Package ‘manynet’

June 10, 2023

Title Many Ways to Make, Manipulate, and Map Myriad Networks

Version 0.1.1

Date 2023-06-09

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://github.com/snlab-ch/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, network, tidygraph

Suggests gganimate, methods, readxl, roxygen2, RSiena, testthat, xml2, patchwork

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 2023-06-10 14:10:02 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>2</td>
</tr>
<tr>
<td>as</td>
<td>3</td>
</tr>
<tr>
<td>attributes</td>
<td>5</td>
</tr>
<tr>
<td>auto_graph</td>
<td>7</td>
</tr>
<tr>
<td>create</td>
<td>9</td>
</tr>
<tr>
<td>from</td>
<td>11</td>
</tr>
<tr>
<td>generate</td>
<td>12</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
</tr>
<tr>
<td>ison_adolescents</td>
<td>16</td>
</tr>
<tr>
<td>ison_algebra</td>
<td>17</td>
</tr>
<tr>
<td>ison_brandes</td>
<td>18</td>
</tr>
<tr>
<td>ison_karateka</td>
<td>19</td>
</tr>
<tr>
<td>ison_lotr</td>
<td>20</td>
</tr>
<tr>
<td>ison_marvel</td>
<td>21</td>
</tr>
<tr>
<td>ison_networkers</td>
<td>23</td>
</tr>
<tr>
<td>ison_projection</td>
<td>24</td>
</tr>
<tr>
<td>ison_southern_women</td>
<td>26</td>
</tr>
<tr>
<td>miss</td>
<td>27</td>
</tr>
<tr>
<td>properties</td>
<td>28</td>
</tr>
<tr>
<td>read</td>
<td>30</td>
</tr>
<tr>
<td>reformat</td>
<td>32</td>
</tr>
<tr>
<td>split</td>
<td>35</td>
</tr>
<tr>
<td>tidy</td>
<td>36</td>
</tr>
<tr>
<td>transform</td>
<td>38</td>
</tr>
</tbody>
</table>

**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}`.

**Usage**

```r
add_nodes(.data, nodes, attribute = NULL)
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 ties to 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))

as  Making networks into other classes
**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**

```r
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:

to/from edgelists matrices igraph tidygraph network siena goldfish
edgelists (data frames) X X X X X X X
matrices X X X X X X X
igraph X X X X X X X
tidygraph X X X X X X X
network X X X X X X X
graphAM X X X X X X X

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)
```

Description

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

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: auto_graph, is(), properties

Examples

  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)

---

auto_graph

Quickly graph networks with sensible defaults

Description

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

autographr(
  .data,
  layout = "stress",
  labels = TRUE,
  node_color = NULL,
  node_shape = NULL,
  node_size = NULL,
  edge_color = NULL,
  ...
)

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, currently defaults to 'stress'.
labels Logical, whether to print node names as labels if present.
node_color Node variable in quotation marks to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting.
Character string in quotation marks referring to the name of a node attribute already present in the graph to be used for the shapes of the nodes. Shapes follow the ordering "circle", "square", "triangle", so this aesthetic should be used for a variable with only a few categories.

Node variable in quotation marks to be used for the size of the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all node-related statistics prior to using this function.

Tie variable in quotation marks 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.

Extra arguments to pass on to autographr() / ggraph() / ggplot().

A list of migraph-compatible networks.

The same migraph-compatible network listed according to a time attribute, waves, or slices.

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.

A ggplot2::ggplot() object.

• 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

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

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

ison_adolescents %>%
mutate(shape = rep(c("circle", "square"), times = 4),
       color = rep(c("blue", "red"), times = 4)) %>%
autographr(node_shape = "shape", node_color = "color")
autographr(ison_karateka, node_size = 8)
autographs(to_egos(ison_adolescents))
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 \( \text{directed} = \text{TRUE} \). This will return a directed network in which the arcs are out-facing or equivalent. This direction can be swapped using \text{to_redirected()}\). In two-mode networks, the directed argument is ignored.

**Usage**

- \text{create_empty}(n, \text{directed} = \text{FALSE})
- \text{create_filled}(n, \text{directed} = \text{FALSE})
- \text{create_ring}(n, \text{directed} = \text{FALSE}, \text{width} = 1, \ldots)
- \text{create_star}(n, \text{directed} = \text{FALSE})
- \text{create_tree}(n, \text{directed} = \text{FALSE}, \text{width} = 2)
- \text{create_lattice}(n, \text{directed} = \text{FALSE}, \text{width} = 8)
- \text{create_components}(n, \text{directed} = \text{FALSE}, \text{membership} = \text{NULL})
- \text{create_core}(n, \text{directed} = \text{FALSE}, \text{membership} = \text{NULL})

**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.

- \( \text{directed} \)  
  Logical whether the graph should be directed. By default \( \text{directed} = \text{FALSE} \). If the opposite direction is desired, use \text{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 neighbourhood.

... Additional arguments passed on to \{igraph\}.

membership

A vector of partition membership as integers. If left as \texttt{NULL} (the default), nodes in each mode will be assigned to two, equally sized partitions.

Value

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

Functions

- \texttt{create_empty}(): Creates an empty graph of the given dimensions.
- \texttt{create_filled}(): Creates a filled graph of the given dimensions, with every possible tie realised.
- \texttt{create_ring}(): Creates a ring or chord graph of the given dimensions that loops around is of a certain width or thickness.
- \texttt{create_star}(): Creates a graph of the given dimensions that has a maximally central node.
- \texttt{create_tree}(): Creates a graph of the given dimensions with successive branches.
- \texttt{create_lattice}(): Creates a lattice graph of the given dimensions with ties to all neighbouring nodes.
- \texttt{create_components}(): Creates a graph in which the nodes are clustered into separate components.
- \texttt{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.

Lattice graphs

\texttt{create_lattice}() creates both two-dimensional grid and triangular lattices with as even dimensions as possible. When the \texttt{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 \texttt{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 \texttt{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

\texttt{as}

Other makes: \texttt{as()}, \texttt{generate}, \texttt{read}
**Examples**

