Package 'manynet'

April 4, 2026

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

Version 2.0.1

Date 2026-04-04

Description Many tools for making, modifying, marking, measuring,
and motifs and memberships of many different types of networks.
All functions operate with matrices, edge lists, and 'igraph', 'network', and 'tidygraph' objects,
on directed, multiplex, multimodal, signed, and other networks.
The package includes functions for importing and exporting, creating and generating networks,
modifying networks and node and tie attributes,
and describing networks with sensible defaults.

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

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

License MIT + file LICENSE

Language en-GB

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports cli, dplyr (>= 1.1.0), igraph (>= 2.1.0), network, pillar,
tidygraph

Suggests knitr, methods, readxl, rmarkdown, RSiena, sna, testthat (>=
3.0.0), tibble, xml2

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown

Config/testthat/parallel true

Config/testthat/edition 3

Config/testthat/start-first mark_is

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID: <https://orcid.org/0000-0002-8361-9647>), Henrique Sposito [ctb] (ORCID: <https://orcid.org/0000-0003-3420-6085>), Christian Steglich [ctb]

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

Repository CRAN

Date/Publication 2026-04-04 08:30:02 UTC

Contents

`class_describe` *Describe a network*

Description

These functions are used to describe components of a given network in terms of a particular phrase.

- `describe_network()` describes the features or properties of a network, such as whether it is two-mode, directed, or complex.
- `describe_nodes()` describes how many of each type of nodes there are and, if available, names the different nodesets or modes.
- `describe_ties()` describes how many of each type of ties there are and, if available, names the different types of ties.
- `describe_changes()` describes the changing features of a network, if any, such as how many waves there are.

These descriptions are constructed to be GRAND-consistent.

Usage

```
describe_network(.data)

describe_nodes(.data)

describe_ties(.data)

describe_changes(.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

coerce_graph *Coercing into graph objects*

Description

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

- as_igraph() coerces the object into an {igraph} graph object.
- as_tidygraph() coerces the object into a {tidygraph} tbl_graph object.
- as_network() coerces the object into a {network} network object.
- as_siena() coerces the (igraph/tidygraph) object into a SIENA dependent variable.
- as_graphAM() coerces the object into a graph adjacency matrix.
- as_diffusion() coerces a table of diffusion events into a diff_model object similar to the output of play_diffusion().
- as_diffnet() coerces a diff_model object into a {netdiffuseR} diffnet object.

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_igraph(.data, twomode = FALSE)

as_tidygraph(.data, twomode = FALSE)

as_network(.data, twomode = FALSE)

as_siena(.data, twomode = FALSE)

as_graphAM(.data, twomode = NULL)

as_diffusion(.data, twomode = FALSE, events)

as_diffnet(.data, twomode = FALSE)

as_stocnet(.data, twomode = 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

twomode — Logical option used to override heuristics for distinguishing incidence (two-mode/bipartite) from adjacency (one-mode/unipartite) networks. By default FALSE.

events — A table (data frame or tibble) of diffusion events with columns t indicating the time (typically an integer) of the event, nodes indicating the number or name of the node involved in the event, and event, which can take on the values "I" for an infection event, "E" for an exposure event, or "R" for a recovery event.

Value

The currently implemented coercions or translations are:

	data.frame	diff_model	diffnet	igraph	list	matrix	mnet	network
as_diffnet		*						
as_diffusion		*	*	*			*	
as_graphAM	*			*		*		*
as_igraph	*	*	*	*		*		*
as_network	*		*	*		*		*
as_siena				*				
as_stocnet				*		*		*
as_tidygraph	*	*	*	*	*	*		*

	network.goldfish	networkDynamic	siena	stocnet	tbl_graph
as_diffnet					
as_diffusion					
as_graphAM	*		*	*	*
as_igraph	*	*	*	*	*
as_network	*	*	*	*	*
as_siena					*
as_stocnet				*	*
as_tidygraph	*	*	*	*	*

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

See Also

Other coercions: coerce_list

Examples

```
test <- data.frame(from = c("A","B","B","C","C"), to = c("I","G","I","G","H"))
as_edgelist(test)
as_matrix(test)
```

```
as_igraph(test)
as_tidygraph(test)
as_network(test)
  # How to create a diff_model object from (basic) observed data
  events <- data.frame(time = c(0,1,1,2,3),
                       node = c(1,2,3,2,4),
                       var = "diffusion",
                       value = c("I","I","I","R","I"))
  add_changes(create_filled(4), events)
```

`coerce_list` *Coercing into lists or matrices*

Description

These functions coerce objects into different objects by extracting and translating the information contained in the original object:

- `as_edgelist()` coerces the object into an edgelist, as data frames or tibbles.
- `as_nodelist()` coerces the object into a nodelist, as a data frame or tibble.
- `as_changelist()` coerces the object into a changelist, as a data frame or tibble.
- `as_infolist()` coerces the object into a list of network-level information, such as the names of the nodes and ties, if not given in the nodelist or edgelist.
- `as_matrix()` coerces the object into an adjacency (one-mode/unipartite) or incidence (two-mode/bipartite) matrix.

These coercions are extractive in the sense that they will lose any information that cannot be contained in the target format. for example, `as_matrix()` will lose any information about edge attributes, such as edge types or weights.

Usage

```
as_nodelist(.data)

as_changelist(.data)

as_edgelist(.data, twomode = FALSE)

as_infolist(.data)

as_matrix(.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.

Value

The currently implemented coercions or translations are:

	data.frame	diff_model	igraph	matrix	network	network.goldfish
as_changelist			*		*	
as_edgelist	*		*	*	*	*
as_infolist			*		*	
as_matrix	*	*	*	*	*	*
as_nodelist			*		*	

	siena	stocnet	tbl_graph
as_changelist		*	*
as_edgelist	*	*	*
as_infolist		*	*
as_matrix	*	*	*
as_nodelist		*	*

See Also

Other coercions: coerce_graph

Examples

```
test <- data.frame(from = c("A","B","B","C","C"), to = c("I","G","I","G","H"))
as_edgelist(test)
as_matrix(test)
as_igraph(test)
as_tidygraph(test)
as_network(test)
```

data_overview *Obtain overview of available network data*

Description

This function makes it easy to get an overview of available data:

- `table_data()` returns a tibble with details of the network datasets included in the packages.

Usage

```
table_data(..., pkg = c("manynet", "migraph"))
```

Arguments

...	Network marks, e.g. directed, twomode, or signed, that are used to filter the results.
pkg	String, name of the package.

Examples

```
table_data()
# to obtain list of all e.g. directed networks:
table_data(pkg = "manynet", directed)
# to obtain overview of unique datasets:
table_data() |>
  dplyr::distinct(directed, weighted, twomode, signed,
                  .keep_all = TRUE)
```

fict_actually *Two-mode network of Love Actually characters and their scene appearances (Robinson 2015)*

Description

Love Actually is a 2003 British romantic comedy film. David Robinson's data is a two-mode network of 20 characters and their appearances across 76 scenes, as parsed from the script.

Added to this are multiplex, one-mode networks of character relationships, including "romantic", "family", "friendship", and "professional" ties. These were added by Korakot Janteerasakul from the following source: https://en.wikipedia.org/wiki/Love_Actually#/media/File:Love_Actually_(2003)_Interconnections.svg.

Usage

```
data(fict_actually)
```

Format

```
#> -- # Love actually interactions ---------------------------------------------
#> # A labelled, multiplex, two-mode network of 20 characters and 76 scenes and
#> 162 appearance, 3 family, 7 friendship, 10 professional, and 7 romance ties
#>
#> -- Nodes
#> # A tibble: 96 x 6
#>   type  name    Actor          Gender Nationality Occupation
#>   <lgl> <chr>   <chr>          <chr>  <chr>       <chr>
#> 1 FALSE Aurelia Lúcia Moniz    Female Portuguese  Assistant
#> 2 FALSE Billy   Bill Nighy     Male   British     Singer
#> 3 FALSE Colin   Kris Marshall  Male   British     Travels to America
#> 4 FALSE Daniel  Liam Neeson    Male   British     Widower
#> 5 FALSE Harry   Alan Rickman   Male   British     Agency Director
#> 6 FALSE John    Martin Freeman Male   British     Nude stand-in
#> # i 90 more rows
#>
#> -- Ties
#> # A tibble: 189 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     7 romance
#> 2     1    51 appearance
#> 3     1    57 appearance
#> 4     1    58 appearance
#> 5     1    63 appearance
#> 6     1    95 appearance
#> # i 183 more rows
#>
```

Author(s)

David Robinson

References

Robinson, David. 2015. "Analyzing networks of characters in 'Love Actually'". http://varianceexplained.org/r/love-actually-network/

`fict_friends` *One-mode undirected Friends character scene co-appearances (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 `fict_friends` is an undirected network containing connections between characters organised by season number, which is reflected in the tie attribute 'wave'. 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(fict_friends)
```

Format

```
#> -- # Friends network ------------------------------------------------------------
#> # A labelled, weighted, undirected network of 650 characters and 2976 scene
#> co-appearance ties
#>
#> -- Nodes
#> # 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
#>
#> -- Ties
#> # A tibble: 2,976 x 3
#>    from    to weight
#>   <int> <int>  <dbl>
#> 1     8     9      3
#> 2     4    10      1
#> 3     8    12      1
#> 4     9    12      1
#> 5     2    14      1
#> 6     3    14      1
#> # i 2,970 more rows
#>
```

References

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

`fict_greys` *One-mode undirected network of characters hook-ups on Grey's Anatomy TV show*

Description

Grey's Anatomy is an American medical drama television series running on ABC since 2005. It focuses on the personal and professional lives of surgical interns, residents, and attendings at Seattle Grace Hospital, later renamed as the Grey Sloan Memorial Hospital. Gary Weissman collected data on the sexual contacts between characters on the television show through observation of the story lines in the episodes and fan pages, and this data was extended by Benjamin Lind including nodal attributes:

- 'name': first and, where available, surname
- 'sex': `F` for female and `M` for male
- 'race': `White`, `Black`, or `Other`
- 'birthyear': year born (some missing data)
- 'position': `"Chief"`, `"Attending"`, `"Resident"`, `"Intern"`, `"Nurse"`, `"Non-Staff"`, `"Other"`
- 'season': season that the character joined the show
- 'sign': character's astrological starsign, if known

The data is current up to (I think?) season 10?

Usage

```
data(fict_greys)
```

Format

```
#> -- # Grey's Anatomy -------------------------------------------------------------
#> # A labelled, undirected network of 53 characters and 56 hook-up ties
#>
#> -- Nodes
#> # A tibble: 53 x 7
#>   name               sex   race  birthyear position  season sign
#>   <chr>              <chr> <chr>     <dbl> <chr>      <dbl> <chr>
#> 1 Addison Montgomery F     White      1967 Attending      1 Libra
#> 2 Adele Webber       F     Black      1949 Non-Staff      2 Leo
#> 3 Teddy Altman       F     White      1969 Attending      6 Pisces
#> 4 Amelia Shepherd    F     White      1981 Attending      7 Libra
#> 5 Arizona Robbins    F     White      1976 Attending      5 Leo
#> 6 Rebecca Pope       F     White      1975 Non-Staff      3 Gemini
#> # i 47 more rows
#>
#> -- Ties
#> # A tibble: 56 x 2
```

```
#>    from    to
#>   <int> <int>
#> 1     5    47
#> 2    21    47
#> 3     5    46
#> 4     5    41
#> 5    18    41
#> 6    21    41
#> # i 50 more rows
#>
```

Author(s)

Gary Weissman and Benjamin Lind

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

Description

The Lord of the Rings is a beloved, epic high fantasy novel written by J.R.R. Tolkien. This is 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.

In addition, the race of these characters has been coded, though not without debate. The most contentious is the coding of Tom Bombadil and Goldberry as Maiar, presumably coded as such to avoid having categories of one.

Usage

```
data(fict_lotr)
```

Format

```
#> -- # Lord of the Rings ------------------------------------------------------
#> # A labelled, complex, undirected network of 36 characters and 66 interaction
#> ties
#>
#> -- Nodes
#> # 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
#>
#> -- Ties
#> # 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
#>
```

`fict_marvel` *Multiplex two-mode affiliation and one-mode signed relationship network of Marvel comic book characters (Yuksel 2017)*

Description

This multiplex network contains two types of ties related to the Marvel *comic book* universe. The "affiliation" ties offer a two-mode affiliation network of 53 Marvel comic book characters and their affiliations to 141 teams. The "relationship" ties offer a one-mode signed network of friendships and enmities between the 53 Marvel comic book characters. Friendships are indicated by a positive sign in the tie `sign` attribute, whereas enmities are indicated by a negative sign in this edge attribute.

Usage

```
data(fict_marvel)
```

Format

```
#>
#> -- # Marvel universe ------------------------------------------------------------
#> # A labelled, complex, multiplex, signed, two-mode network of 53 characters and
#> 141 teams and 683 affiliation and 558 relationship ties
#>
#> -- Nodes
#> # A tibble: 194 x 11
#>   type  name         Gender Appearances Attractive  Rich Intellect Omnilingual
#>   <lgl> <chr>        <chr>       <int>      <int> <int>     <int>       <int>
#> 1 FALSE Abomination  Male          427          0     0         1           1
#> 2 FALSE Ant-Man      Male          589          1     0         1           0
#> 3 FALSE Apocalypse   Male         1207          0     0         1           1
#> 4 FALSE Beast        Male         7609          1     0         1           0
#> 5 FALSE Black Panther Male        2189          1     1         1           0
```

```
#> 6 FALSE Black Widow   Female       2907         1     0        1          0
#> # i 188 more rows
#> # i 3 more variables: PowerOrigin <chr>, UnarmedCombat <int>, ArmedCombat <int>
#>
#> -- Ties
#> # A tibble: 1,241 x 4
#>    from    to type          sign
#>   <int> <int> <chr>        <dbl>
#> 1     1     1 relationship    -1
#> 2     1     4 relationship    -1
#> 3     1    11 relationship    -1
#> 4     1    12 relationship    -1
#> 5     1    23 relationship    -1
#> 6     1    24 relationship    -1
#> # i 1,235 more rows
#>
```

Details

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

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

See also https://graphics.straitstimes.com/STI/STIMEDIA/Interactives/2018/04/marvel-cinematic-universe-whos-who-interactive/index.html.

Source

Umut Yuksel, 31 March 2017

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

Description

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

Usage

