Package 'ideanet'

July 3, 2025

Type Package

Title Integrating Data Exchange and Analysis for Networks ('ideanet')

Version 1.1.1

Date 2025-06-02

Description A suite of convenient tools for social network analysis geared toward students, entry-level users, and non-expert practitioners. 'ideanet' features unique functions for the processing and measurement of sociocentric and egocentric network data. These functions automatically generate node- and system-level measures commonly used in the analysis of these types of networks. Outputs from these functions maximize the ability of novice users to employ network measurements in further analyses while making all users less prone to common data analytic errors. Additionally, 'ideanet' features an R Shiny graphic user interface that allows novices to explore network data with minimal need for coding.

Maintainer Tom Wolff <tom.wolff@northwestern.edu>

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports CliquePercolation, cluster, colorspace, concorR, cowplot, data.table, dplyr, forcats, ggplot2, ggthemes, grDevices, gridGraphics, igraph (>= 2.1.0), intergraph, jsonlite, magrittr, Matrix, methods, moments, network, readxl, reshape2, rlang, RSpectra, shiny, sna, stringr, tibble, tidyr, tidyselect

Suggests DT, devtools, egor, ergm, shinythemes, shinyWidgets, knitr, rmarkdown, shinycssloaders, visNetwork

RoxygenNote 7.3.2

Depends R (>= 3.5.0), igraphdata

VignetteBuilder knitr

NeedsCompilation no

Author Tom Wolff [aut, cre] (ORCID: <https://orcid.org/0000-0002-4884-251X>), Jonathan Howard Morgan [aut] (ORCID: <https://orcid.org/0000-0001-5181-9903>),

Gabriel Varela [aut] (ORCID: <https://orcid.org/0000-0003-2800-1577>),
Kieran Lele [aut],
Ethan Bhojani [aut],
Emily Heraty [aut],
Dana Pasquale [aut] (ORCID: <https://orcid.org/0000-0001-6686-7844>),
Peter Mucha [aut] (ORCID: <https://orcid.org/0000-0002-0648-7230>),
James Moody [aut] (ORCID: <https://orcid.org/0000-0002-3311-4173>)

Repository CRAN

Date/Publication 2025-07-03 16:50:02 UTC

Contents

CHAMP *Find the Convex Hull of Admissible Modularity Partitions (*CHAMP*)*

Description

The Convex Hull of Admissible Modularity Partitions (CHAMP) method post-processes an input set of partitions as collected by get_partitions (or as formatted similarly from some other source of selected partitions) to identify the partitions that are somewhere optimal in the resolution parameter and the associated domains of (generalized) modularity optimization. That is, given the input set of partitions of nodes in a network into communities, CHAMP identifies which input partition is optimal at each value of the resolution parameter, gamma. Importantly, CHAMP is deterministic and polynomial in time given a specified input set of partitions; that is, all of the computational complexity and pseudo-stochastic heuristic nature of community detection is in identifying a good input set in *get_partitions*.

The CHAMP method was developed and studied in Weir, William H., Scott Emmons, Ryan Gibson, Dane Taylor, and Peter J. Mucha. "Post-Processing Partitions to Identify Domains of Modularity Optimization." Algorithms 10, no. 3 (August 19, 2017): 93. doi:10.3390/a10030093.

See also https://github.com/wweir827/CHAMP and https://github.com/ragibson/ModularityPruning.

Usage

```
CHAMP(network, partitions, plottitle = NULL)
```

Arguments

network	The network, as igraph object, to be clustered into communities. Only undirected networks are currently supported. If the object has a 'weight' edge attribute, then that attribute will be used.
partitions	A list of unique partitions (in the format generated by get_partitions).
plottitle	Optional title for generated plot of (generalized) modularity versus resolution parameter.

Value

CHAMP returns the input list of partitions with a $CHAMPsummary about which partitions are somewhere optimal (in the sense of modularity Q with a resolution parameter gamma) and their domains of optimality, along with the generated $CHAMPfigure plot of the upper envelope of Q(gamma). The returned list object also contains the original list entered into the partitions argument.