```r
create_empty(10)
create_filled(10)
create_ring(8, width = 2)
create_star(12)
create_tree(c(7,8))
create_lattice(c(7,8))
create_components(10, membership = c(1,1,1,2,2,2,3,3,3,3))
create_core(6)
```

---

**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**

```r
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()
```

**generate**

*Make 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.

Usage

```r
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
Proportion of possible ties in the network that are realised or, if integer greater than 1, the number of ties in the network.

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

as

Other makes: as(), create, read

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_smallworld(c(6,6), 0.025))
autographr(generate_smallworld(c(12,6), 0.25))
autographr(generate_scalefree(12, 0.25))
autographr(generate_scalefree(12, 1.25))
autographr(generate_scalefree(c(12,6), 0.25))
autographr(generate_scalefree(c(12,6), 1.25))
autographr(ison_adolescents)
autographr(generate_permutation(ison_adolescents))

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)

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.

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_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

See Also
Other mapping: `attributes()`, `auto_graph`, `properties`

Examples

```r
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))
```

---

### 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

```r
#> # A labelled, undirected tbl_graph 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

**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

```r
data(ison_algebra)
```
Format

```r
#> # A labelled, multiplex, directed tbl_graph with 16 nodes and 144 arcs
#> # A tibble: 16 x 1
#> name
#> <chr>
#> 1 Melinda
#> 2 Abby
#> 3 Darryl
#> 4 Veronica
#> 5 Rylan
#> 6 Lindsey
#> # i 10 more rows
#> # 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 11 0 0.3 0
#> 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_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.

Usage

```r
data(ison_brandes)
data(ison_brandes2)
```
Format

```r
#> # A undirected tbl_graph with 11 nodes and 12 ties
#> # 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
```

```r
#> # A two-mode tbl_graph with 11 nodes and 12 ties
#> # A tibble: 11 x 1
#> type
#> <lgl>
#> 1 FALSE
#> 2 FALSE
#> 3 TRUE
#> 4 FALSE
#> 5 TRUE
#> 6 TRUE
#> # i 5 more rows
```

```r
#> # A undirected tbl_graph with 11 nodes and 12 ties
#> # 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_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 tbl_graph 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


---

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

```r
#> # A labelled, complex, undirected tbl_graph 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 (Yüksel 2017)*

Description

This package includes two datasets related to the Marvel *comic book* universe. The first, `izon_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, `izon_marvel_relations`, 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

```r
data(izon_marvel_teams)
data(izon_marvel_relations)
```
Format

```r
#> # A labelled, two-mode tbl_graph 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 tbl_graph 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 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 Mutant
#> 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

data(ison_networkers)

Format

```r
#> # A labelled, weighted, directed tbl_graph 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
```
ison_projection

#> # i 26 more rows
#> # A tibble: 440 x 3
#> # i 434 more rows
#> 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

Source

networkdata package

References


ison_projection Two-mode projection examples (Hollway 2021)

Description

These datasets are for demonstration purposes and do not describe any real world network. All examples contain named nodes.

Usage

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

```r
#> # A labelled, two-mode tbl_graph 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
#> # A labelled, two-mode tbl_graph 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
```