```
data(fict_potter)
```

Format

```
#> -- # Harry Potter support network ----------------------------------------------
#> # A longitudinal, labelled, complex, directed network of 64 students and 544
#> support arcs over 6 waves
#>
#> -- Nodes
#> # A tibble: 64 x 5
#>   name              schoolyear gender house      active
#>   <chr>                  <int> <chr>  <chr>      <logi>
#> 1 Adrian Pucey            1989 male   Slytherin  TRUE
#> 2 Alicia Spinnet          1989 female Gryffindor TRUE
#> 3 Angelina Johnson        1989 female Gryffindor TRUE
#> 4 Anthony Goldstein       1991 male   Ravenclaw  TRUE
#> # i 60 more rows
#>
#> -- Changes
#> # A tibble: 81 x 4
#>    time  node var    value
#>   <int> <int> <chr>  <lgl>
#> 1     2     9 active TRUE
#> 2     2    21 active TRUE
#> 3     2    35 active TRUE
#> 4     2    39 active FALSE
#> # i 77 more rows
#>
#> -- Ties
#> # A tibble: 544 x 3
#>    from    to  wave
```

```
#>   <int> <int> <dbl>
#> 1    11    11     1
#> 2    11    25     1
#> 3    11    26     1
#> 4    11    44     1
#> # i 540 more rows
#>
```

References

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

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

Description

One-mode network dataset collected by Gabasova (2016) on the interactions between Star Wars characters in each movie from Episode 1 ("The Phantom Menace") to Episode 7 ("The Force Awakens"). The data was constructed by parsing the scripts, as described in https://evelinag.com/blog/2015/12-15-star-wars-social-network/index.html.

Characters are named (eg. R2-D2, Anakin, Chewbacca) and the following node attributes are provided where available: height, mass, hair color, skin color, eye color, birth year, sex, homeworld, and species. The node attribute 'faction' has also been added, denoting the faction (eg. Jedi, Rebel Alliance, etc) that Star Wars characters belong to in each episode (coding completed by Yichen Shen, Tiphaine Aeby, and James Hollway).

Weighted ties represent the number of times characters speak within the same scene of each film, indicated by the wave (1-7).

Change in the composition of the network is tracked by the variable 'active', though several other variables also change (mostly as Anakin becomes *spoiler alert*).

Usage

```
data(fict_starwars)
```

Format

```
#> -- # Star Wars network data ------------------------------------------------
#> # A longitudinal, labelled, complex, weighted, directed network of 110
#> characters and 563 interaction arcs over 7 waves
#>
#> -- Nodes
#> # A tibble: 110 x 12
#>   name          species homeworld sex   height hair_color eye_color skin_color
```

```
#>   <chr>           <chr>   <chr>     <chr>  <int> <chr>      <chr>     <chr>
#> 1 Admiral Ackbar  Mon Ca~ Mon Cala  male     180 none      orange    brown mot~
#> 2 Admiral Statura Human   Garel     male     172 black     brown     light
#> 3 Anakin          Human   Tatooine  male     188 blond     blue      fair
#> 4 Bail Organa     Human   Alderaan  male     191 black     brown     tan
#> # i 106 more rows
#> # i 4 more variables: birth_year <dbl>, mass <dbl>, faction <chr>, active <lgl>
#>
#> -- Changes
#> # A tibble: 184 x 4
#>    time  node var    value
#>   <int> <int> <chr>  <chr>
#> 1     2     7 active TRUE
#> 2     2    10 active TRUE
#> 3     2    11 active FALSE
#> 4     2    13 active FALSE
#> # i 180 more rows
#>
#> -- Ties
#> # A tibble: 563 x 4
#>    from    to weight  wave
#>   <int> <int>  <int> <int>
#> 1    80    73     11     1
#> 2    80    79     14     1
#> 3    80     3     16     1
#> 4    80   106      3     1
#> # i 559 more rows
#>
```

Details

The network for each episode may be extracted and used separately, eg. `to_time(fict_starwars, 1)` for Episode 1.

References

Gabasova, Evelina. 2016. *Star Wars social network (Version 1.0.1)*. *Zenodo*. doi:10.5281/zenodo.1411479

`fict_thrones` *One-mode Game of Thrones kinship (Glander 2017)*

Description

The original dataset was put together by Erin Pierce and Ben Kahle for an assignment for a course on Bayesian statistics. The data included information on when characters died in the Song of Ice and Fire books, and some predictive factors such as whether they were nobles, married, etc. Shirin Glander extended this data set on character deaths in the TV series Game of Thrones with

the kinship relationships between the characters, by scraping "A Wiki of Ice and Fire" and adding missing information by hand. There is certainly more that can be done here.

Usage

```
data(fict_thrones)
```

Format

```
#> -- # Game of Thrones Kinship ----------------------------------------------------
#> # A labelled, multiplex, directed network of 208 characters and 404 kinship
#> arcs
#>
#> -- Nodes
#> # A tibble: 208 x 10
#>   name         culture house  popularity Gender title birth death noble married
#>   <chr>        <fct>   <chr>       <dbl> <chr>  <chr> <int> <int> <lgl> <lgl>
#> 1 Alys Arryn    <NA>    House~     0.0803 female ""       NA    NA FALSE TRUE
#> 2 Elys Waynwood <NA>    House~     0.0702 female "Ser"    NA    NA TRUE  TRUE
#> 3 Jasper Arryn  <NA>    House~     0.0435 male   "Eyr~    NA    NA TRUE  FALSE
#> 4 Jeyne Royce   <NA>    House~      0      female  <NA>    NA    NA NA    NA
#> 5 Jon Arryn     Valemen House~     0.836  male   "Eyr~   217   298 TRUE  TRUE
#> 6 Lysa Arryn    <NA>    House~     0      female "Lad~   266   300 TRUE  TRUE
#> # i 202 more rows
#>
#> -- Ties
#> # A tibble: 404 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     2 spouse
#> 2     2     1 spouse
#> 3     3     1 parent
#> 4     3     5 parent
#> 5     4     5 spouse
#> 6     5     4 spouse
#> # i 398 more rows
#>
```

References

Pierce, Erin, and Ben Kahle. 2015. "Bayesian Survival Analysis in A Song of Ice and Fire".

Glander, Shirin. 2017. "Network analysis of Game of Thrones".

glossary *Adding network glossary items*

Description

This function adds a glossary item, useful in tutorials.

Usage

```
gloss(text, ref = NULL)

print_glossary()

clear_glossary()
```

Arguments

text	The text to appear.
ref	The name of the glossary item to index. If NULL, then the function will search the glossary for 'text' instead.

interface *Console command line interface*

Description

These functions wrap {cli} functions and elements to build an attractive command line interface (CLI).

- snet_info() for general information messages.
- snet_minor_info() for minor information messages.
- snet_warn() for warning messages.
- snet_abort() for error messages.
- snet_success() for success messages.
- snet_prompt() for prompts to the user.
- snet_unavailable() for features that are not yet available.

If you wish to receive fewer messages in the console, run options(snet_verbosity = 'quiet').

Usage

```
snet_info(..., .envir = parent.frame())

snet_minor_info(..., .envir = parent.frame())

snet_warn(..., .envir = parent.frame())

snet_abort(..., .envir = parent.frame())

snet_success(..., .envir = parent.frame())

snet_prompt(..., .envir = parent.frame())

snet_unavailable(..., .envir = parent.frame())
```

Arguments

`...`	One or more character strings. For most of these functions, if multiple strings are passed these will be pasted together.
`.envir`	This argument is just to inherit the parent frame in the (likely) event that the function is used within another function.

---

irps_911 *One-mode multiplex network of relationships between 9/11 hijackers (Krebs 2002)*

Description

This network records two different types of relationships between and surrounding the hijackers of four planes in the United States on September 11, 2001, culminating in those planes crashing into four locations: New York's World Trade Center (North and South buildings), as well as the Pentagon and a location in Somerset County, Pennsylvania.

The hijackers were members of al-Qaeda. Valdis Krebs collected further information from newspapers on the broader network of associates of these hijackers, reflecting on the challenges of collecting this information even after the fact.

The data includes two types of ties: "trust"ed prior contacts among the hijackers, and "association" ties among the hijackers but also their broader associates. All associates are named, along with a logical vector about whether they were a hijacker or not, and if so which their (eventual) target was.

Usage

```
data(irps_911)
```

Format

```
#> -- # 911 Terrorist network -----------------------------------------------------
#> # A labelled, complex, multiplex, undirected network of 60 terrorists and 153
#> association ties and 153 trust ties
#>
#> -- Nodes
#> # A tibble: 60 x 3
#>   name              hijacker target
#>   <chr>             <lgl>    <chr>
#> 1 Majed Moqed       TRUE     Pentagon
#> 2 Khalid Al Mihdhar TRUE     Pentagon
#> 3 Hani Hanjour      TRUE     Pentagon
#> 4 Nawaf Alhazmi     TRUE     Pentagon
#> 5 Salem Alhazmi     TRUE     Pentagon
#> 6 Ahmed Alnami      TRUE     Pennsylvania
#> # i 54 more rows
#>
#> -- Ties
#> # A tibble: 153 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     3 association
#> 2     1     3 trust
#> 3     2     3 association
#> 4     2     3 trust
#> 5     2     4 association
#> 6     2     4 trust
#> # i 147 more rows
#>
```

References

Krebs, Valdis. 2002. "Mapping networks of terrorist cells". *Connections* 24(3): 43-52.

`irps_blogs` *One-mode directed network of links between US political blogs (Adamic and Glance 2005)*

Description

This network consists of the blogosphere around the time of the 2004 US presidential election until February 2005. The 2004 election was the first in which blogging played a significant role. Ties were constructed from a crawl of the front page of each blog.

Political leaning is indicated as "Liberal" (or left leaning) or "Conservative" (or right leaning), sourced from blog directories. Some blogs were labelled manually, based on incoming and outgoing links and posts.

Usage

```
data(irps_blogs)
```

Format

```
#> -- # US political blogosphere circa 2004 ---------------------------------------
#> # A labelled, complex, directed network of 1490 blogs and 19090 link arcs
#>
#> -- Nodes
#> # A tibble: 1,490 x 3
#>   name                       Leaning Source
#>   <chr>                      <chr>   <chr>
#> 1 100monkeystyping.com       Liberal Blogarama
#> 2 12thharmonic.com/wordpress Liberal BlogCatalog
#> 3 40ozblog.blogspot.com      Liberal Blogarama,BlogCatalog
#> 4 4lina.tblog.com            Liberal Blogarama
#> 5 750volts.blogspot.com      Liberal Blogarama
#> 6 95theses.blogspot.com      Liberal Blogarama
#> # i 1,484 more rows
#>
#> -- Ties
#> # A tibble: 19,090 x 2
#>    from    to
#>   <int> <int>
#> 1   267  1394
#> 2   267   483
#> 3   267  1051
#> 4   904  1479
#> 5   904   919
#> 6   904  1045
#> # i 19,084 more rows
#>
```

References

Adamic, Lada, and Natalie Glance. 2005. "The political blogosphere and the 2004 US Election: Divided they blog". *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, 36-43. doi:10.1145/1134271.1134277

`irps_books` *One-mode undirected network of co-purchased books about US politics on Amazon*

Description

This network consists of books about US politics sold by Amazon.com. Ties represent books that are often purchased together, as revealed by Amazon's 'customers who bought this book also bought these other books' section on those books' pages on the website.

Information about the book's leaning "Liberal", "Neutral", or "Conservative" were added separately by Mark Newman based on the abstracts, descriptions, and reviews posted on Amazon.

These data should be cited as V. Krebs, unpublished, http://www.orgnet.com/.

Usage

```
data(irps_books)
```

Format

```
#> -- # Co-purchased US political books ----------------------------------------
#> # A labelled, undirected network of 105 books and 441 co-purchasing ties
#>
#> -- Nodes
#> # A tibble: 105 x 2
#>   name                      Leaning
#>   <chr>                     <chr>
#> 1 1000 Years for Revenge    Neutral
#> 2 Bush vs. the Beltway      Conservative
#> 3 Charlie Wilson's War      Conservative
#> 4 Losing Bin Laden          Conservative
#> 5 Sleeping With the Devil   Neutral
#> 6 The Man Who Warned America Conservative
#> # i 99 more rows
#>
#> -- Ties
#> # A tibble: 441 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     1     3
#> 3     1     4
#> 4     2     4
#> 5     1     5
#> 6     3     5
#> # i 435 more rows
#>
```

Author(s)

Valdis Krebs, Mark Newman

irps_nuclear *Two-mode dynamic discourse network of Germany's nuclear energy phase-out (Haunss and Hollway 2023)*

Description

Following the 11 March 2011 Fukushima nuclear disaster in Japan, there was a vigorous public debate in Germany about the future of nuclear energy. This network captures the discourse established by 337 actors, including individual politicians, experts, parties, and the media, and their claims about nuclear energy and German nuclear energy policy. These claims were with respect to 54 concepts coded, and could be supportive or critical, and could also be repeated.

Usage