Author(s)

Peter J. Mucha (<peter.j.mucha@dartmouth.edu>), Alex Craig, Rachel Matthew, Sydney Rosenbaum and Ava Scharfstein

Examples

```
# Use get_partitions and CHAMP to generate multiple partitions of the
# Zachary karate club and identify the domains of optimality in the
# resolution parameter for different partitions
data(karate, package = "igraphdata")
partitions <- get_partitions(karate, n_runs = 500)
partitions <- CHAMP(karate, partitions, plottitle = "Weighted Karate Club")
```

---

comm_detect *Community Detection Across Multiple Routines (*comm_detect*)*

Description

The comm_detect function runs a set of several commonly-used community detection routines on a network and provides community assignments from these routines. Need to mention that only supports undirected nets and that for some routines the median community value is used.

Usage

```
comm_detect(g, modres = 1, slow_routines = FALSE, shiny = FALSE)
```

Arguments

g	An igraph object. If the igraph object contains a directed network, the function will treat the network as undirected before running community detection routines.
modres	A modularity resolution parameter used when performing community detection using the Leiden method.
slow_routines	A logical indicating whether time-intensive community detection routines should be performed on larger networks. Edge betweenness, leading eigenvector, link communities, and stochastic blockmodeling each take a very long time to identify communities in networks consisting of more than a few thousand nodes. By default, comm_detect will skip performing these routines on networks with more than 5,000 nodes and inform the user that it is doing so.
shiny	An argument indicating whether the output from the comm_detect function will be fed into the IDEANet visualization app.

Value

comm_detect returns a list contianing three data frames. comm_members indicates each node's assigned community membership from each community detection routine. comm_summaries indicates the number of communities inferred from each routine as well as the modularity score arising from community assignments. comp_scores contains a matrix indicating the similarity of community assignments between each pair of community detection routines, measured using adjusted rand scores. A fourth element in the list, plots, contains a series of network visualizations in which nodes are colored by their assigned community memberships from each routine. If shiny == FALSE, this function will display these visualizations in the user's plot window.

Examples

```
# Run netwrite
nw_fauxmesa <- netwrite(nodelist = fauxmesa_nodes,
                        node_id = "id",
                        i_elements = fauxmesa_edges$from,
                        j_elements = fauxmesa_edges$to,
                        directed = TRUE,
                        net_name = "faux_mesa",
                        output = "graph")

# Run comm_detect
faux_communities <- comm_detect(g = nw_fauxmesa$faux_mesa)
```

---

ego_homophily *Measuring Homophily in Ego Networks (*ego_homophily*)*

Description

The ego_homophily function identifies how similar ego is from their alters on a given attribute.

Usage

```
ego_homophily(
  ego_id,
  ego_measure,
  alter_ego,
  alter_measure,
  prefix = NULL,
  suffix = NULL,
  prop = FALSE
)
```

Arguments

ego_id	A vector of unique ego identifiers located in an ego dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled egos.
ego_measure	A vector of attributes corresponding to each ego.
alter_ego	A vector of ego identifiers located in an alter dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled alters.
alter_measure	A vector of attributes corresponding to each alter
prefix	A character value indicating the desired prefix for the calculated homophily measure.
suffix	A character value indicating the desired suffix for the calculated homophily measure.
prop	A logical value indicating whether homophily should be represented as a count or as a proportion.

Value

ego_homophily returns a dataframe of vectors that include the ego identifier and the number or proportion of alters with the same selected attribute

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


# Homophily as a Count
race_homophily_count <- ego_homophily(ego_id = ngq_nw$egos$ego_id,
                                      ego_measure = ngq_nw$egos$race,
                                      alter_ego = ngq_nw$alters$ego_id,
                                      alter_measure = ngq_nw$alters$race,
                                      suffix = "race")

race_homophily_count

# Homophily as a Proportion
race_homophily_prop <- ego_homophily(ego_id = ngq_nw$egos$ego_id,
                                     ego_measure = ngq_nw$egos$race,
                                     alter_ego = ngq_nw$alters$ego_id,
                                     alter_measure = ngq_nw$alters$race,
                                     prop = TRUE,
                                     suffix = "race")
race_homophily_prop
```

---

ego_netwrite *Ego Network Cleaning and Measure Calculation (*ego_netwrite*)*

Description

The ego_netwrite function reads in data pertaining to ego networks and processes them into a set of standardized outputs, including measures commonly calculated for ego networks.

Usage

