)

Description

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

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_anti | 1 | 1 | 0 | 1 | 1 |
| to_blocks | 1 | 1 | 0 | 1 | 1 |
| to_eulerian | 0 | 1 | 0 | 0 | 1 |
| to_giant | 1 | 1 | 0 | 1 | 1 |
| to_matching | 1 | 1 | 0 | 1 | 1 |
| to_mode1 | 1 | 1 | 0 | 1 | 1 |
| to_mode2 | 1 | 1 | 0 | 1 | 1 |
| to_no_isolates | 1 | 1 | 1 | 1 | 1 |
| to_subgraph | 1 | 1 | 0 | 1 | 1 |
| to_ties | 1 | 1 | 0 | 1 | 1 |

Usage

to_mode1(.data, similarity = c("count", "jaccard", "rand", "pearson", "yule"))
to_mode2(.data, similarity = c("count", "jaccard", "rand", "pearson", "yule"))
to_giant(.data)
to_subgraph(.data, ...)
to_ties(.data)
to_blocks(.data, membership, FUN = mean)

to_matching(.data, mark = "type")

to_mentoring(.data, elites = 0.1)

to_eulerian(.data)

to_anti(.data)

to_no_isolates(.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.

... 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.

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).
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_mode1()**: Results in a weighted one-mode object that retains the row nodes from a two-mode object, and weights the ties between them on the basis of their joint ties to nodes in the second mode (columns).
- **to_mode2()**: Results in a weighted one-mode object that retains the column nodes from a two-mode object, and weights the ties between them on the basis of their joint ties to nodes in the first mode (rows).
- **to_giant()**: Returns an object that includes only the main component without any smaller components or isolates.
- **to_subgraph()**: Returns a network subgraph filtered on the basis of some node-related logical statement.
- **to_ties()**: Returns a matrix (named if possible) where the edges are the nodes.
- **to_blocks()**: Returns a reduced graph from a given partition membership vector. 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.
- **to_matching()**: Returns a network with only matching ties.
- **to_mentoring()**: Returns a network where each node is connected only to their closest mentor.
- **to_eulerian()**: Returns a network with only the Eulerian path.
- **to_anti()**: Returns the complement of a network where only ties not present in the original network are included in the new network.
- **to_no_isolates()**: Removes all nodes without ties.
- **to_galois()**: Galois derivations.

**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


See Also

Other manipulations: add, from, miss, reformat, split(), tidy

Examples

```r
to_model(ison_southern_women)
to_mode2(ison_southern_women)
#autographr(to_mode2(ison_southern_women))
to_ties(ison_adolescents)
#autographr(to_ties(ison_adolescents))
to_matching(ison_southern_women)
#autographr(to_matching(ison_southern_women))
autographr(to_mentoring(ison_adolescents))
  to_eulerian(delete_nodes(ison_konigsberg, "Lomse"))
  #autographr(to_eulerian(delete_nodes(ison_konigsberg, "Lomse")))
to_anti(ison_southern_women)
#autographr(to_anti(ison_southern_women))
ison_adolescents %>%
  activate(edges) %>%
      mutate(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.

Usage