```
data(irps_nuclear)
```

Format

```
#> -- # German nuclear discourse network ----------------------------------------
#> # A dynamic, labelled, two-mode network of 337 speakers and 54 concepts and
#> 1164 claim ties from 2011-03-11 to 2011-06-30
#>
#> -- Nodes
#> # A tibble: 391 x 10
#>   name        type  present politician govt  coalition office org   party power
#>   <chr>       <lgl> <lgl>   <lgl>     <lgl> <lgl>     <lgl>  <chr> <chr> <int>
#> 1 VfEW        FALSE TRUE    FALSE     FALSE NA        FALSE  VfEW  <NA>      0
#> 2 Angela Merk~ FALSE TRUE    TRUE      TRUE  TRUE      TRUE   CDU   31        2
#> 3 Sunday Times FALSE TRUE    FALSE     FALSE NA        FALSE  Sund~ <NA>      0
#> 4 SPD         FALSE TRUE    TRUE      FALSE FALSE     FALSE  SPD   32        1
#> 5 Norbert Röt~ FALSE TRUE    TRUE      TRUE  TRUE      TRUE   CDU   31        1
#> 6 Torsten Kra~ FALSE TRUE    FALSE     FALSE NA        FALSE  Die ~ <NA>      0
#> # i 385 more rows
#>
#> -- Ties
#> # A tibble: 1,164 x 5
#>    from    to time       increment default
#>   <int> <int> <date>         <int> <lgl>
#> 1     1   338 2011-03-11        -1 FALSE
#> 2     2   339 2011-03-12         1 TRUE
#> 3     3   340 2011-03-13        -1 FALSE
#> 4     4   341 2011-03-13         1 TRUE
#> 5     2   342 2011-03-13        -1 TRUE
#> 6     5   343 2011-03-13         1 TRUE
#> # i 1,158 more rows
#>
```

References

Haunss Sebastian, James Hollway. 2023. "Multimodal mechanisms of political discourse dynamics and the case of Germany's nuclear energy phase-out". *Network Science*, 11(2):205-223. doi:10.1017/nws.2022.31

`irps_revere` *Two-mode network of Paul Revere's (Fischer 1995)*

Description

This network is of Paul Revere and 253 of his contemporary's overlapping memberships in seven colonial organisations. The data has been collected by Kieran Healy from the appendix to David Hackett Fischer's "Paul Revere's Ride". It highlights Paul Revere's centrality in this network, and thus his ability to mobilise the towns he rode through on horseback north from Boston on the night of April 18, 1775. This is in contrast to William Dawes, who set out the same night, but south. Despite both men coming from similar class and backgrounds, and riding through towns with similar demography and political leanings, only Paul Revere was able to mobilise those he encountered, and his social network was thought key to this.

Usage

```
data(irps_revere)
```

Format

```
#> # A labelled, two-mode network of 254 nodes and 7 nodes and 319 ties
#>
#> -- Nodes
#> # A tibble: 261 x 2
#>   type  name
#>   <lgl> <chr>
#> 1 FALSE Adams.John
#> 2 FALSE Adams.Samuel
#> 3 FALSE Allen.Dr
#> 4 FALSE Appleton.Nathaniel
#> 5 FALSE Ash.Gilbert
#> 6 FALSE Austin.Benjamin
#> # i 255 more rows
#>
#> -- Ties
#> # A tibble: 319 x 2
#>    from    to
#>   <int> <int>
#> 1     1   257
#> 2     1   258
#> 3     2   257
#> 4     2   258
```

```
#> 5     2   260
#> 6     2   261
#> # i 313 more rows
#>
```

References

Fischer, David Hackett. 1995. "Paul Revere's Ride". Oxford: Oxford University Press.

Han, Shin-Kap. 2009. "The Other Ride of Paul Revere: The Brokerage Role in the Making of the American Revolution". *Mobilization: An International Quarterly*, 14(2): 143-162. doi:10.17813/maiq.14.2.g360870167085210

Healy, Kieran. 2013. "Using Metadata to find Paul Revere".

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

Description

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

Usage

```
data(irps_usgeo)
```

Format

```
#> -- # US State Contiguity ------------------------------------------------------
#> # A labelled, undirected network of 50 states and 107 contiguity ties
#>
#> -- Nodes
#> # A tibble: 50 x 3
#>   name  capitol     population
#>   <chr> <chr>            <int>
#> 1 AK    Juneau              NA
#> 2 AL    Montgomery     4780127
#> 3 AR    Little Rock    2915958
#> 4 AZ    Phoenix        6392307
#> 5 CA    Sacramento    37252895
#> 6 CO    Denver         5029324
#> # i 44 more rows
#>
#> -- Ties
#> # A tibble: 107 x 2
```

```
#>    from    to
#>   <int> <int>
#> 1     2     9
#> 2     2    10
#> 3     2    25
#> 4     2    42
#> 5     3    18
#> 6     3    24
#> # i 101 more rows
#>
```

References

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

`irps_wwi` *One-mode signed network of relationships between European major powers (Antal et al. 2006)*

Description

This network records the evolution of the major relationship changes between the protagonists of World War I (WWI) from 1872 to 1907. It is incomplete both in terms of (eventual) parties to the war as well as some other relations, but gives a good overview of the main alliances and enmities.

The data series begins with the Three Emperors' League (1872, revived in 1881) between Germany, Austria-Hungary, and Russia. The Triple Alliance in 1882 joined Germany, Austria-Hungary, and Italy into a bloc that lasted until WWI. A bilateral alliance between Germany and Russia lapsed in 1890, and a French-Russian alliance developed between 1891-1894. The Entente Cordiale thawed and then fostered relations between Great Britain and France in 1904, and a British-Russian agreement in 1907 bound Great Britain, France, and Russia into the Triple Entente.

Usage

```
data(irps_wwi)
```

Format

```
#> -- # World War I Protagonists ------------------------------------------------
#> # A dynamic, labelled, signed, undirected network of 6 European major powers
#> and 20 relationship ties from 1872 to 1918
#>
#> -- Nodes
#> # A tibble: 6 x 1
#>   name
#>   <chr>
#> 1 GBR
```

```
#> 2 FRA
#> 3 RUS
#> 4 AUH
#> 5 DEU
#> 6 ITA
#>
#> -- Ties
#> # A tibble: 20 x 5
#>    from    to  sign begin   end
#>   <int> <int> <dbl> <dbl> <dbl>
#> 1     1     2    -1  1872  1904
#> 2     1     3    -1  1872  1907
#> 3     1     4    -1  1872  1918
#> 4     2     3    -1  1872  1890
#> 5     2     4    -1  1872  1918
#> 6     2     5    -1  1872  1918
#> # i 14 more rows
#>
```

References

Antal, Tibor, Pavel Krapivsky, and Sidney Redner. 2006. "Social balance on networks: The dynamics of friendship and enmity". *Physica D* 224: 130-136. doi:10.1016/j.physd.2006.09.028

`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

```
#> -- # The Adolescent Society ---------------------------------------------------
#> # A labelled, undirected network of 8 adolescents and 10 friendships ties
#>
#> -- Nodes
#> # A tibble: 8 x 1
#>   name
#>   <chr>
```

```
#> 1 Betty
#> 2 Sue
#> 3 Alice
#> 4 Jane
#> 5 Dale
#> 6 Pam
#> # i 2 more rows
#>
#> -- Ties
#> # A tibble: 10 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     2     3
#> 3     3     4
#> 4     2     5
#> 5     3     5
#> 6     4     5
#> # i 4 more rows
#>
```

References

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

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

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

Description

Multiplex graph object of friends, social, and task ties between 16 anonymous students in an honors algebra class (M182). Each type of tie is weighted: the `friends` ties are weighted `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

```
#> -- # M182 Algebra Class ---------------------------------------------------------
#> # A multiplex, weighted, directed network of 16 nodes and 62 friendship, 129
#> social, and 88 task ties
```

```
#>
#> -- Ties
#> # A tibble: 279 x 4
#>    from    to type   weight
#>   <int> <int> <chr>   <dbl>
#> 1     1     5 social   1.2
#> 2     1     5 tasks    0.3
#> 3     1     8 social   0.15
#> 4     1     9 social   2.85
#> 5     1     9 tasks    0.3
#> 6     1    10 social   6.45
#> # i 273 more rows
#>
```

Source

See also `data(studentnets.M182, package = "NetData")`

Larger comprehensive data set publicly available, contact Daniel A. McFarland for details.

References

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

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

Description

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

Usage

```
data(ison_brandes)
```

Format

```
#> # A undirected network of 11 nodes and 12 ties
#>
#> -- Nodes
#> # A tibble: 11 x 1
#>   twomode_type
#>   <lgl>
#> 1 FALSE
#> 2 FALSE
#> 3 TRUE
```

```
#> 4 FALSE
#> 5 TRUE
#> 6 TRUE
#> # i 5 more rows
#>
#> -- 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_dolphins` *One-mode, undirected network of frequent associations in a dolphin pod (Lusseau et al. 2003)*

Description

These data contain the frequent associations between the 62 dolphins of a pod of dolphins living off Doubtful Sound, New Zealand. Additional information can be found in the literature cited below.

Usage

```
data(ison_dolphins)
```

Format

```
#> -- # Doubtful Sound dolphins ---------------------------------------------------
#> # A labelled, undirected network of 62 dolphins and 159 frequent association
#> ties
#>
#> -- Nodes
#> # A tibble: 62 x 1
#>   name
#>   <chr>
#> 1 Beak
#> 2 Beescratch
#> 3 Bumper
#> 4 CCL
#> 5 Cross
#> 6 DN16
```

```
#> # i 56 more rows
#>
#> -- Ties
#> # A tibble: 159 x 2
#>    from    to
#>   <int> <int>
#> 1     4     9
#> 2     6    10
#> 3     7    10
#> 4     1    11
#> 5     3    11
#> 6     6    14
#> # i 153 more rows
#>
```

References

Lusseau, David, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. 2003. "The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations", *Behavioral Ecology and Sociobiology* 54, 396-405.

Lusseau, David. 2003. "The emergent properties of a dolphin social network", *Proc. R. Soc. London B* 270(S): S186-S188. doi:10.1098/rsbl.2003.0057

Lusseau, David. 2007. "Evidence for social role in a dolphin social network". *Evolutionary Ecology* 21: 357–366. doi:10.1007/s1068200691050

`ison_emotions` *One-mode, weighted network of emotional transitions (Trampe et al. 2015)*

Description

Emotions are highly interconnected, and one emotion often follows another. This network describes the transitions between 18 different emotions as experienced in everyday life. The data is collected from 11,000 participants who completed daily questionnaires on the emotions they felt at a given moment. While Trampe et al. (2015) created and analysed an undirected network in their paper, the directed network constructed by Will Hipson is shared here.

Usage

```
data(ison_emotions)
```

Format

```
#> -- # Emotional transitions ------------------------------------------------
#> # A labelled, complex, weighted, directed network of 18 emotions and 315
#> transition arcs
#>
```

```
#> -- Nodes
#> # A tibble: 18 x 1
#>   name
#>   <chr>
#> 1 Alertness
#> 2 Amusement
#> 3 Anger
#> 4 Anxiety
#> 5 Awe
#> 6 Disdain
#> # i 12 more rows
#>
#> -- Ties
#> # A tibble: 315 x 3
#>    from    to weight
#>   <int> <int>  <int>
#> 1     1     1   6191
#> 2     1     2     97
#> 3     1     3    214
#> 4     1     4   4784
#> 5     1     5     14
#> 6     1     6     82
#> # i 309 more rows
#>
```

References

Trampe, Debra, Jordi Quoidbach, and Maxime Taquet. 2015. "Emotions in everyday life". *PLOS ONE*. doi:10.1371/journal.pone.0145450

Hipson, Will. 2019. https://www.r-bloggers.com/2019/03/network-analysis-of-emotions/

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

Description

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

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

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

- *age*: the manager's age in years

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

Usage

```
data(ison_hightech)
```

Format

```
#> -- # High-tech company managers ------------------------------------------------
#> # A multiplex, directed network of 21 managers and 190 advice, 102 friendship,
#> and 20 report ties
#>
#> -- Nodes
#> # A tibble: 21 x 4
#>     age tenure level dept
#>   <dbl>  <dbl> <dbl> <chr>
#> 1    33      9     1 E
#> 2    42     20     2 E
#> 3    40     13     1 C
#> 4    33      8     1 E
#> 5    32      3     1 C
#> 6    59     28     1 B
#> # i 15 more rows
#>
#> -- Ties
#> # A tibble: 312 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     2 friends
#> 2     1     2 advice
#> 3     1     2 reports
#> 4     1     4 friends
#> 5     1     4 advice
#> 6     1     8 friends
#> # i 306 more rows
#>
```

References

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

`ison_judo_moves` *One-mode judo moves network (Bastazini 2025)*

Description

Judo is a martial art with a long history and many different techniques. It involves a dynamic 'chess match' of throws, holds, locks, submission techniques, and other maneuvers. The techniques are often combined in sequences to create fluid and effective combinations to score points or achieve victory. As the author of this network describes, "While individual techniques (called waza) are foundational, the real artistry lies in how they are chained together – through renraku-waza (combination techniques) and renzoku-waza (continuous combination techniques)" This network describes the relationships between 33 individual judo moves, as recognised by the Kodokan (the official international governing body of judo), where an arc indicates that one move can be followed by another.

Usage

```
data(ison_judo_moves)
```

Format

```
#> -- # Judo attack combinations ------------------------------------------------
#> # A labelled, complex, directed network of 33 attacks and 81 sequence arcs
#>
#> -- Nodes
#> # A tibble: 33 x 1
#>   name
#>   <chr>
#> 1 Seoi.nage
#> 2 Seoi.otoshi
#> 3 O.uchi.gari
#> 4 Ko.uchi.gari
#> 5 Ippon.seoi.nage
#> 6 Osoto.gari
#> # i 27 more rows
#>
#> -- Ties
#> # A tibble: 81 x 2
#>    from    to
#>   <int> <int>
#> 1     1     2
#> 2     1     3
#> 3     1     4
#> 4     3     4
#> 5     3     6
#> 6     3     7
#> # i 75 more rows
#>
```

References

Bastazini, Vinicius. 2025. "The Dynamics of the “Gentle Way”: Exploring Judo Attack Combinations as Networks in R", https://geekcologist.wordpress.com/2025/05/27/the-dynamics-of-the-gentle-way-exploring-judo-attack-combinations-as-networks-in-r/

Kashiwazaki, Katsuhiko, and Hidetoshi Nakanishi. 1995. *Attacking Judo: A Guide to Combinations and Counters*. Ippon Books.

Kawaishi, Mikinosuke. 1963. *Standing judo: The combinations and counter-attacks*. Budoworks.

van Haesendonck, F.M. 1968. *Judo: Ecyclopédie par l'Image*. Éditions Erasme: Anvers-Bruxelles.