```
ego_netwrite(
  egos,
  ego_id,
  alters = NULL,
  alter_id = NULL,
  alter_ego = NULL,
  alter_types = NULL,
  max_alters = Inf,
  alter_alter = NULL,
  aa_ego = NULL,
  i_elements = NULL,
  j_elements = NULL,
  directed = FALSE,
  aa_type = NULL,
  missing_code = 99999,
  na.rm = FALSE,
  egor = FALSE,
  egor_design = NULL
)
```

Arguments

egos	A data frame containing measures of ego attributes.
ego_id	A vector of unique identifiers corresponding to each ego, or a single character value indicating the name of the column in egos containing ego identifiers.
alters	A data frame containing measures of alter attributes.
alter_id	A vector of identifiers indicating which alter is associated with a given row in alters, or a single character value indicating the name of the column in alters containing alter identifiers.
alter_ego	A vector of identifiers indicating which ego is associated with a given alter, or a single character value indicating the name of the column in alters containing ego identifiers.
alter_types	A character vector indicating the columns in alters that indicate whether a given alter has certain types of relations with ego. These columns should all contain binary measures indicating whether alter has a particular type of relation with ego.
max_alters	A numeric value indicating the maximum number of alters an ego in the dataset could have nominated
alter_alter	A data frame containing an edgelist indicating ties between alters in each ego's network. This edgelist is optional, but ego_netwrite will not provide certain measures without it.
aa_ego	A vector of identifiers indicating which ego is associated with a given tie between alters, or a single character indicating the name of the column in alter_alter containing ego identifiers.

i_elements	A vector of identifiers indicating which alter is on one end of an alter-alter tie, or a single character indicating the name of the column in alter_alter containing these identifiers.
j_elements	A vector of identifiers indicating which alter is on the other end of an alter-alter tie, or a single character indicating the name of the column in alter_alter containing these identifiers.
directed	A logical value indicating whether network ties are directed or undirected.
aa_type	A numeric or character vector indicating the types of relationships represented in the alter edgelist, or a single character value indicating the name of the column in alter_alter containing relationship type. If alter_type is specified, ego_netwrite will treat the data as a set of multi-relational networks and produce additional outputs reflecting the different types of ties occurring in each ego network.
missing_code	A numeric value indicating "missing" values in the alter-alter edgelist.
na.rm	A logical value indicating whether NA values should be excluded when calculating continuous measures.
egor	A logical value indicating whether output should include an egor object, which is often useful for visualizaton and for simulation larger networks from egocentric data.
egor_design	If creating an egor object, a list of arguments to srvyr::as_survey_design specifying the sampling design for egos. This argument corresponds to ego_design in egor::egor.

Value

ego_netwrite returns a list containing several output objects. Users may find it easier to access and work with outputs by applying list2env to this list, which will separate outputs and store them in the R Global Environment. Note, however, that this risks overwriting existing objects in the Global Environment should those objects share names with objects in netwrite's output. Outputs include a data frame containing measures of ego attributes, another data frame containing measures of alter attributes and network position, a third containing the alter-alter edgelist (when applicable), a fourth containing summary measures for each individual ego network, and a fifth providing summary measures for the overall dataset. Additionally, ego_netwrite returns a list of igraph objects constructed for each individual ego network, as well as an egor object for the overall dataset if desired.

Examples

```
# Simple Processing, Ignoring Ego-Alter or Alter-Alter Relation Types
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
```

```
                        aa_ego = ngq_aa$ego_id,
                        i_elements = ngq_aa$alter1,
                        j_elements = ngq_aa$alter2,
                        directed = FALSE)


# View summaries of individual ego networks
head(ngq_nw$summaries)

# View summary of overall dataset
head(ngq_nw$overall_summary)

# View sociogram of fourth ego network
plot(ngq_nw$igraph_objects[[4]]$igraph_ego)


# For advanced applications involving multiple relationship types
# and `egor` object creation, please consult the `ego_netwrite` vignette
vignette("ego_netwrite", package = "ideanet")
```

---

ego_reshape *Reshaping Egocentric Data (*ego_reshape*)*

Description

The ego_reshape function reshapes egocentric network data stored in a single wide dataset into three dataframes optimized for use with ego_netwrite.

Usage