```r
run_tute(tute)
```

```r
extract_tute(tute)
```

Arguments

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

Functions

- `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
Examples

#run_tute("tutorial2")
#extract_tute("tutorial2")
Index

* colors
  iheid_palette, 16
* datasets
  ison_adolescents, 20
  ison_algebra, 21
  ison_brandes, 22
  ison_friends, 22
  ison_karateka, 23
  ison_konigsberg, 24
  ison_laterals, 25
  ison_lawfirm, 27
  ison_lotr, 28
  ison_marvel, 29
  ison_networkers, 31
  ison_southern_women, 32
  ison_starwars, 33
* makes
  as, 4
  create, 11
  generate, 14
  read, 41
* manipulations
  add, 2
  from, 13
  miss, 37
  reformat, 43
  split, 47
  tidy, 49
  transform, 51
* mapping
  attributes, 6
  autographing, 7
  configuration_layouts, 10
  is, 17
  partition_layouts, 38
  properties, 40
* marks
  is, 17
  add, 2, 14, 38, 46, 48, 51, 54
  add_node_attribute (add), 2
  add_nodes (add), 2
  add_tie_attribute (add), 2
  add_ties (add), 2
  as, 4, 13, 16, 43
  as_edgelist (as), 4
  as_graphAM (as), 4
  as_igraph (as), 4
  as_matrix (as), 4
  as_network (as), 4
  as_siena (as), 4
  as_tidygraph (as), 4
  attributes, 6, 9, 10, 19, 40, 41
  autographd (autographing), 7
  autographing, 7, 7, 10, 19, 40, 41
  autographr (autographing), 7
  autographs (autographing), 7
  bind_node_attributes (tidy), 49
  configuration_layouts, 7, 9, 10, 19, 40, 41
  create, 5, 11, 16, 43
  create_components (create), 11
  create_core (create), 11
  create_empty (create), 11
  create_explicit (create), 11
  create_filled (create), 11
  create_lattice (create), 11
  create_ring (create), 11
  create_star (create), 11
  create_tree (create), 11
  delete_nodes (add), 2
  extract_tute (tutorials), 54
  filter_ties (tidy), 49
  from, 4, 13, 38, 46, 48, 51, 54
  from_egos (from), 13
  from_slices (from), 13
  from_subgraphs (from), 13

56
INDEX

from_waves (from), 13

generate, 5, 13, 14, 43
generate_permutation (generate), 14
generate_random (generate), 14
generate_scalefree (generate), 14
generate_smallworld (generate), 14
generate_utilities (generate), 14

igraph::graph_from_literal(), 13
iheid_palette, 16
is, 7, 9, 10, 17, 40, 41
is_acyclic (is), 17
is_aperiodic (is), 17
is_complex (is), 17
is_connected (is), 17
is_directed (is), 17
is_dynamic (is), 17
is_edgelist (is), 17
is_eulerian (is), 17
is_graph (is), 17
is_labelled (is), 17
is_longitudinal (is), 17
is_manynet (is), 17
is_multiplex (is), 17
is_perfect_matching (is), 17
is_signed (is), 17
is_twomode (is), 17
is_uniplex (is), 17
is_weighted (is), 17
ison_adolescents, 20
ison_algebra, 21
ison_brandes, 22
ison_friends, 22
ison_karateka, 23
ison_konigsberg, 24
ison_laterals, 25
ison_lawfirm, 27
ison_lotr, 28
ison_marvel, 29
ison_marvel_relationships
  (ison_marvel), 29
ison_marvel_teams (ison_marvel), 29
ison_networkers, 31
ison_southern_women, 32
ison_starwars, 33

join_nodes (tidy), 49
join_ties (tidy), 49

layout_tbl_graph_alluvial
  (partition_layouts), 38
layout_tbl_graph_concentric
  (partition_layouts), 38
layout_tbl_graph_hierarchy
  (partition_layouts), 38
layout_tbl_graph_ladder
  (partition_layouts), 38
layout_tbl_graph_multilevel
  (partition_layouts), 38
layout_tbl_graph_quad
  (configuration_layouts), 10
layout_tbl_graph_railway
  (partition_layouts), 38
layout_tbl_graph_triad
  (configuration_layouts), 10

miss, 4, 14, 37, 46, 48, 51, 54
mutate_ties (tidy), 49

na_to_mean (miss), 37
na_to_zero (miss), 37
network_dims (properties), 40
network_node_attributes (properties), 40
network_nodes (properties), 40
network_tie_attributes (properties), 40
network_ties (properties), 40
node_attribute (attributes), 6
node_mode (attributes), 6
node_names (attributes), 6

partition_layouts, 7, 9, 10, 19, 38, 41
properties, 7, 9, 10, 19, 40, 40

read, 5, 13, 16, 41
read_dynetml (read), 41
read_edgelist (read), 41
read_matrix (read), 41
read_modelist (read), 41
read_pajek (read), 41
read_ucinet (read), 41
reformat, 4, 14, 38, 43, 48, 51, 54
rename_ties (tidy), 49
run_tute (tutorials), 54

scale_color_centres (scales), 46
scale_color_iheid (scales), 46
scale_color_sdgs (scales), 46
scale_colour_centres (scales), 46
scale_colour_iheid (scales), 46
scale_colour_sdgs (scales), 46
scale_fill_centres (scales), 46
scale_fill_iheid (scales), 46
scale_fill_sdgs (scales), 46
select_ties (tidy), 49
split, 4, 14, 38, 46, 47, 51, 54
summarise_ties (tidy), 49

theme_iheid (themes), 49
themes, 49
tidy, 4, 14, 38, 46, 48, 49, 54
tie_attribute (attributes), 6
tie_signs (attributes), 6
tie_weights (attributes), 6
to_acyclic (reformat), 43
to_anti (transform), 51
to_blocks (transform), 51
to_components (split), 47
to_directed (reformat), 43
to_egos (split), 47
to_eulerian (transform), 51
to_galois (transform), 51
to_giant (transform), 51
to_matching (transform), 51
to_mentoring (transform), 51
to_mode1 (transform), 51
to_mode2 (transform), 51
to_multilevel (reformat), 43
to_named (reformat), 43
to_no_isolates (transform), 51
to_onemode (reformat), 43
to_reciprocated (reformat), 43
to_redirected (reformat), 43
to_simplex (reformat), 43
to_slices (split), 47
to_subgraph (transform), 51
to_subgraphs (split), 47
to_ties (transform), 51
to_twomode (reformat), 43
to_undirected (reformat), 43
to_uniplex (reformat), 43
to_unnamed (reformat), 43
to_unsigned (reformat), 43
to_unweighted (reformat), 43
to_waves (split), 47
transform, 4, 14, 38, 46, 48, 51, 54
tutorials, 54

write_edgelist (read), 41
write_matrix (read), 41
write_nodelist (read), 41
write_pajek (read), 41
write_ucinet (read), 41