`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

```
#> -- # Zachary's karate club network ------------------------------------------
#> # A labelled, weighted, undirected network of 34 club members and 78
#> association ties
#>
#> -- Nodes
#> # A tibble: 34 x 2
#>   name  allegiance
#>   <chr>      <dbl>
#> 1 Mr Hi          1
#> 2 2              1
#> 3 3              1
#> 4 4              1
#> 5 5              1
#> 6 6              1
#> # i 28 more rows
#>
#> -- Ties
#> # A tibble: 78 x 3
#>    from    to weight
#>   <int> <int>  <dbl>
```

```
#> 1     1     2     4
#> 2     1     3     5
#> 3     2     3     6
#> 4     1     4     3
#> 5     2     4     3
#> 6     3     4     3
#> # i 72 more rows
#>
```

References

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

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

Description

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

Usage

```
data(ison_koenigsberg)
```

Format

```
#> -- # Seven Bridges of Koenigsberg network ------------------------------------
#> # A labelled, undirected network of 4 landmasses and 7 bridge ties
#>
#> -- Nodes
#> # 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
#>
#> -- Ties
```

```
#> # A tibble: 7 x 3
#>    from    to name
#>   <int> <int> <chr>
#> 1     1     2 Kraemer Bruecke
#> 2     1     2 Schmiedebruecke
#> 3     1     3 Holzbruecke
#> 4     2     3 Honigbruecke
#> 5     2     4 Gruene Bruecke
#> 6     2     4 Koettelbruecke
#> # i 1 more row
#>
```

Source

{igraphdata}

References

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

---

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

Description

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

Usage

```
data(ison_laterals)
```

Format

```
#> $ison_bb
#> # A labelled, two-mode network of 4 nodes and 6 nodes and 12 ties
#>
#> -- Nodes
#> # A tibble: 10 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 B     FALSE
#> 3 C     FALSE
#> 4 D     FALSE
#> 5 U     TRUE
```

```
#> 6 V     TRUE
#> # i 4 more rows
#>
#> -- Ties
#> # A tibble: 12 x 2
#>    from    to
#>   <int> <int>
#> 1     1     5
#> 2     1     6
#> 3     2     5
#> 4     2     7
#> 5     2     8
#> 6     2     9
#> # i 6 more rows
#>
#>
#> $ison_bm
#> # A labelled, two-mode network of 4 nodes and 4 nodes and 9 ties
#>
#> -- Nodes
#> # A tibble: 8 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 B     FALSE
#> 3 C     FALSE
#> 4 D     FALSE
#> 5 U     TRUE
#> 6 V     TRUE
#> # i 2 more rows
#>
#> -- Ties
#> # A tibble: 9 x 2
#>    from    to
#>   <int> <int>
#> 1     1     5
#> 2     1     6
#> 3     2     5
#> 4     2     7
#> 5     2     8
#> 6     3     6
#> # i 3 more rows
#>
#>
#> $ison_mb
#> # A labelled, two-mode network of 4 nodes and 4 nodes and 9 ties
#>
#> -- Nodes
```

```
#> # A tibble: 8 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 B     FALSE
#> 3 C     FALSE
#> 4 D     FALSE
#> 5 M     TRUE
#> 6 X     TRUE
#> # i 2 more rows
#>
#> -- Ties
#> # A tibble: 9 x 2
#>    from    to
#>   <int> <int>
#> 1     1     5
#> 2     2     5
#> 3     2     6
#> 4     2     7
#> 5     3     5
#> 6     3     6
#> # i 3 more rows
#>
#>
#> $ison_mm
#> # A labelled, two-mode network of 4 nodes and 2 nodes and 6 ties
#>
#> -- Nodes
#> # A tibble: 6 x 2
#>   name  type
#>   <chr> <lgl>
#> 1 A     FALSE
#> 2 B     FALSE
#> 3 C     FALSE
#> 4 D     FALSE
#> 5 M     TRUE
#> 6 N     TRUE
#>
#> -- Ties
#> # A tibble: 6 x 2
#>    from    to
#>   <int> <int>
#> 1     1     5
#> 2     2     5
#> 3     2     6
#> 4     3     5
#> 5     3     6
#> 6     4     6
```

```
#>
```

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

Description

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

Usage

```
data(ison_lawfirm)
```

Format

```
#> -- # Lazega's Lawyers ------------------------------------------------------------
#> # A multiplex, directed network of 71 attorneys and 892 advice, 1104 cowork,
#> and 575 friendship ties
#>
#> -- Nodes
#> # A tibble: 71 x 7
#>   status  gender office   seniority   age practice   school
#>   <chr>   <chr>  <chr>        <dbl> <dbl> <chr>      <chr>
#> 1 partner man    Boston          31    64 litigation Harvard/Yale
#> 2 partner man    Boston          32    62 corporate  Harvard/Yale
#> 3 partner man    Hartford        13    67 litigation Harvard/Yale
#> 4 partner man    Boston          31    59 corporate  Other
#> 5 partner man    Hartford        31    59 litigation UConn
#> 6 partner man    Hartford        29    55 litigation Harvard/Yale
#> # i 65 more rows
#>
#> -- Ties
#> # A tibble: 2,571 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     2 friends
#> 2     1     2 advice
#> 3     1     4 friends
#> 4     1     8 friends
#> 5     1    17 friends
#> 6     1    17 advice
#> # i 2,565 more rows
#>
```

Details

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

Source

{networkdata}

References

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

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

Description

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

- `type = "like"` records whom novices said they liked most at three time points/waves
- `type = "esteem"` records whom novices said they held in esteem (sign > 0) and disesteem (sign < 0)
- `type = "praise"` records whom novices said they praised (sign > 0) and blamed (sign < 0)
- `type = "influence"` records whom novices said were a positive influence (sign > 0) and negative influence (sign < 0)

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

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

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

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

Usage

```
data(ison_monks)
```

Format

```
#> -- # Sampson's Monks ------------------------------------------------------------
#> # A longitudinal, labelled, multiplex, signed, weighted, directed network of 18
#> nodes and 112 esteem, 103 influence, 168 like, and 80 praise ties over 3 waves
#>
#> -- Nodes
#> # A tibble: 18 x 3
#>   name        groups        left
#>   <chr>       <chr>        <dbl>
#> 1 Romuald     Interstitial     3
#> 2 Bonaventure Loyal            4
#> 3 Ambrose     Loyal            4
#> 4 Berthold    Loyal            4
#> 5 Peter       Loyal            3
#> 6 Louis       Loyal            4
#> # i 12 more rows
#>
#> -- Ties
#> # A tibble: 463 x 6
#>    from    to  sign type  weight  wave
#>   <int> <int> <dbl> <chr>  <dbl> <dbl>
#> 1     1     2     1 like       1     2
#> 2     1     2     1 like       1     3
#> 3     1     3     1 like       1     3
#> 4     1     5     1 like       3     1
#> 5     1     5     1 like       3     2
#> 6     1     5     1 like       3     3
#> # i 457 more rows
#>
```

References

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

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

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

Description

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

Usage

```
data(ison_networkers)
```

Format

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

Source

networkdata package

References

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

Irvine CA, University of California.

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

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

Description

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

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

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

Additional questions and records of prescriptions provided additional information:

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

Usage

```
data(ison_physicians)
```

Format

```
#> $Peoria
#> # A multiplex, directed network of 117 nodes and 242 advice, 154 discussion,
#> and 147 friendship ties
#>
#> -- Nodes
#> # A tibble: 117 x 12
#>   adoption specialty    conferences journals practice   community patients
#>      <dbl> <chr>        <chr>          <dbl> <chr>      <chr>     <chr>
#> 1        1 Pediatrician Specialty          9 1920..1929 20+yrs    101-150
#> 2       12 GP           None               5 1945..     -1yr      76-100
#> 3        8 Internist    General            7 1935..1939 10-20yrs  76-100
#> 4        9 GP           General            6 1940..1944 5-10yrs   51-75
#> 5        9 GP           General            4 1935..1939 10-20yrs  51-75
#> 6       10 Internist    None               7 1930..1934 10-20yrs  101-150
#> # i 111 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#>
#> -- Ties
#> # A tibble: 543 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     8 friendship
#> 2     1    58 friendship
#> 3     1    87 advice
#> 4     1    90 advice
#> 5     1   110 advice
#> 6     1   112 friendship
#> # i 537 more rows
#>
#>
#> $Bloomington
#> # A multiplex, directed network of 50 nodes and 94 advice, 57 discussion, and
#> 60 friendship ties
#>
#> -- Nodes
#> # A tibble: 50 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1       98 Internist Specialty          8 1930..1934 10-20yrs  101-150
#> 2        1 GP        General            3 1945..     5-10yrs   76-100
#> 3       98 GP        Specialty          4 1930..1934 10-20yrs  101-150
#> 4        7 Internist None               3 1945..     -1yr      26-50
#> 5        6 Internist General            9 1935..1939 5-10yrs   76-100
#> 6        1 GP        Specialty          5 1935..1939 10-20yrs  101-150
#> # i 44 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
```

```
#> #   doc_club <dbl>, doc_proximity <chr>
#>
#> -- Ties
#> # A tibble: 211 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     3 friendship
#> 2     1    10 discussion
#> 3     1    24 advice
#> 4     1    44 advice
#> 5     2     4 advice
#> 6     2     6 advice
#> # i 205 more rows
#>
#>
#> $Quincy
#> # A multiplex, directed network of 44 nodes and 70 advice, 52 discussion, and
#> 52 friendship ties
#>
#> -- Nodes
#> # A tibble: 44 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1        2 Internist None               6 1935..1939 10-20yrs  151+
#> 2       18 GP        General            3 1920..1929 20+yrs    151+
#> 3       18 Internist None               5 1945..     -1yr      -25
#> 4        4 GP        General            3 1930..1934 20+yrs    151+
#> 5       18 GP        Specialty          4 1935..1939 10-20yrs  151+
#> 6        5 Internist General            5 ..1919     20+yrs    51-75
#> # i 38 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#>
#> -- Ties
#> # A tibble: 174 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     8 advice
#> 2     1     9 advice
#> 3     1    10 discussion
#> 4     1    13 friendship
#> 5     1    15 advice
#> 6     1    22 discussion
#> # i 168 more rows
#>
#>
#> $Galesburg
#> # A multiplex, directed network of 35 nodes and 74 advice, 46 discussion, and
```

```
#> 51 friendship ties
#>
#> -- Nodes
#> # A tibble: 35 x 12
#>   adoption specialty conferences journals practice   community patients
#>      <dbl> <chr>     <chr>          <dbl> <chr>      <chr>     <chr>
#> 1       18 GP        General            4 1935..1939 5-10yrs   101-150
#> 2       18 GP        None               4 1935..1939 -1yr      151+
#> 3        4 GP        General            6 1945..     2-5yrs    51-75
#> 4        5 GP        None               4 1935..1939 10-20yrs  101-150
#> 5        8 Internist General            6 1935..1939 5-10yrs   151+
#> 6        4 Internist Specialty          8 ..1919     20+yrs    76-100
#> # i 29 more rows
#> # i 5 more variables: doc_freetime <dbl>, doc_discuss <dbl>, doc_friends <dbl>,
#> #   doc_club <dbl>, doc_proximity <chr>
#>
#> -- Ties
#> # A tibble: 171 x 3
#>    from    to type
#>   <int> <int> <chr>
#> 1     1     5 advice
#> 2     1     6 advice
#> 3     1    20 discussion
#> 4     1    23 discussion
#> 5     1    30 friendship
#> 6     1    31 friendship
#> # i 165 more rows
#>
```

Source

{networkdata}

References

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

`ison_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 pattern of a group of women's participation at informal social events in Old City during a 9 month period, as reported in the *Old City Herald* in 1936. By convention, the nodes are named by the women's first names and the code numbers of the events, but the women's surnames and titles (Miss, Mrs.) are

recorded here too. The events' dates are recorded in place of the Surname, and these dates are also offered as a tie attribute.

Usage

```
data(ison_southern_women)
```

Format

```
#> -- # Southern Women Data ------------------------------------------------------
#> # A labelled, two-mode network of 18 womens and 14 social events and 89
#> participation ties
#>
#> -- Nodes
#> # A tibble: 32 x 4
#>   type  name      Surname    Title
#>   <lgl> <chr>     <chr>      <chr>
#> 1 FALSE Evelyn    Jefferson  Mrs
#> 2 FALSE Laura     Mandeville Miss
#> 3 FALSE Theresa   Anderson   Miss
#> 4 FALSE Brenda    Rogers     Miss
#> 5 FALSE Charlotte McDowd     Miss
#> 6 FALSE Frances   Anderson   Miss
#> # i 26 more rows
#>
#> -- Ties
#> # A tibble: 89 x 3
#>    from    to date
#>   <int> <int> <date>
#> 1    14    29 1936-02-23
#> 2    15    29 1936-02-23
#> 3    17    29 1936-02-23
#> 4    18    29 1936-02-23
#> 5     1    23 1936-02-25
#> 6     2    23 1936-02-25
#> # i 83 more rows
#>
```

References

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

`make_cran` *Making networks of inter- and intra-package dependencies*

Description

These functions read information from CRAN or within an R package's working directory to create a networks of a package's dependencies:

- `read_cran()` creates a network of a package's dependencies on other packages available on CRAN. It looks for the `Depends`, `Imports` and `Suggests` fields in the package's DESCRIPTION file and creates a network where nodes are packages and ties are dependencies.
- `read_pkg()` creates a network of function dependencies from R scripts in a directory. It looks for function definitions and function calls within the scripts and creates a network where nodes are functions and edges are function calls. It can also include function calls to functions not defined within the scripts.

Usage

```
collect_cran(pkg = "all")

collect_pkg(dir = getwd())
```

Arguments

`pkg`	The name
`dir`	Character string or vector of character strings with the directory or directories to search for R scripts. If `NULL` (the default), the current working directory is used.

Details

Note that these functions are not as actively maintained as others in the package, so please let us know if any are not currently working for you or if there are missing import routines by raising an issue on Github.