```
ego_reshape(
  data,
  ego_id,
  ego_vars,
  alters,
  alter_vars,
  alter_alter,
  aa_vars = NULL,
  directed = NULL,
  loops = NULL,
  missing_code = 99999,
  output_name = "ego_long"
)
```

Arguments

data	A data frame containing egocentric network data in a wide format.

ego_id — A character value indicating the name of the column in data containing ego identifiers, or a numeric value indicating the position of the column containing ego identifiers.

ego_vars — A character vector indicating the names of the columns in data containing ego-level measures, or a numeric vector indicating the positions of the columns containing ego-level measures.

alters — A character vector indicating the names of the columns in data containing ego-alter ties, or a numeric vector indicating the positions of the columns containing ego-alter ties.

alter_vars — A character vector indicating the names of the columns in data containing alter-level measures, or a numeric vector indicating the positions of the columns containing alter-level measures. Variables are assumed to be ordered such that each consecutive set of columns represent a single alter-level variable, and that columns within this set are ordered such that the first column represents the value associated with the alter represented by the first column specified in alters, the second column in the set is associated with the alter represented by the second column in alters, and so on. If a certain variable was collected for only the first n alters in a survey instrument, we recommend creating placeholder columns in which all values are coded NA.

alter_alter — A character vector indicating the names of the columns in data that indicate the presence of alter-alter ties, or a numeric vector indicating the positions of the columns indicating alter-alter ties. These columns should be ordered such that their values appear as they would were one to move left-to-right, top-to-bottom in an adjacency matrix. For example, the value of column 1 should usually indicate whether a tie exists between alter 1 and alter 2, the value of column 2 should indicate the presence of a tie between alter 1 and alter 3, the value of column 3 should indicate the presence of a tie between alter 1 and alter 4, and so on. The number of columns needed to represent the full set of possible ties between alters may vary depending on a) whether ties in the network are directed or undirected, and b) whether it is possible for alters to be tied to themselves. If users do not specify these conditions using the directed and loops arguments, respectively, This function uses the number of columns specified in alters to detect the presenece of directed ties and self-loops in the network.

aa_vars — A character vector indicating the names of the columns in data representing edge-level characteristics of alter-alter ties. Columns should be ordered in a similar fashion as with alter_vars where consecutive sets of n columns represent a single variable and columns within these sets are ordered in the same way as their corresponding edge indicator columns are ordered in alter_alter.

directed — A logical value indicating whether alter-alter ties are directed or undirected.

loops — A logical value indicating whether alter-alter ties contain self-loops (alters can be tied to themselves).

missing_code — A numeric value indicating "missing" values in the alter-alter edgelist.

output_name — A character value indicating the name or prefix that should be given to output objects.

Value

A list containing three data frames: an ego list, an ego-alter edgelist, and an alter-alter edgelist. These dataframes are optimized for use with ego_netwrite.

---

ei_index *Krackhardt and Stern's E-I Index (*ei_index*)*

Description

Linear transformation of the proportion homophilous measure (Krackhardt and Stern 1988; Perry et al. 2018)

Usage

```
ei_index(
  ego_id,
  ego_measure,
  alter_ego,
  alter_measure,
  prefix = NULL,
  suffix = NULL
)
```

Arguments

ego_id	A vector of unique ego identifiers located in an ego dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled egos.
ego_measure	A vector of attributes corresponding to each ego
alter_ego	A vector of ego identifiers located in an alter dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled alters.
alter_measure	A vector of attributes corresponding to each alter
prefix	A character value indicating the desired prefix for the calculated E-I measure
suffix	A character value indicating the desired suffix for the calculated E-I measure

Value

ei_index returns a dataframe of vectors that include the ego identifier and the ei-index value for the selected attribute

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


# Calculate E-I Index for Race
race_ei <- ei_index(ego_id = ngq_nw$egos$ego_id, ego_measure = ngq_nw$egos$race,
                    alter_ego = ngq_nw$alters$ego_id, alter_measure = ngq_nw$alters$race,
                    prefix = "race")

race_ei
```