```r
#> # A labelled, two-mode tbl_graph with 8 nodes and 9 ties
#> # A tibble: 8 x 2
#> name type
#> <chr> <lgl>
#> 1 A FALSE
```
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**

```r
data(ison_southern_women)
```

**Format**

```r
#> # A labelled, two-mode tbl_graph with 32 nodes and 93 ties
#> # A tibble: 32 x 2
#> type name
#> <lgl> <chr>
#> 1 FALSE EVELYN
#> 2 FALSE LAURA
#> 3 FALSE THERESA
#> # i 26 more rows
#> # A tibble: 93 x 2
#> from to
#> <int> <int>
#> 1 1 19
#> 2 1 20
#> 3 1 21
#> # i 87 more rows
```

**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**

```r
na_to_zero(.data)
```

```r
na_to_mean(.data)
```
properties

Describing network properties

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

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)
### properties

**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()`, `auto_graph.is()`

**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)`
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 `{mynet}` 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.
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()`, `create`, `generate`

Description

These functions offer tools for reformatting migraph-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.

Usage

```r
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

.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

to_acyclic matrices igraph tidygraph network
unweighted X X X X X
undirected X X X X X
redirected X X X X
unsigned X X X X
uniplex X X X X
unnamed X X X X
named X X X X

eedge 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".

Details

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

to_acyclic matrices igraph tidygraph network
unweighted X X X X X
undirected X X X X X
redirected X X X X
unsigned X X X X
uniplex X X X X
unnamed X X X X X
named X X X X X
### Functions

- `to_uniplex()`: Returns an object that includes only a single type of tie
- `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_directed()`: Returns a directed object. Note that ties’ direction will be randomly assigned. To flip the direction, use `to_redirected()`. To match the direction, use `to_reciprocated()`.
- `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_acyclic()`: Returns an object where all ties are acyclic.
- `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
- `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 `to_mode1()` or `to_mode2()`, which return only some of the nodes and new ties established by coincidence.
- `to_multilevel()`: Returns a network that is not divided into two mode types but embeds two or more modes into a multimodal network structure.
- `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")
```
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).

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.

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.

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

```r
# Example 1
ison_adolescents <- to_egos(ison_adolescents)
autographs(to_egos(ison_adolescents, 2))
# Example 2
ison_adolescents %>%
  mutate(unicorn = sample(c("yes", "no"), 8, replace = TRUE)) %>%
  to_subgraphs(attribute = "unicorn")
# Example 3
ison_marvel_relationships <- to_components(ison_marvel_relationships)
# Example 4
ison_adolescents %>%
  mutate_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
  to_waves(attribute = "wave")
# Example 5
ison_adolescents %>%
  mutate_ties(wave = sample(1995:1998, 10, replace = TRUE)) %>%
  to_waves(attribute = "wave", panels = c(1995, 1996))
# Example 6
ison_adolescents %>%
  mutate_ties(time = 1:10, increment = 1) %>%
  add_ties(c(1, 2), list(time = 3, increment = -1)) %>%
  to_slices(slice = 7)
```
Usage

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

```r
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`
  - edgelist, a data frame from `base`
  - 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

See Also

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

Examples

```r
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)
mutate_ties(other, form = 1:6) %>% filter_ties(form < 4)
```

---

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_anti(.data)


to_no_isolates(.data)

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.

Usage

```r
to_model(.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_anti(.data)

to_no_isolates(.data)
```
transform

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
count 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 recom-
  mended options include median, sum, min or max.

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

Details

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

<table>
<thead>
<tr>
<th>to_</th>
<th>edgelists</th>
<th>matrices</th>
<th>igraph</th>
<th>tidygraph</th>
<th>network</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>mode2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>giant</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>subgraph</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ties</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>blocks</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>matching</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Value

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

autographr(to_mode1(ison_southern_women))
autographr(to_mode2(ison_southern_women))
autographr(to_ties(ison_adolescents))
autographr(to_matching(ison_southern_women))
autographr(to_anti(ison_southern_women))
ison_adolescents %>%
transform

activate(edges) %>%
mutate(wave = sample(1995:1998, 10, replace = TRUE)) %>%
to_waves(attribute = "wave") %>%
to_no_isolates()
Index

* datasets
  ison_adolescents, 16
  ison_algebra, 17
  ison_brandes, 18
  ison_karateka, 19
  ison_lotr, 20
  ison_marvel, 21
  ison_networkers, 23
  ison_projection, 24
  ison_southern_women, 26

* makes
  add, 2
  from, 11
  miss, 27
  reformat, 32
  split, 35
  tidy, 36
  transform, 38

* manipulations
  add, 2
  from, 11
  miss, 27
  reformat, 32
  split, 35
  tidy, 36
  transform, 38