Value

A `tidygraph` object representing the network of package dependencies or function dependencies in a package.

Author(s)

Jakob Gepp

Source

https://www.r-bloggers.com/2016/01/r-graph-objects-igraph-vs-network/

https://github.com/STATWORX/helfRlein/blob/master/R/get_network.R

See Also

as

Other makes: `make_create`, `make_ego`, `make_explicit`, `make_learning`, `make_motifs`, `make_play`, `make_random`, `make_read`, `make_stochastic`, `make_write`

Examples

```
# mnet <- collect_cran()
# mnet <- to_ego(mnet, "manynet", max_dist = 2)
# mnet <- collect_pkg()
```

`make_create` *Making networks with defined structures*

Description

These functions create networks with particular structural properties.

- `create_empty()` creates an empty network without any ties.
- `create_filled()` creates a filled network with every possible tie realised.
- `create_ring()` creates a ring or chord network where each nodes' neighbours form a clique.
- `create_star()` creates a network with a maximally central node.
- `create_tree()` creates a network with successive branches.
- `create_lattice()` creates a network that forms a regular tiling.
- `create_components()` creates a network that clusters nodes into separate components.
- `create_core()` creates a network in which a certain proportion of 'core' nodes are densely tied to each other, and the rest peripheral, tied only to the core.
- `create_degree()` creates a network with a given (out/in)degree sequence, which can also be used to create k-regular networks.

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

Usage

```
create_empty(n, directed = FALSE)

create_filled(n, directed = FALSE)

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

create_star(n, directed = FALSE)

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

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

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

create_degree(n, outdegree = NULL, indegree = NULL)

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

create_windmill(n)

create_cycle(n, directed = FALSE)

create_wheel(n, directed = FALSE)
```

Arguments

n	Given: • A single integer, e.g. n = 10, a one-mode network will be created. • A vector of two integers, e.g. n = c(5,10), a two-mode network will be created. • A manynet-compatible object, a network of the same dimensions will be created.
directed	Logical whether the graph should be directed. By default directed = FALSE. If the opposite direction is desired, use to_redirected() on the output of these functions.
width	Integer specifying the width of the ring, breadth of the branches, or maximum extent of the neighbourbood.
...	Additional arguments passed on to igraph::make_ring().
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.
outdegree	Numeric scalar or vector indicating the desired outdegree distribution. By default NULL and is required. If n is an existing network object and the outdegree is not specified, then the outdegree distribution will be inferred from that of the network. Note that a scalar (single number) will result in a k-regular graph.
indegree	Numeric vector indicating the desired indegree distribution. By default NULL but not required unless a directed network is desired. If n is an existing directed network object and the indegree is not specified, then the indegree distribution will be inferred from that of the network.
mark	A logical vector the length of the nodes in the network. This can be created by, among other things, any node_is_*() function.

Value

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

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

Lattice graphs

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

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

See Also

as

Other makes: `make_cran`, `make_ego`, `make_explicit`, `make_learning`, `make_motifs`, `make_play`, `make_random`, `make_read`, `make_stochastic`, `make_write`

Examples

```
create_empty(10)
create_filled(10)
create_ring(8, width = 2)
create_star(12)
create_tree(c(7,8))
create_lattice(12, width = 4)
create_components(10, membership = c(1,1,1,2,2,2,3,3,3,3))
create_degree(10, outdegree = rep(1:5, 2))
create_core(6)
  create_windmill(6)
  create_cycle(6)
  create_wheel(6)
```

---

make_ego *Making ego networks through interviewing*

Description

This function creates an ego network through interactive interview questions. It currently only supports a simplex, directed network of one or two modes. These directed networks can be reformatted as undirected using `to_undirected()`. Multiplex networks can be collected separately and then joined together afterwards.

The function supports the use of rosters or a maximum number of alters to collect. If a roster is provided it will offer ego all names. The function can also prompt ego to interpret each node's attributes, or about how ego considers their alters to be related.

Usage

```
collect_ego(
  ego = NULL,
  max_alters = Inf,
  roster = NULL,
  interpreter = FALSE,
  interrelater = FALSE,
  twomode = FALSE
)
```

Arguments

ego	A character string. If desired, the name of ego can be declared as an argument. Otherwise the first prompt of the function will be to enter a name for ego.
max_alters	The maximum number of alters to collect. By default infinity, but many name generators will expect a maximum of e.g. 5 alters to be named.
roster	A vector of node names to offer as potential alters for ego.
interpreter	Logical. If TRUE, then it will ask for which attributes to collect and give prompts for each attribute for each node in the network. By default FALSE.
interrelater	Logical. If TRUE, then it will ask for the contacts from each of the alters perspectives too.
twomode	Logical. If TRUE, then it will assign ego to the first mode and all alters to a second mode.

See Also

Other makes: make_cran, make_create, make_explicit, make_learning, make_motifs, make_play, make_random, make_read, make_stochastic, make_write

---

make_explicit *Making networks with explicit ties*

Description

This function creates a network from a vector of explicitly named nodes and ties between them. create_explicit() largely wraps igraph::graph_from_literal(), but will also accept character input and not just a formula, and will never simplify the result.

Ties are indicated by -, and directed ties (arcs) require + at either or both ends. Ties are separated by commas, and isolates can be added as an additional, unlinked node after the comma within the formula. Sets of nodes can be linked to other sets of nodes through use of a semi-colon. See the example for a demonstration.

Usage

```
create_explicit(...)
```

Arguments

...	Arguments passed on to {igraph}.

See Also

Other makes: make_cran, make_create, make_ego, make_learning, make_motifs, make_play, make_random, make_read, make_stochastic, make_write

Examples

```
create_explicit(A -+ B, B -+ C, A +-+ C, D, E:F:G-+A, E:F+-+G:H)
```

---

make_learning *Making learning models on networks*

Description

These functions allow learning games to be played upon networks.

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

Usage

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

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

Arguments

.data	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
beliefs	A vector indicating the probabilities nodes put on some outcome being 'true'.
closeness	A threshold at which beliefs are too different to influence each other. By default Inf, i.e. there is no threshold.

`steps` The number of steps forward in learning. By default the number of nodes in the network.

`epsilon` The maximum difference in beliefs accepted for convergence to a consensus.

`attribute` A string naming some nodal attribute in the network. Currently only tested for binary attributes.

`heterophily` A score ranging between -1 and 1 as a threshold for how heterophilous nodes will accept their neighbours to be. A single proportion means this threshold is shared by all nodes, but it can also be a vector the same length of the nodes in the network for issuing different thresholds to different nodes. By default this is 0, meaning nodes will be dissatisfied if more than half of their neighbours differ on the given attribute.

`who_moves` One of the following options: "ordered" (the default) checks each node in turn for whether they are dissatisfied and there is an available space that they can move to, "random" will check a node at random, and "most_dissatisfied" will check (one of) the most dissatisfied nodes first.

`choice_function`
One of the following options: "satisficing" (the default) will move the node to any coordinates that satisfy their heterophily threshold, "optimising" will move the node to coordinates that are most homophilous, and "minimising" distance will move the node to the next nearest unoccupied coordinates.

Learning models

The default is a Degroot learning model, but if `closeness` is defined as anything less than infinity, this becomes a Deffuant model. A Deffuant model is similar to a Degroot model, however nodes only learn from other nodes whose beliefs are not too dissimilar from their own.

References

DeGroot, Morris H. 1974. "Reaching a consensus", *Journal of the American Statistical Association*, 69(345): 118–21. doi:10.1080/01621459.1974.10480137

Deffuant, Guillaume, David Neau, Frederic Amblard, and Gérard Weisbuch. 2000. "Mixing beliefs among interacting agents", *Advances in Complex Systems*, 3(1): 87-98. doi:10.1142/S0219525900000078

Golub, Benjamin, and Matthew O. Jackson 2010. "Naive learning in social networks and the wisdom of crowds", *American Economic Journal*, 2(1): 112-49. doi:10.1257/mic.2.1.112

See Also

Other makes: `make_cran`, `make_create`, `make_ego`, `make_explicit`, `make_motifs`, `make_play`, `make_random`, `make_read`, `make_stochastic`, `make_write`

Other models: `make_play`

Examples

```
play_learning(ison_networkers,
    rbinom(net_nodes(ison_networkers),1,prob = 0.25))
startValues <- rbinom(100,1,prob = 0.5)
```

```
startValues[sample(seq_len(100), round(100*0.2))] <- NA
latticeEg <- create_lattice(100)
latticeEg <- add_node_attribute(latticeEg, "startValues", startValues)
latticeEg
play_segregation(latticeEg, "startValues", 0.5)
```

`make_mnet` *Multilevel, multiplex, multimodal, signed, dynamic or longitudinal changing networks*

Description

The 'mnet' class of network object is an additional class layered on top of the 'igraph' and 'tbl_graph' classes. Under the hood it is an 'igraph' object, which enables all the igraph functions to operate. It is also a 'tbl_graph' object, which enables it to be used with {ggraph}. However, 'mnet' objects offer prettier printing and a consistent structure that enables more complex forms of networks to be contained in a single object.

Usage

```
## S3 method for class 'mnet'
print(x, ..., n = 12)

print_all(x, ...)
```

Arguments

`x`	An object of class "mnet" or "tbl_graph".
`...`	Other arguments passed to or from other methods.
`n`	Number of observations to print across all network components, i.e. nodes, changes, and ties. By default 12.

Nodes

Nodes are held as vertices and vertex attributes in the 'igraph' object, but printed as a nodelist. Here the convention is for the first column of the nodelist to be called 'name' and records the labels of the nodes. Additional reserved columns include 'active' for changing networks and 'type' for multimodal networks.

Changes

Changes, that is a list of changes to the nodes in the network, are held internally as a graph attribute in the 'igraph' object, but printed as a changelist. Here the convention is for the 'wave' or 'time' column to appear first, followed by 'node' indicating to which node the change applies, 'var' for the variable to which the change applies, and 'value' for the new value to be applied.

Ties

Ties are held as edges and edge attributes in the 'igraph' object, but printed as an edgelist. Here the convention is for the first column of the edgelist to be called 'from' and the second column 'to', even if the network is not directed. Additional reserved columns include 'weight' for weighted networks, 'wave' for longitudinal networks, 'type' for multiplex networks, and 'sign' for signed networks.

Printing

When printed, 'mnet' objects will print to the console any information stored about the network's name, or its types of nodes or ties. It will also describe key features of the network, such as whether the network is multiplex, weighted, directed, etc.

It will then print tibbles for the nodes, changes, and ties in the network, as appropriate. That is, if there is no nodal data (e.g. it is an unlabelled network without any other nodal attributes), then this will be skipped. Similarly, if no nodal changes are logged, this information will be skipped too.

`make_motifs` *Making motifs*

Description

`create_motifs()` is used to create a list of networks that represent the subgraphs or motifs corresponding to a certain number of nodes and direction. Note that currently only `n==2` to `n==4` is implemented, and the latter only for undirected networks.

Usage

```
create_motifs(n, directed = FALSE)
```

Arguments

`n`	Given: • A single integer, e.g. `n = 10`, a one-mode network will be created. • A vector of two integers, e.g. `n = c(5,10)`, a two-mode network will be created. • A manynet-compatible object, a network of the same dimensions will be created.
`directed`	Logical whether the graph should be directed. By default `directed = FALSE`. If the opposite direction is desired, use `to_redirected()` on the output of these functions.

See Also

Other makes: `make_cran`, `make_create`, `make_ego`, `make_explicit`, `make_learning`, `make_play`, `make_random`, `make_read`, `make_stochastic`, `make_write`

make_play *Making diffusion models on networks*

Description

These functions simulate diffusion or compartment models upon a network.

- play_diffusion() runs a single simulation of a compartment model, allowing the results to be visualised and examined.

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

Usage

```
play_diffusion(
  .data,
  seeds = 1,
  contact = NULL,
  prevalence = 0,
  thresholds = 1,
  transmissibility = 1,
  latency = 0,
  recovery = 0,
  waning = 0,
  fatality = 0,
  immune = NULL,
  steps,
  old_version = FALSE
)
```

Arguments

.data	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
seeds	A valid mark vector the length of the number of nodes in the network.
contact	A matrix or network that replaces ".data" with some other explicit contact network, e.g. create_components(.data, membership = node_in_structural(.data)). Can be of arbitrary complexity, but must of the same dimensions as .data.
prevalence	The proportion that global prevalence contributes to diffusion. That is, if prevalence is 0.5, then the current number of infections is multiplied by 0.5 and added "prevalence" is 0 by default, i.e. there is no global mechanism. Note that this is endogenously defined and is updated at the outset of each step.

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

`transmissibility` The transmission rate probability, β. By default 1, which means any node for which the threshold is met or exceeded will become infected. Anything lower means a correspondingly lower probability of adoption, even when the threshold is met or exceeded.

`latency` The inverse probability those who have been exposed become infectious (infected), σ or κ. For example, if exposed individuals take, on average, four days to become infectious, then $\sigma = 0.75$ (1/1-0.75 = 1/0.25 = 4). By default 0, which means those exposed become immediately infectious (i.e. an SI model). Anything higher results in e.g. a SEI model.

`recovery` The probability those who are infected recover, γ. For example, if infected individuals take, on average, four days to recover, then $\gamma = 0.25$. By default 0, which means there is no recovery (i.e. an SI model). Anything higher results in an SIR model.

`waning` The probability those who are recovered become susceptible again, ξ. For example, if recovered individuals take, on average, four days to lose their immunity, then $\xi = 0.25$. By default 0, which means any recovered individuals retain lifelong immunity (i.e. an SIR model). Anything higher results in e.g. a SIRS model. $\xi = 1$ would mean there is no period of immunity, e.g. an SIS model.

`fatality` The probability those who are infected are removed from the network, α. Note that fatality is distinct from a natural mortality rate. By default $\alpha = 0$, which means that there is no fatality. Where $\alpha > 0$, the nodal attribute 'active' will be added if it is not already present.

`immune` A logical or numeric vector identifying nodes that begin the diffusion process as already recovered. This could be interpreted as those who are vaccinated or equivalent. Note however that a waning parameter will affect these nodes too. By default NULL, indicating that no nodes begin immune.