---

euclidean_distance *Euclidean Distance (*euclidean_distance*)*

Description

Typical difference between between ego and their alters for a given continuous attribute (Perry et al. 2018)

Usage

```
euclidean_distance(
  ego_id,
  ego_measure,
  alter_ego,
  alter_measure,
  prefix = NULL,
  suffix = NULL
)
```

Arguments

ego_id	A vector of unique ego identifiers located in an ego dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled egos.
ego_measure	A vector of attributes corresponding to each ego.

alter_ego	A vector of ego identifiers located in an alter dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled alters.
alter_measure	A vector of attributes corresponding to each alter.
prefix	A character value indicating the desired prefix for the calculated homophily measure.
suffix	A character value indicating the desired suffix for the calculated homophily measure.

Value

euclidean_distance returns a dataframe of vectors that include the ego identifier and euclidean distance for the desired continuous attribute

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


# Calculate Euclidean Distance
pol_euc <- euclidean_distance(ego_id = ngq_nw$egos$ego_id, ego_measure = ngq_nw$egos$pol,
                      alter_ego = ngq_nw$alters$ego_id, alter_measure = ngq_nw$alters$pol,
                              prefix = "pol")
pol_euc
```

---

fauxmesa_edges *Goodreau's Faux Mesa High School (Edgelist)*

Description

This data set (originally found in as a network object in the ergm package) represents a simulation of an in-school friendship network. The network is named "Faux Mesa High" because the school community on which it is based is in the rural western US, with a student body that is largely Hispanic and Native American.

Usage

```
fauxmesa_edges
```

Format

A data frame with 203 rows and 2 columns:

from Outgoing node

to Receiving node ...

Source

The data set is based upon a model fit to data from one school community from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

A vector representing the sex of each student in the school was randomly re-ordered. The same was done with the students' response to questions on race and grade. These three attribute vectors were permuted independently. Missing values for each were randomly assigned with weights determined by the size of the attribute classes in the school.

The following `ergm` formula was used to fit a model to the original data:

```
 ~ edges + nodefactor("Grade") + nodefactor("Race") +
nodefactor("Sex") + nodematch("Grade",diff=TRUE) +
nodematch("Race",diff=TRUE) + nodematch("Sex",diff=FALSE) +
gwdegree(1.0,fixed=TRUE) + gwesp(1.0,fixed=TRUE) + gwdsp(1.0,fixed=TRUE)
```

The resulting model fit was then applied to a network with actors possessing the permuted attributes and with the same number of edges as in the original data.

The processes for handling missing data and defining the race attribute are described in Hunter, Goodreau & Handcock (2008).

`fauxmesa_nodes` *Goodreau's Faux Mesa High School (Nodelist)*

Description

This data set (originally found in as a `network` object in the `ergm` package) represents a simulation of an in-school friendship network. The network is named "Faux Mesa High" because the school community on which it is based is in the rural western US, with a student body that is largely Hispanic and Native American.

Usage

```
fauxmesa_nodes
```

Format

A data frame with 205 rows and 4 columns:

id Node ID

grade Student grade year

race Student race

sex Student sex ...

Source

The data set is based upon a model fit to data from one school community from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

A vector representing the sex of each student in the school was randomly re-ordered. The same was done with the students' response to questions on race and grade. These three attribute vectors were permuted independently. Missing values for each were randomly assigned with weights determined by the size of the attribute classes in the school.

The following `ergm` formula was used to fit a model to the original data:

```
 ~ edges + nodefactor("Grade") + nodefactor("Race") +
nodefactor("Sex") + nodematch("Grade",diff=TRUE) +
nodematch("Race",diff=TRUE) + nodematch("Sex",diff=FALSE) +
gwdegree(1.0,fixed=TRUE) + gwesp(1.0,fixed=TRUE) + gwdsp(1.0,fixed=TRUE)
```

The resulting model fit was then applied to a network with actors possessing the permuted attributes and with the same number of edges as in the original data.

The processes for handling missing data and defining the race attribute are described in Hunter, Goodreau & Handcock (2008).

`florentine_edges` *Edgelist of marriage alliances and business relationships between Florentine families during the Italian Renaissance*

Description

Breiger & Pattison (1986), in their discussion of local role analysis, use a subset of data on the social relations among Renaissance Florentine families collected by John Padgett from historical documents. The two relations are business ties (recorded financial ties such as loans, credits and joint partnerships) and marriage alliances. This dataset has since become a standard for illustrating role analysis methods and working with networks featuring multiple types of relations.

Usage