* mapping
  attributes, 5
  auto_graph, 7
  is, 14
  properties, 28

  add, 2, 12, 28, 34, 36, 38, 40
  add_node_attribute (add), 2
  add_nodes (add), 2
  add_tie_attribute (add), 2
  add_ties (add), 2
  as, 3, 10, 14, 32
  as_edgelist (as), 3
  as_graphAM (as), 3
  as_igraph (as), 3
  as_matrix (as), 3
  as_network (as), 3

  as_siena (as), 3
  as_tidygraph (as), 3
  attributes, 5, 8, 16, 29
  auto_graph, 6, 7, 16, 29
  autographd (auto_graph), 7
  autographr (auto_graph), 7
  autographs (auto_graph), 7

  bind_node_attributes (tidy), 36
  create, 5, 9, 14, 32
  create_components (create), 9
  create_core (create), 9
  create_empty (create), 9
  create_filled (create), 9
  create_lattice (create), 9
  create_ring (create), 9
  create_star (create), 9
  create_tree (create), 9

  filter_ties (tidy), 36
  from, 3, 11, 28, 34, 36, 38, 40
  from_egos (from), 11
  from_slices (from), 11
  from_subgraphs (from), 11
  from_waves (from), 11

  generate, 5, 10, 12, 32
  generate_permutation (generate), 12
  generate_random (generate), 12
  generate_scalefree (generate), 12
  generate_smallworld (generate), 12
  generate_utilities (generate), 12

  is, 6, 8, 14, 29
  is_complex (is), 14
  is_directed (is), 14
  is_dynamic (is), 14
  is_edgelist (is), 14
  is_graph (is), 14
  is_labelled (is), 14
is_longitudinal (is), 14
is_manynet (is), 14
is_multiplex (is), 14
is_signed (is), 14
is_twomode (is), 14
is_uniplex (is), 14
is_weighted (is), 14
ison_adults (ison), 16
ison_algebra (ison), 17
ison_bb (ison), 24
ison_bm (ison), 24
ison_brandes (ison), 18
ison_brandes2 (ison), 18
ison_karateka (ison), 19
ison_lo (ison), 20
ison_marvel (ison), 21
ison_marvel_relationships (ison), 21
ison_marvel_teams (ison), 21
ison_mb (ison), 24
ison_mm (ison), 24
ison_networkers (ison), 23
ison_projection (ison), 24
ison_southern_women (ison), 26
join_nodes (ison), 36
join_ties (ison), 36
miss (ison), 3, 12, 27, 34, 36, 38, 40
mutate_ties (ison), 36
na_to_mean (ison), 27
na_to_zero (ison), 27
network_dims (ison), 28
network_node_attributes (ison), 28
network_nodes (ison), 28
network_tie_attributes (ison), 28
network_ties (ison), 28
node_attribute (ison), 5
node_mode (ison), 5
node_names (ison), 5
properties (ison), 6, 8, 16, 28
read (ison), 5, 10, 14, 30
read_dynom (ison), 30
read_edgelist (ison), 30
read_matrix (ison), 30
read_nodelist (ison), 30
read_pajek (ison), 30
read_ucinet (ison), 30
reformat (ison), 12, 28, 32, 36, 38, 40
rename_ties (ison), 36
select_ties (ison), 36
split (ison), 3, 12, 28, 34, 35, 38, 40
summarise_ties (ison), 36
tidy (ison), 3, 12, 28, 36, 38, 40
tie_attribute (ison), 5
tie_signs (ison), 5
tie_weights (ison), 5
to_acyclic (ison), 32
to_anti (ison), 38
to_blocks (ison), 38
to_components (ison), 35
to_directed (ison), 32
to_ego (ison), 35
to_giant (ison), 38
to_matching (ison), 38
to_model1 (ison), 38
to_model2 (ison), 38
to_multilevel (ison), 32
to_named (ison), 32
to_no_isolates (ison), 38
to_onepoint (ison), 32
to_reciprocated (ison), 32
to_redirected (ison), 32
to_simplex (ison), 32
to_slices (ison), 35
to_subgraph (ison), 38
to_subgraphs (ison), 35
to_ties (ison), 38
to_twomode (ison), 32
to_undirected (ison), 32
to_uniplex (ison), 32
to_unnamed (ison), 32
to_unsigned (ison), 32
to_unweighted (ison), 32
to_weighted (ison), 32
to_waves (ison), 35
transform (ison), 3, 12, 28, 34, 36, 38, 38
write_edgelist (ison), 30
write_matrix (ison), 30
write_nodelist (ison), 30
write_pajek (ison), 30
write_ucinet (ison), 30