`steps` The number of steps forward in the diffusion to play. By default the number of nodes in the network. If `steps = Inf` then the diffusion process will continue until there are no new infections or all nodes are infected.

`old_version` This is included to maintain backward compatibility with the old version of this function, that would return a special object. The new version adds the diffusion event record as changes to the original network.

Simple and complex diffusion

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

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

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

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

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

Compartment models

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

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

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

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

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

See Also

Other makes: `make_cran`, `make_create`, `make_ego`, `make_explicit`, `make_learning`, `make_motifs`, `make_random`, `make_read`, `make_stochastic`, `make_write`

Other models: `make_learning`

Examples

```
smeg <- generate_smallworld(15, 0.025)
```

`make_random` *Making unconditional and conditional random networks*

Description

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

- `generate_random()` generates a random network with ties appearing at some probability.
- `generate_configuration()` generates a random network consistent with a given degree distribution.
- `generate_man()` generates a random network conditional on the dyad census of Mutual, Asymmetric, and Null dyads, respectively.
- `generate_utilities()` generates a random utility matrix.

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

Usage

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

generate_configuration(.data)

generate_man(n, man = NULL)

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.

`.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
`man`	Vector of Mutual, Asymmetric, and Null dyads, respectively. Can be specified as proportions, e.g. `c(0.5, 0.5, 0.5)`, or as a count, e.g. `c(10,0,20)`. Is inferred from `n` if it is an existing network object.
`steps`	Number of simulation steps to run. By default 1: a single, one-shot simulation. If more than 1, further iterations will update the utilities depending on the values of the volatility and threshold parameters.
`volatility`	How much change there is between steps. Only if volatility is more than 1 do further simulation steps make sense. This is passed on to `stats::rnorm` as the `sd` or standard deviation parameter.
`threshold`	This parameter can be used to mute or disregard stepwise changes in utility that are minor. The default 0 will recognise all changes in utility, but raising the threshold will mute any changes less than this threshold.

Value

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

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

References

On random networks:

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

On configuration models:

Bollobas, Bela. 1980. "A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs". *European Journal of Combinatorics* 1: 311-316.

On dyad-census conditioned networks:

Holland, Paul W., and Samuel Leinhardt. 1976. “Local Structure in Social Networks.” In D. Heise (Ed.), *Sociological Methodology*, pp 1-45. San Francisco: Jossey-Bass.

See Also

Other makes: `make_cran`, `make_create`, `make_ego`, `make_explicit`, `make_learning`, `make_motifs`, `make_play`, `make_read`, `make_stochastic`, `make_write`

Examples

```
generate_random(12, 0.4)
# generate_random(c(6, 6), 0.4)
```

`make_read` *Making networks from external files*

Description

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

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

Usage

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

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

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

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

read_ucinet(file = file.choose())

read_dynetml(file = file.choose())

read_graphml(file = file.choose())

read_gml(file = file.choose())

read_gdf(file = file.choose())
```

Arguments

file	A character string with the system path to the file to import. If left unspecified, an OS-specific file picker is opened to help users select it. Note that in read_ucinet() the file path should be to the header file (.##h), if it exists and that it is currently not possible to import multiple networks from a single UCINET file. Please convert these one by one.
sv	Allows users to specify whether their csv file is "comma" (English) or "semi-colon" (European) separated.
...	Additional parameters passed to the read/write function.
ties	A character string indicating the ties/network, where the data contains several.

Details

Note that these functions are not as actively maintained as others in the package, so please let us know if any are not currently working for you or if there are missing import routines by raising an issue on Github.

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

- UCINET data
- networkdata
- GML datasets
- UCIrvine Network Data Repository
- SNAP Stanford Large Network Dataset Collection

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

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

Value

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

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

Source

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

See Also

as

Other makes: make_cran, make_create, make_ego, make_explicit, make_learning, make_motifs, make_play, make_random, make_stochastic, make_write

---

`make_stochastic` *Making networks with a stochastic element*

Description

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

- `generate_smallworld()` generates a small-world structure via ring rewiring at some probability.
- `generate_scalefree()` generates a scale-free structure via preferential attachment at some probability.
- `generate_fire()` generates a forest fire model.
- `generate_islands()` generates an islands model.
- `generate_citations()` generates a citations model.

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

Usage

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

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

generate_fire(n, contacts = 1, their_out = 0, their_in = 1, directed = FALSE)

generate_islands(n, islands = 2, p = 0.5, bridges = 1, directed = FALSE)

generate_citations(
  n,
  ties = sample(1:4, 1),
  agebins = max(1, n/10),
  directed = FALSE
)
```

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.
width	Integer specifying the width of the ring, breadth of the branches, or maximum extent of the neighbourbood.
contacts	Number of contacts or ambassadors chosen from among existing nodes in the network. By default 1. See igraph::sample_forestfire().
their_out	Probability of tieing to a contact's outgoing ties. By default 0.
their_in	Probability of tieing to a contact's incoming ties. By default 1.
islands	Number of islands or communities to create. By default 2. See igraph::sample_islands() for more.
bridges	Number of bridges between islands/communities. By default 1.
ties	Number of ties to add per new node. By default a uniform random sample from 1 to 4 new ties.
agebins	Number of aging bins. By default either $\frac{n}{10}$ or 1, whichever is the larger. See igraphr::sample_last_cit() for more.

Value

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

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

References

On small-world networks:

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

On scale-free networks:

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

On the forest-fire model:

Leskovec, Jure, Jon Kleinberg, and Christos Faloutsos. 2007. "Graph evolution: Densification and shrinking diameters". *ACM transactions on Knowledge Discovery from Data*, 1(1): 2-es.

See Also

Other makes: make_cran, make_create, make_ego, make_explicit, make_learning, make_motifs, make_play, make_random, make_read, make_write

Examples

```
generate_smallworld(12, 0.025)
generate_smallworld(12, 0.25)
generate_scalefree(12, 0.25)
generate_scalefree(12, 1.25)
generate_fire(10)
generate_islands(10)
generate_citations(10)
```

---

make_stocnet *Multilevel, multiplex, multimodal, signed, dynamic or longitudinal changing networks*

Description

The 'stocnet' class of network object is a list of four main elements: nodes, ties, (nodal) changes, and info metadata about the network as a whole. This offers a consistent and flexible structure that enables more complex forms of networks to be contained in a single object. Unlike 'mnet' objects, 'stocnet' objects are not layered on top of 'igraph' or 'tbl_graph' objects, but instead are a list of tibbles and metadata. Unlike 'igraph' or 'tbl_graph' objects, 'stocnet' objects typically include more complex multimodal, longitudinal, or dynamic networks. They also typically include more metadata about the network, such as the names of the types of nodes and ties in the network. In other words, they are made not just for network analysis, but also network modelling.

Usage

```
make_stocnet(info = NULL, nodes = NULL, ties = NULL, changes = NULL)

## S3 method for class 'stocnet'
print(x, ..., n = 12)

validate_stocnet(.data)
```

Arguments

info	A list of metadata about the network as a whole. This can include the name of the network, as well as the names of the types of nodes and ties in the network. For example, the info component could include a 'name' element with the name of the network, a 'nodes' element with a character vector of the names of the types of nodes in the network, and a 'ties' element with a character vector of the names of the types of ties in the network. By default NULL.

nodes	A tibble of nodes in the network, with one row per node and one column for the node labels, which should be called 'name'. Additional columns can be included for node attributes, such as 'active' for changing networks and 'type' for multimodal networks. By default NULL.
ties	A tibble of ties in the network, with one row per tie and at least two columns for the node labels of the tie endpoints, which should be called 'from' and 'to', even if the network is not directed. Additional columns can be included for tie attributes, such as 'weight' for weighted networks and 'type' for multiplex networks. By default NULL.
changes	A tibble of nodal changes in the network, with one row per change and at least three columns for the node label of the change, which should be called 'node', the variable to which the change applies, which should be called 'var', and the new value to be applied, which should be called 'value'. Additional columns can be included for the time of the change, such as 'wave' or 'time'. By default NULL.
x	An object of class "mnet" or "tbl_graph".
...	Other arguments passed to or from other methods.
n	Number of observations to print across all network components, i.e. nodes, changes, and ties. By default 12.
.data	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package

Details

The package includes a validation function for stocnet objects, `validate_stocnet()`. This checks that the object has the correct structure and required components, and suggests improvements to the structure (e.g. correcting or adding reserved names) where possible. The required and reserved names for the components of a stocnet object are described below.

Info

There are several reserved names for the elements of the info component of a stocnet object.

- 'name' should be a single character string with the name of the network.
- 'modes' should be a character vector of the names of the modes of the nodes in a multimodal network.
- 'layers' should be a character vector of the names of the layers of the ties in a multiplex or multilayer network.
- 'directed' should be a logical indicating whether each layer is directed or undirected. If there are multiple layers, this can be a named logical vector with the directedness of each layer, where the names correspond to the layer names.

- 'dependent' should be a character string indicating which layer is the dependent layer in a multiplex network.
- 'doi' can be a character string with the DOI of the network, if it is from a published source.
- 'date' can be an integer of the year or the date the network represents.
- 'location' can be a character string with the location of the network.
- 'source' can be a character string indicating whether the network is observed or synthetic. If it is observed, the 'method' of data collection can be further specified. For example, the source could be 'observed', 'synthetic', 'survey', 'archival', 'digital trace', etc. If it is synthetic, then more details about how the network was generated can be included.

Many of these elements are drawn from the GRAND project's metadata standards for networks, which are designed to be consistent with the FAIR principles for data management.

In addition to these reserved names, the info component can include metadata relating to each layer of the network, such as the names of the types of nodes and ties in each layer, as well as the names of the dependent and independent layers in a multiplex network. These must be named as one of the layer names. There are some reserved names for these elements too:

- 'sender' should be a character string naming the type of node that sends ties in this layer.
- 'recipient' should be a character string naming the type of node that receives ties in this layer.
- 'update' should be a character string indicating whether layer changes are by "increment" or "replace".

Nodes

There are several reserved names for the columns of the nodes component of a stocnet object.

- 'label' should be a character vector of the labels of the nodes in the network.
- 'mode' should be a character vector of the modes of the nodes in a multimodal network.
- 'present' should be a logical vector indicating the initial active status of the node in changing networks. It can also be used to indicate missing nodes in a network, if the network is not changing but some nodes are missing, or the network is changing but some nodes never appear.

Changes

There are several required names for the columns of the changes component of a stocnet object (if one is included).

- 'time' can be an integer (e.g. for a wave) or date (e.g. POSIXct or mdate) vector of the times at which changes occur.
- 'node' must be an index (or names) of the node to which the change applies.
- 'var' must be a string vector naming the variable to which the change applies, such as 'active' for changing networks.
- 'value' must be the new value that should be applied at that change (or incremented, as appropriate). Note that the value column can be of any class, such as logical for changes to active status, or numeric for changes to a nodal attribute. These values are held internally as a list within the tibble, so that they can be of any class and length, but printed as a tibble with a 'value' column that shows the first value and a type label for the class of the value.

Ties

There are several required names for the columns of the ties component of a stocnet object (if one is included).

- 'from' must be an integer vector of the nodes sending each tie
- 'to' must be an integer vector of the nodes receiving each tie

There are also several reserved names for the columns of the ties component of a stocnet object.

- 'layer' should be a character vector of the layer of each tie in a multiplex or multilayer network
- 'weight' should be a numeric vector of the weights of the ties in a weighted network If the weight vector includes also negative values, then the network is a signed network, and the sign of the tie can be determined from the weight. Missing weights can be used to indicate missing ties in a network.
- 'time' should be a character or date vector of the time at which each tie is active in a longitudinal network.

Printing

When printed, 'stocnet' objects will print to the console any information stored about the names of the network, its modes, or layers. It will also describe key features of the network, such as whether the network is multiplex, weighted, directed, etc.

It will then print tibbles for the nodes, changes, and ties in the network, as appropriate. That is, if there is no nodal data (e.g. it is an unlabelled network without any other nodal attributes), then this will be skipped. Similarly, if no nodal changes are logged, this information will be skipped too.

`make_write` *Making networks to external files*

Description

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

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

Usage

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

write_edgelist(.data, filename, ...)

write_nodelist(.data, filename, ...)

write_pajek(.data, filename, ...)

write_ucinet(.data, filename, name)

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

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`filename`	Character string filename. If missing, the files will have the same name as the object and be saved to the working directory. An appropriate extension will be added if not included.
`...`	Additional parameters passed to the write function.
`name`	Character string to name the network internally, e.g. in UCINET. By default the name will be the same as the object.

Details

Note that these functions are not as actively maintained as others in the package, so please let us know if any are not currently working for you or if there are missing import routines by raising an issue on Github.

Value

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

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

Source

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

See Also

as

Other makes: make_cran, make_create, make_ego, make_explicit, make_learning, make_motifs, make_play, make_random, make_read, make_stochastic

---

manip_changes *Manipulating changes to nodes over time*

Description

These functions offer ways to modify data held about how nodes change over time. They include:

- add_changes() adds a table of changes to the nodes of a network.
- mutate_changes() can be used to update network changes.
- filter_changes() is used to subset network changes.
- gather_changes() is similar to filter_changes(), but collects the cumulative changes up to a time point.
- apply_changes() applies the changes collected up to a time point to a network, removing the changes.

An example of when this might be useful is to track change in the composition of a network (when nodes are present or absent over time), though the function can flexibly accommodate changes in other nodal attributes.

Usage

```
add_changes(.data, changes)

delete_changes(.data)

mutate_changes(.data, ...)

filter_changes(.data, ..., .by = NULL)

select_changes(.data, ..., .by = NULL)

gather_changes(.data, time)

apply_changes(.data, time)
```

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

changes	A data frame of changes. Ideally this will be in the form of "wave", "node", "var", and "value", but there are internal routines from some otherwise common formats. A data frame of composition change can be just two columns.
...	Additional parameters and arguments passed on internally.
.by	An attribute name to join objects by. By default, NULL.
time	A time point or wave at which to present the network.