```
florentine_edges
```

Format

A data frame with 35 rows and 4 columns:

source Outgoing node

target Receiving node

weight A placeholder variable for tie/edge weights, set to 1

type Relation type ...

Source

John Padgett

References

Ronald L. Breiger and Philippa E Pattison. 1986. "Cumulated social roles: The duality of persons and their algebras." *Social Networks* 8(13):215-256.

---

florentine_nodes — *Nodelist of marriage alliances and business relationships between Florentine families during the Italian Renaissance*

Description

Breiger & Pattison (1986), in their discussion of local role analysis, use a subset of data on the social relations among Renaissance Florentine families collected by John Padgett from historical documents. The two relations are business ties (recorded financial ties such as loans, credits and joint partnerships) and marriage alliances. This dataset has since become a standard for illustrating role analysis methods and working with networks featuring multiple types of relations.

Usage

```
florentine_nodes
```

Format

A data frame with 16 rows and 2 columns:

id Unique node ID number

family Name of family corresponding to node ...

Source

John Padgett

References

Ronald L. Breiger and Philippa E Pattison. 1986. "Cumulated social roles: The duality of persons and their algebras." *Social Networks* 8(13):215-256.

`football` *American College Football*

Description

Network of American football games between Division IA colleges during regular season Fall 2000.

Usage

```
football
```

Format

An 'igraph' object containing 613 edges between 115 vertices (nodes). Vertices contain three attributes:

id Unique identification number.

label Name of college team represented by vertex.

value A numberic indicator of football conference affiliation. ...

Source

Included by permission of M. Girvan and M. E. J. Newman (Website)

References

M. Girvan and M. E. J. Newman. 2002. "Community structure in social and biological networks." *Proc. Natl. Acad. Sci. USA* 99:7821-7826.

`get_CHAMP_map` *Find the SBM-equivalence iterative map on the CHAMP set of somewhere optimal partitions*

Description

`get_CHAMP_map` calculates the iterative map defined by Newman's equivalence between modularity optimization and inference on the degree-corrected planted partition stochastic block model on a *CHAMP* set of partitions. That is, given an input set of partitions of nodes in a network into communities, calculated by *get_partitions* or by other means and coerced into that format, `CHAMP` identifies which input partition is optimal at each value of the resolution parameter, gamma, and then `get_CHAMP_map` calculates the iterative map of this set onto itself. Importantly, a fixed point of this map, where a partition points to itself, indicates that partition is self-consistent in the sense of this equivalence between modularity and planted partition models. As with `CHAMP`, the `get_CHAMP_map` code is deterministic and fast given a specified input set of partitions; that is, all of the computational complexity and pseudo-stochastic heuristic nature of community detection is in identifying a good input set in *get_partitions*.

The CHAMP method was developed and studied in Weir, William H., Scott Emmons, Ryan Gibson, Dane Taylor, and Peter J. Mucha. "Post-Processing Partitions to Identify Domains of Modularity Optimization." Algorithms 10, no. 3 (August 19, 2017): 93. doi:10.3390/a10030093.

The equivalence between modularity optimization and planted partition inference was derived by M. E. J. Newman in "Equivalence between Modularity Optimization and Maximum Likelihood Methods for Community Detection." Physical Review E 94, no. 5 (November 22, 2016): 052315. doi:10.1103/PhysRevE.94.052315.

The iterative map on the CHAMP set was developed and studied in Gibson, Ryan A., and Peter J. Mucha. "Finite-State Parameter Space Maps for Pruning Partitions in Modularity-Based Community Detection." Scientific Reports 12, no. 1 (September 23, 2022): 15928. doi:10.1038/s41598-022201426.

See also https://github.com/wweir827/CHAMP and https://github.com/ragibson/ModularityPruning.

Usage