Value

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

See Also

to_time()

Other manipulations: manip_info, manip_nodes_attr, manip_nodes_num, manip_ties_attr, manip_ties_num

Examples

```
add_changes(ison_algebra,
            data.frame(wave = 2, node = 1, var = "active", value = FALSE))
filter_changes(fict_starwars, node == "Anakin")
select_changes(fict_starwars, node)
gather_changes(fict_starwars, time = 3)
collect_changes(fict_starwars, time = 3)
```

---

manip_info *Manipulating network information*

Description

These functions allow users to add and edit information about the network itself. This includes the name, year, and mode of collection of the network, as well as definitions of the nodes and ties in the network. Where available, this information is printed for tidygraph-class objects, and can be used for printing a grand table in the {grand} package.

Usage

```
add_info(.data, ...)

mutate_info(.data, ...)

net_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
`...`	Named attributes. The following are currently recognised: "name", "year", and "doi" of the network, "collection" or "mode" of the network ("survey", "interview","sensor","observation","archival", or "simulation"), "nodes" (a vector of the names of the nodes) or "vertex1"/"vertex2", "ties" or "edge.pos"/"edge.neg" for defining the ties.

Value

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

See Also

Other manipulations: manip_changes, manip_nodes_attr, manip_nodes_num, manip_ties_attr, manip_ties_num

Examples

```
add_info(ison_algebra, name = "Algebra")
```

---

manip_nodes_attr *Manipulating node attributes*

Description

These functions allow users to add nodes attributes:

- add_node_attribute(), mutate(), or mutate_nodes() offer ways to add a vector of values to a network as a nodal attribute.
- rename_nodes() and rename() rename nodal attributes.
- bind_node_attributes() appends all nodal attributes from one network to another, and join_nodes() merges all nodal attributes from one network to another.

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

Usage

```
add_node_attribute(.data, attr_name, vector)

mutate(.data, ...)

mutate_nodes(.data, ...)

select(.data, ...)

select_nodes(.data, ...)

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

bind_node_attributes(.data, object2)

rename_nodes(.data, ...)

rename(.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
attr_name	Character string naming a nodal attribute. The attribute itself may be a logical mark, numeric measure, or character membership vector.
vector	A vector of values for the new attribute.
...	Additional parameters and arguments passed on internally.
object2	A second object to copy nodes or ties from.
.by	An attribute name to join objects by. By default, NULL.
join_type	A type of join to be used. Options are "full","left", "right", "inner".

Value

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

See Also

Other nodes: manip_nodes_num

Other manipulations: manip_changes, manip_info, manip_nodes_num, manip_ties_attr, manip_ties_num

Examples

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

---

manip_nodes_num *Manipulating number of nodes*

Description

These functions allow users to add and delete nodes:

- add_nodes() adds an additional number of nodes to network data.
- delete_nodes() deletes nodes from network data.
- filter_nodes() subsets nodes based on some nodal attribute-related logical statement.

While add_*()/delete_*() functions operate similarly as comparable {igraph} functions, filter_*(), works like a {tidyverse} or {dplyr}-style function.

Usage

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

delete_nodes(.data, nodes)

filter_nodes(.data, ..., .by = 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
nodes	The number of nodes to be added.
attribute	A named list to be added as tie or node attributes.
...	Additional parameters and arguments passed on internally.
.by	An attribute name to join objects by. By default, NULL.

Details

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

	igraph	network	tbl_graph
add_nodes	1	1	1
delete_nodes	1	1	1

Value

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

See Also

Other nodes: manip_nodes_attr

Other manipulations: manip_changes, manip_info, manip_nodes_attr, manip_ties_attr, manip_ties_num

Examples

```
other <- create_filled(4) |> mutate(name = c("A", "B", "C", "D"))
add_nodes(other, 4, list(name = c("Matthew", "Mark", "Luke", "Tim")))
```

---

manip_ties_attr *Manipulating tie attributes*

Description

These functions allow users to add and delete tie attributes:

- `add_tie_attribute()` and `mutate_ties()` offer ways to add a vector of values to a network as a tie attribute.
- `rename_ties()` renames tie attributes.
- `bind_ties()` appends the tie data from two networks and `join_ties()` merges ties from two networks, adding a tie attribute identifying the newly added ties.

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

Usage

```
add_tie_attribute(.data, attr_name, vector)

mutate_ties(.data, ...)

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

```
arrange_ties(.data, ...)

bind_ties(.data, ...)

join_ties(.data, object2, attr_name)

select_ties(.data, ...)

summarise_ties(.data, ...)
```

Arguments

.data	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
attr_name	Character string naming a nodal attribute. The attribute itself may be a logical mark, numeric measure, or character membership vector.
vector	A vector of values for the new attribute.
...	Additional parameters and arguments passed on internally.
object2	A second object to copy nodes or ties from.

Value

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

See Also

Other ties: manip_ties_num, modif_direction, modif_weight

Other manipulations: manip_changes, manip_info, manip_nodes_attr, manip_nodes_num, manip_ties_num

Examples

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

manip_ties_num *Manipulating number of ties*

Description

These functions allow users to add and delete ties:

- `add_ties()` adds additional ties to network data
- `delete_ties()` deletes ties from network data
- `filter_ties()` subsets ties based on some tie attribute-related logical statement.

While `add_*()`/`delete_*()` functions operate similarly as comparable `{igraph}` functions, `filter*()`, etc work like `{tidyverse}` or `{dplyr}`-style functions.

Usage

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

delete_ties(.data, ties)

filter_ties(.data, ...)
```

Arguments

`.data`	An object of a `{manynet}`-consistent class: • matrix (adjacency or incidence) from `{base}` R • edgelist, a data frame from `{base}` R or tibble from `{tibble}` • igraph, from the `{igraph}` package • network, from the `{network}` package • tbl_graph, from the `{tidygraph}` package
`ties`	The number of ties to be added or an even list of ties.
`attr_list`	A list of attributes to be added to the new ties.
`...`	Additional parameters and arguments passed on internally.

Details

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

```
            igraph network tbl_graph
add_ties         *       *         *
delete_ties      *       *         *
```

Value

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

See Also

Other ties: manip_ties_attr, modif_direction, modif_weight

Other manipulations: manip_changes, manip_info, manip_nodes_attr, manip_nodes_num, manip_ties_attr

Examples

```
  other <- create_filled(4) |> mutate(name = c("A", "B", "C", "D"))
  mutate_ties(other, form = 1:6) |> filter_ties(form < 4)
  add_tie_attribute(other, "weight", c(1, 2, 2, 2, 1, 2))
ison_adolescents |> add_ties(c("Betty","Tina"))
delete_ties(ison_adolescents, 3)
delete_ties(ison_adolescents, "Alice|Sue")
```

---

mark_features *Marking networks features*

Description

These functions implement logical tests for various network features.

- `is_connected()` tests whether network is strongly connected, or weakly connected if undirected.
- `is_perfect_matching()` tests whether there is a matching for a network that covers every node in the network.
- `is_eulerian()` tests whether there is a Eulerian path for a network where that path passes through every tie exactly once.
- `is_acyclic()` tests whether network is a directed acyclic graph.
- `is_aperiodic()` tests whether network is aperiodic.

Usage

```
is_connected(.data)

is_perfect_matching(.data, mark = "type")

is_eulerian(.data)

is_acyclic(.data)

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

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`mark`	A logical vector marking two types or modes. By default "type".
`max_path_length`	Maximum path length considered. If negative, paths of all lengths are considered. By default 4, to avoid potentially very long computation times.

Value

TRUE if the condition is met, or FALSE otherwise.

is_connected

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

is_perfect_matching

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

Source

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

References

On perfect matching:

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

On aperiodicity:

Jarvis, J.P, and D.R. Shier. 1996. "Graph-theoretic analysis of finite Markov chains", in Shier, D.R., Wallenius, K.T. (eds) *Applied Mathematical Modeling: A Multidisciplinary Approach*. CRC Press.

See Also

Other marking: `mark_is`

Examples

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

`mark_format_change` *Marking networks change formats*

Description

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

- `is_longitudinal()` marks networks TRUE if they contain multiple waves of data.
- `is_dynamic()` marks networks TRUE if they contain time-stamped data.
- `is_changing()` marks networks TRUE if they contain any nodal changes.

Usage

```
is_longitudinal(.data)

is_dynamic(.data)

is_changing(.data)
```

Arguments

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

See Also

Other marks: `mark_format_node`, `mark_format_tie`

Examples

```
is_longitudinal(create_tree(5, 3))
is_dynamic(create_tree(3))
is_changing(fict_starwars)
```

`mark_format_node` *Marking networks nodal formats*

Description

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

- `is_twomode()` marks networks TRUE if they contain two sets of nodes.
- `is_labelled()` marks networks TRUE if there is a 'names' attribute for the nodes.
- `is_attributed()` marks networks TRUE if there are other nodal attributes than 'names' or 'type'.
- `is_egonet()` marks networks TRUE if it is a list of networks where each network contains only one node and its ties.

Usage

```
is_twomode(.data)

is_labelled(.data)

is_attributed(.data)

is_egonet(.data)
```

Arguments

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

See Also

Other marks: `mark_format_change`, `mark_format_tie`

Examples

```
is_twomode(create_filled(c(2,2)))
is_labelled(create_empty(3))
is_attributed(ison_algebra)
is_egonet(fict_starwars)
```

`mark_format_tie` *Marking networks tie formats*

Description

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

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

Usage

```
is_weighted(.data)

is_directed(.data)

is_signed(.data)

is_complex(.data)

is_multiplex(.data)

is_uniplex(.data)
```

Arguments

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

See Also

Other marks: mark_format_change, mark_format_node

Examples

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

---

mark_is *Marking networks classes*

Description

These functions implement logical tests for networks' classes.

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

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

Usage

```
is_manynet(.data)

is_graph(.data)

is_edgelist(.data)

is_list(.data)
```

Arguments

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

Value

TRUE if the condition is met, or FALSE otherwise.

See Also

Other marking: mark_features

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)))
```

---

measure_attributes_nodes

Describing attributes of nodes in a network

Description

These functions extract certain attributes from network data:

- node_attribute() returns an attribute's values for the nodes in a network.
- node_names() returns the names of the nodes in a network.
- node_is_mode() returns the mode of the nodes in a network.

These functions are also often used as helpers within other functions. node_*() always return vectors the same length as the number of nodes in the network.

Usage

```
node_attribute(.data, attr_name)

node_names(.data)

node_is_mode(.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
`attr_name`	Character string naming a nodal attribute. The attribute itself may be a logical mark, numeric measure, or character membership vector.

See Also

Other measures: measure_attributes_ties, measure_dims

Examples

```
node_attribute(fict_lotr, "Race")
node_names(ison_southern_women)
node_is_mode(ison_southern_women)
```

---

measure_attributes_ties

Describing attributes of ties in a network

Description

These functions extract certain attributes from network data:

- `tie_attribute()` returns an attribute's values for the ties in a network.
- `tie_weights()` returns the weights of the ties in a network.
- `tie_signs()` returns the signs of the ties in a network.
- `tie_is_twomode()` returns whether each tie in a network is a cross-mode tie.

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

Usage

```
tie_attribute(.data, attr_name)

tie_weights(.data)

tie_signs(.data)

tie_is_twomode(.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
attr_name	Character string naming a nodal attribute. The attribute itself may be a logical mark, numeric measure, or character membership vector.

See Also

Other measures: measure_attributes_nodes, measure_dims

Examples

```
tie_attribute(ison_algebra, "task_tie")
tie_weights(to_mode1(ison_southern_women))
tie_signs(to_uniplex(fict_marvel,"relationship"))
tie_is_twomode(fict_actually)
```

---

measure_dims *Describing network dimensions*

Description

These functions extract certain attributes from given network data:

- net_nodes() returns the total number of nodes (of any mode) in a network.
- net_ties() returns the number of ties in a network.
- net_dims() returns the dimensions of a network in a vector as long as the number of modes in the network.

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

Usage

```
net_nodes(.data)

net_modes(.data)

net_ties(.data)

net_layers(.data)

net_dims(.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

`net_*()` functions always relate to the overall graph or network, usually returning a scalar. `net_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.

See Also

Other measures: `measure_attributes_nodes`, `measure_attributes_ties`

Examples

```
net_nodes(ison_southern_women)
net_modes(ison_southern_women)
net_ties(ison_southern_women)
net_layers(ison_southern_women)
net_dims(ison_southern_women)
net_dims(to_mode1(ison_southern_women))
```

`member_names` *Describing network names*

Description

These functions extract certain attributes from given network data:

- `net_name()` returns the name of the network, if it has one.
- `mode_names()` returns a vector of the names of the modes in a network, if they have been defined.
- `net_node_attributes()` returns a vector of nodal attributes in a network.
- `layer_names()` returns a vector of the names of the layers in a network, if they have been defined.
- `net_tie_attributes()` returns a vector of tie attributes in a network.

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

Usage

```
net_name(.data, prefix = NULL)

mode_names(.data)

net_node_attributes(.data)

layer_names(.data)

net_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
prefix	An optional string to be added before the name of the network.

Value

net_*() functions always relate to the overall graph or network, usually returning a scalar. net_*_attributes() returns a string vector with the names of all node or tie attributes in the network.

Examples

```
net_name(ison_southern_women)
  mode_names(ison_algebra)
  net_node_attributes(fict_lotr)
  layer_names(ison_algebra)
  net_tie_attributes(ison_algebra)
```

---

modif_correlation *Node correlation*

Description

This function performs a Pearson pairwise correlation on a given matrix or network data. It includes a switch: whereas for a two-mode network it will perform a regular correlation, including all rows, for an undirected network it will perform a correlation on a matrix with the diagonals removed, for a reciprocated network it will include the difference between reciprocated ties, and for complex networks it will include also the difference between the self ties in each pairwise calculation. This function runs in $O(mn^2)$ complexity.