```
get_CHAMP_map(network, partitions, plotlabel = NULL, shiny = FALSE)
```

Arguments

network	The network, as igraph object, to be clustered into communities. Only undirected networks are currently supported. If the object has a 'weight' edge attribute, then that attribute will be used, though it is important to emphasize that the underlying equivalence between modularity and planted partitons defining the iterative map was derived for unweighted networks.
partitions	List of unique partitions with CHAMP summary generated by CHAMP.
plotlabel	Optional label to include as annotation on the generated figure.
shiny	A logical value indicating whether get_CHAMP_map is being called within ideanetViz. If TRUE, get_CHAMP_map returns an output data frame that ideanetViz uses for visualization.

Value

get_CHAMP_map returns the input list of partitions with the $CHAMPsummary updated to indicate the iterative map, that is, information about the next partition that each partition points to in the map, along with the generated $CHAMPmap plot of the partitions in the CHAMP set (by their numbers of communities) versus gamma. If shiny = TRUE, the returned list also includes a data frame entitled shiny_partitions that is used for visualizations in ideanetViz.

Author(s)

Peter J. Mucha (<peter.j.mucha@dartmouth.edu>), Alex Craig, Rachel Matthew, Sydney Rosenbaum and Ava Scharfstein

Examples

```
# Use get_partitions, CHAMP, and get_CHAMP_map to generate
# multiple partitions of the Zachary karate club and identify
# the domains of optimality in the resolution parameter for
```

```
# different partitions
data(karate, package = "igraphdata")
partitions <- get_partitions(karate, n_runs = 2500)
partitions <- CHAMP(karate, partitions, plottitle = "Weighted Karate Club")
partitions <- get_CHAMP_map(karate, partitions, plotlabel = "Weighted Karate Club")
```

---

get_egograph *Selecting Individual Networks from* ego_netwrite *Output (*get_egograph*)*

Description

The get_egograph function extracts one or more specific ego networks from a list object created by ego_netwrite. Specifically, it extracts the igraph objects associated with the ego networks selected by the user. This can be useful for close inspection and/or comparison of individual ego networks in your data.

Usage

```
get_egograph(egonw = NULL, ego_id = NULL)
```

Arguments

egonw	A list created by ego_netwrite.
ego_id	A single numeric value indicating the ego_id number associated with the ego network you want to extract, or a vector of numeric ego_id numbers.

Value

get_egograph returns a list containing the contents of the igraph_objects list found in egonw corresponding to the values specified in ego_id.

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)
```

```
# Select `igraph` objects associated with `ego_id` 3.
ego3 <- get_egograph(ngq_nw, 3)
```

`get_partitions` *Gather a collection of community detection partitions (*`get_partitions`*)*

Description

The `get_partitions` function is a wrapper to gather a collection of community detection partitions using igraph's `cluster_leiden` for maximizing modularity at various resolution parameter values, along with the routines called by the `comm_detect` function, to gather different partitions for subsequent input to the `CHAMP` code for post-processing partitions to identify domains of modularity optimization.

Usage

```
get_partitions(
  network,
  gamma_range = c(0, 3),
  n_runs = 100,
  n_iterations = 2,
  seed = NULL,
  add_comm_detect = TRUE
)
```

Arguments

`network`	The network, as igraph object, to be clustered into communities. Only undirected networks are currently supported. If the object has a 'weight' edge attribute, then that attribute will be used.
`gamma_range`	The range of the resolution parameter gamma (default from 0 to 4).
`n_runs`	The number of `cluster_leiden` runs to be attempted (default = 100).
`n_iterations`	Parameter to be passed to cluster_leiden (default = 2).
`seed`	Optional random seed for reproducing pseudo-random results.
`add_comm_detect`	Boolean to decide whether to also call the clustering algorithms included in `comm_detect` (default = T). Alternatively, the output of `comm_detect` can be provided directly here.

Value

`get_partitions` returns a list of unique partitions appropriate for subsequent input to `CHAMP`.