Usage

```
to_correlation(.data, method = NULL)

to_cosine(.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
method	One of the following: "all" includes all information, "diag" excludes the diagonal (self-ties), "recip" excludes the diagonal but compares pairs' reciprocal ties, and "complex" compares pairs' reciprocal ties and their self ties. By default the appropriate method is chosen based on the network format.

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_split, modif_weight

---

modif_direction *Modifying networks by formatting their directionality*

Description

These functions reformat manynet-consistent data.

- to_directed() reformats undirected network data to a directed network.
- to_undirected() reformats directed network data to an undirected network, so that any pair of nodes with at least one directed edge will be connected by an undirected edge in the new network. This is equivalent to the "collapse" mode in {igraph}..
- to_redirected() formats directed network data by flipping/transposing any existing direction such that senders become receivers and receivers become senders. This essentially has no effect on undirected networks or reciprocated ties.
- to_reciprocated() reformats directed network data such that every directed tie is reciprocated.
- to_acyclic() reformats network data to an acyclic graph.

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

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

Usage

```
to_directed(.data)

to_undirected(.data)

to_redirected(.data)

to_reciprocated(.data)

to_acyclic(.data)
```

Arguments

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

Details

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

	data.frame	igraph	matrix	network	tbl_graph
to_acyclic	*	*	*	*	*
to_directed	*	*	*	*	*
to_reciprocated	*	*	*	*	*
to_redirected	*	*	*	*	*
to_undirected	*	*	*	*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other ties: `manip_ties_attr`, `manip_ties_num`, `modif_weight`

Other modifications: `modif_correlation`, `modif_from`, `modif_labels`, `modif_levels`, `modif_miss`, `modif_paths`, `modif_permutation`, `modif_plexity`, `modif_project`, `modif_scope`, `modif_split`, `modif_weight`

modif_from *Joining lists of networks, graphs, and matrices*

Description

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

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

Usage

```
from_subgraphs(netlist)

from_egos(netlist)

from_waves(netlist)

from_slices(netlist, remove.duplicates = FALSE)

from_ties(netlist, netnames)
```

Arguments

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

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: `modif_correlation`, `modif_direction`, `modif_labels`, `modif_levels`, `modif_miss`, `modif_paths`, `modif_permutation`, `modif_plexity`, `modif_project`, `modif_scope`, `modif_split`, `modif_weight`

Examples

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

---

modif_labels *Modifying node labels*

Description

These functions add some format to manynet-consistent data.

- `to_named()` reformats unlabelled network data to labelled network data from a vector of names or random baby names.
- `to_unnamed()` reformats labelled network data to unlabelled network data.

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

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

Usage

```
to_named(.data, names = NULL)

to_unnamed(.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

names Character vector of the node names. NULL by default.

Details

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

	data.frame	igraph	matrix	network	tbl_graph
to_named	*	*	*	*	*
to_unnamed	*	*	*	*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_split, modif_weight

---

modif_levels *Modifying network levels*

Description

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

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

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

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

Usage

```
to_onemode(.data)

to_twomode(.data, mark)

to_multilevel(.data)
```

Arguments

.data	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
mark	A logical vector marking two types or modes. By default "type".

Details

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

	igraph	matrix	network	tbl_graph
to_multilevel	*	*		*
to_onemode	*	*		*
to_twomode	*		*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_split, modif_weight

---

modif_miss *Modifying missing tie data*

Description

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

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

Usage

```
na_to_zero(.data)

na_to_mean(.data)
```

Arguments

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

Value

A tidygraph object modified as explained in the function description, details, or section.

References

On missing data:

Krause, Robert, Mark Huisman, Christian Steglich, and Tom A.B. Snijders. 2020. "Missing data in cross-sectional networks: An extensive comparison of missing data treatment methods". *Social Networks*, 62: 99-112. doi:10.1016/j.socnet.2020.02.004

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_split, modif_weight

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)
```

`modif_paths` *Modifying networks paths*

Description

These functions return tidygraphs containing only special sets of ties:

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

Usage

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

to_mentoring(.data, elites = 0.1)

to_eulerian(.data)

to_tree(.data)

to_dominating(.data, from, direction = c("out", "in"))
```

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`mark`	A logical vector marking two types or modes. By default "type".
`capacities`	An integer or vector of integers the same length as the nodes in the network that describes the maximum possible degree the node can have in the matched network.
`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).

from	The index or name of the node from which the path should be traced.
direction	Character string, “out” bases the measure on outgoing ties, “in” on incoming ties, and "all" on either/the sum of the two. By default "all".

Details

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

	data.frame	igraph	matrix	network	tbl_graph
to_eulerian		*			*
to_matching	*	*	*	*	*
to_mentoring		*			*

Value

A tidygraph object modified as explained in the function description, details, or section.

`to_matching()`

This function attempts to solve the stable matching problem, also known as the stable marriage problem, upon a given two-mode network (or other network with a binary mark).

In the basic version, `to_matching()` uses `igraph::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.

In the more general version, each node may have a larger capacity, or even different capacities. Here an implementation of the Gale-Shapley algorithm is used, in which an iterative process of proposal and acceptance is repeated until all are matched or have exhausted their lists of preferences. This is, however, computationally slower.

References

On matching:

Gale, David, and Lloyd Stowell Shapley. 1962. "College admissions and the stability of marriage". *The American Mathematical Monthly*, 69(1): 9–14. doi:10.2307/2312726

Goldberg, Andrew V., and Robert E. Tarjan. 1986. "A new approach to the maximum flow problem". *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*. 136-146. doi:10.1145/12130.12144

On mentoring:

Valente, Thomas, and Rebecca Davis. 1999. "Accelerating the Diffusion of Innovations Using Opinion Leaders", *Annals of the American Academy of Political and Social Science* 566: 56-67. doi:10.1177/000271629956600105

On Eulerian trails:

Euler, Leonard. 1736. "Solutio problematis ad geometriam situs pertinentis". *Comment. Academiae Sci. I. Petropolitanae* 8: 128–140.

Hierholzer, Carl. 1873. "Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren". *Mathematische Annalen*, 6(1): 30–32. doi:10.1007/BF01442866

On minimum spanning trees:

Boruvka, Otakar. 1926. "O jistem problemu minimalnim". *Prace Mor. Prirodoved. Spol. V Brne III* 3: 37-58.

Kruskal, Joseph B. 1956. "On the shortest spanning subtree of a graph and the travelling salesman problem". *Proceedings of the American Mathematical Society* 7(1): 48-50. doi:10.1090/S0002-9939195600786867

Prim, R.C. 1957. "Shortest connection networks and some generalizations". *Bell System Technical Journal* 36(6):1389-1401. doi:10.1002/j.15387305.1957.tb01515.x

See Also

Other modifications: `modif_correlation`, `modif_direction`, `modif_from`, `modif_labels`, `modif_levels`, `modif_miss`, `modif_permutation`, `modif_plexity`, `modif_project`, `modif_scope`, `modif_split`, `modif_weight`

Examples

```
to_matching(ison_southern_women)
  to_eulerian(delete_nodes(ison_koenigsberg, "Lomse"))
```

---

modif_permutation *Network permutation*

Description

`to_permuted()` permutes the network using a Fisher-Yates shuffle on both the rows and columns (for a one-mode network) or on each of the rows and columns (for a two-mode network).

Usage

```
to_permuted(.data, with_attr = TRUE)
```

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
`with_attr`	Logical whether any attributes of the object should be retained. By default TRUE.

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: `modif_correlation`, `modif_direction`, `modif_from`, `modif_labels`, `modif_levels`, `modif_miss`, `modif_paths`, `modif_plexity`, `modif_project`, `modif_scope`, `modif_split`, `modif_weight`

`modif_plexity` *Modifying network complexity*

Description

These functions reformat manynet-consistent data.

- `to_anti()` reformats network data into its complement, where only ties *not* present in the original network are included in the new network.
- `to_simplex()` reformats complex network data, containing loops, to simplex network data, without any loops.
- `to_uniplex()` reformats multiplex network data to a single type of tie.

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

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

Usage

```
to_anti(.data)

to_simplex(.data)

to_uniplex(.data, tie)
```

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`tie`	Character string naming a tie attribute to retain from a graph.

Details

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

	data.frame	igraph	matrix	network	tbl_graph
to_anti	*	*	*	*	*
to_simplex	*	*	*	*	*
to_uniplex	*	*	*	*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_project, modif_scope, modif_split, modif_weight

Examples

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

---

modif_project *Modifying networks projection*

Description

These functions offer tools for projecting manynet-consistent data:

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

Usage

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

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

to_ties(.data)
```

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`similarity`	Method for establishing ties, currently "count" (default), "jaccard", or "rand". • "count" calculates the number of coinciding ties, and can be interpreted as indicating the degree of opportunities between nodes. • "jaccard" uses this count as the numerator in a proportion, where the denominator consists of any cell where either node has a tie. It can be interpreted as opportunity weighted by participation. • "rand", or the Simple Matching Coefficient, is a proportion where the numerator consists of the count of cells where both nodes are present or both are absent, over all possible cells. It can be interpreted as the (weighted) degree of behavioral mirroring between two nodes. • "pearson" (Pearson's coefficient) and "yule" (Yule's Q) produce correlations for valued and binary data, respectively. Note that Yule's Q has a straightforward interpretation related to the odds ratio.

Details

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

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

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_scope, modif_split, modif_weight

Examples

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

`modif_scope` *Modifying networks scope*

Description

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

- `to_ego()` scopes a network into the local neighbourhood of a given node.
- `to_giant()` scopes a network into one including only the main component and no smaller components or isolates.
- `to_no_isolates()` scopes a network into one excluding all nodes without ties.
- `to_no_missing()` scopes a network to one retaining only complete cases, i.e. nodes with no missing values.
- `to_subgraph()` scopes a network into a subgraph by filtering on some node-related logical statement.
- `to_blocks()` reduces a network to ties between a given partition membership vector.

Usage

```
to_no_missing(.data)

to_ego(.data, node, max_dist = 1, min_dist = 0, direction = c("out", "in"))

to_time(.data, time)

to_giant(.data)

to_no_isolates(.data)

to_subgraph(.data, ...)

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

Arguments

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

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.
direction	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. By default "all".
time	A time point or wave at which to present the network.
...	Arguments passed on to dplyr::filter
membership	A vector of partition memberships.
FUN	A function for summarising block content. By default mean. Other recommended options include median, sum, min or max.

Details

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

	data.frame	igraph	list	matrix	network	tbl_graph
to_blocks	*	*		*	*	*
to_ego		*				*
to_egos	*	*		*	*	*
to_giant	*	*		*	*	*
to_no_isolates	*	*	*	*	*	*
to_no_missing						*
to_subgraph	*	*		*	*	*
to_subgraphs		*			*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

`to_blocks()`

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

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_split, modif_weight

Examples

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

`modif_split` *Splitting networks into lists*

Description

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

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

Usage

```
to_egos(.data, max_dist = 1, min_dist = 0, direction = c("out", "in"))

to_subgraphs(.data, attribute)

to_components(.data)

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

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

Arguments

`.data`	An object of a {manynet}-consistent class: • matrix (adjacency or incidence) from {base} R • edgelist, a data frame from {base} R or tibble from {tibble} • igraph, from the {igraph} package • network, from the {network} package • tbl_graph, from the {tidygraph} package
`max_dist`	The maximum breadth of the neighbourhood. By default 1.
`min_dist`	The minimum breadth of the neighbourhood. By default 0. Increasing this to 1 excludes the ego, and 2 excludes ego's direct alters.
`direction`	Character string, "out" bases the measure on outgoing ties, "in" on incoming ties, and "all" on either/the sum of the two. By default "all".
`attribute`	One or two attributes used to slice data.
`panels`	Would you like to select certain waves? NULL by default. That is, a list of networks for every available wave is returned. Users can also list specific waves they want to select.

cumulative	Whether to make wave ties cumulative. FALSE by default. That is, each wave is treated isolated.
slice	Character string or character list indicating the date(s) or integer(s) range used to slice data (e.g slice = c(1:2, 3:4)).

Details

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

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

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_weight

Examples

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

`modif_weight` *Modifying tie weight formats*

Description

These functions reformat tie attributes like their weight or sign:

- `to_unweighted()` reformats weighted network data to unweighted network data, with all tie weights removed.
- `to_unsigned()` reformats signed network data to unsigned network data keeping just the "positive" or "negative" ties.

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

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

Usage

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

to_unweighted(.data, threshold = 1)

to_signed(.data, mark = NULL)

to_weighted(.data, measure = 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
`keep`	In the case of a signed network, whether to retain the "positive" or "negative" ties.
`threshold`	For a matrix, the threshold to binarise/dichotomise at.
`mark`	A mark (logical vector) the length of the ties in the network.
`measure`	A numeric vector (measure) that will be added as the tie weights to the network. If this is NULL, then the tie weights will be drawn from a Poisson distribution with $\lambda = 4$.

Details

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

	data.frame	igraph	matrix	network	tbl_graph
to_signed		*	*	*	*
to_unsigned		*	*	*	*
to_unweighted		*	*	*	*
to_weighted			*	*	*

Value

A tidygraph object modified as explained in the function description, details, or section.

See Also

Other ties: manip_ties_attr, manip_ties_num, modif_direction

Other modifications: modif_correlation, modif_direction, modif_from, modif_labels, modif_levels, modif_miss, modif_paths, modif_permutation, modif_plexity, modif_project, modif_scope, modif_split

---

progress *Console command line interface*

Description

These functions wrap {cli} functions and elements to build an attractive command line interface (CLI).

- snet_progress_step() for progress steps.
- snet_progress_along() for progress along a vector.
- snet_progress_seq() for progress along a sequence.
- snet_progress_nodes() for progress along the nodes of a network.

If you wish to receive fewer messages in the console, run options(snet_verbosity = 'quiet').

Usage

```
snet_progress_step(..., .envir = parent.frame())

snet_progress_along(..., .envir = parent.frame())

snet_progress_seq(..., .envir = parent.frame())

snet_progress_nodes(..., .envir = parent.frame())

seq_nodes(.data)

seq_ties(.data)
```

Arguments

`...`	One or more character strings. For most of these functions, if multiple strings are passed these will be pasted together.
`.envir`	This argument is just to inherit the parent frame in the (likely) event that the function is used within another 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

Index