Author(s)

Peter J. Mucha (<peter.j.mucha@dartmouth.edu>), Alex Craig, Rachel Matthew, Sydney Rosenbaum and Ava Scharfstein

Examples

```
# Use get_partitions to generate multiple partitions of the
# Zachary karate club at different resolution parameters
data(karate, package = "igraphdata")
partitions <- get_partitions(karate, n_runs = 2500)
```

---

hightech *Multiplex Network of Relationships Between Managers of a High-Tech Company*

Description

A network of a small hi-tech computer firm that sold, installed, and maintained computer systems, represented as an edgelist. Relationships in the network can take on three modes: 1 represents advice relationships, 2 represents friendship relationships, and 3 represents chain of command (e.g., "reporting-to").

Usage

```
hightech
```

Format

A data frame with 312 rows and 4 columns:

node Outgoing node

target Receiving node

weight A placeholder variable for tie/edge weights, set to 1

layer Relation type ...

Source

Carnegie Mellon University

References

David Krackhardt. 1987. "Cognitive social structures". *Social Networks* 9(2):104-134. https://doi.org/10.1016/0378-8733(87)90009-8

h_index *H-Index (*h_index*)*

Description

Measure of ego network diversity for categorical attributes (Perry et al. 2018)

Usage

```
h_index(ego_id, measure, prefix = NULL, suffix = NULL)
```

Arguments

ego_id	A vector of ego identifiers located in an alter dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled alters.
measure	A vector of alter attributes for a given categorical measure.
prefix	A character value indicating the desired prefix for the calculated homophily measure.
suffix	A character value indicating the desired suffix for the calculated homophily measure.

Value

h_index returns a dataframe of vectors that include the ego identifier and h-index of diversity for the desired categorical attribute.

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


# Get H-index for race
race_hindex <- h_index(ego_id = ngq_nw$alters$ego_id,
                       measure = ngq_nw$alters$race,
                       prefix = "race")
```

race_hindex

---

ideanetViz *Interactive GUI for Working with Sociocentric Networks (*ideanetViz*)*

Description

ideanetViz is a Shiny app that presents the output of ideanet's workflow for sociocentric data (i.e. netwrite) in a clear and accessible GUI. This GUI is convenient for users with limited R experience and is useful for classrooms, workshops, and other educational spaces. It is also useful for experienced users interested in quick exploration of network data. Moreover, ideanetViz streamlines customization of network visualizations and provides quick access into ideanet's more advanced analytic tools for sociocentric networks.

ideanetViz's design is centered around a series of tabs lining the top of the app, which are ordered according to a typical workflow for acquiring, processing, exploring, and modeling data.

Usage

```
ideanetViz()
```

Value

Launches an external window in which users can interact with the ideanetViz GUI. At different points in working with the GUI, users have the option to export generated data as CSV files and visualizations as image files.

---

iqv *Agresti's Index of Qualitative Variation (*iqv*)*

Description

A normalized value of the h-index for measuring the diversity of an ego's network for categorical attributes (Perry et al. 2018)

Usage

```
iqv(ego_id, measure, prefix = NULL, suffix = NULL)
```

Arguments

ego_id	A vector of ego identifiers located in an alter dataframe. If using data objects created by ego_netwrite, this should be the data frame entitled alters.
measure	A vector of alter attributes for a given categorical measure.
prefix	A character value indicating the desired prefix for the calculated homophily measure.
suffix	A character value indicating the desired suffix for the calculated homophily measure.

Value

iqv returns a dataframe of vectors that include the ego identifier and iqv value of diversity for the desired categorical attribute.

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


# Get IQV for sex
sex_iqv <- iqv(ego_id = ngq_nw$alters$ego_id,
               measure = ngq_nw$alters$sex,
               prefix = "sex")

sex_iqv
```

---

marvel *Character Relations in Marvel Comics*

Description

A network, represented as edgelist, containing weighted edges between Marvel Comics characters. Edge weights were calculated based on how many times two characters' names appeared within 15 words of one another in a comic.

Usage

```
marvel
```

Format

A data frame with 9891 rows and 3 columns:

Source Outgoing node

Target Receiving node

Weight Edge weight ...

Source

Melanie Walsh (Github), adapted from data originally compiled by Cesc Rosselló, Ricardo Alberich, and Joe Miro from Russ Chappell (Website)

---

nc_merge *Merging Network Canvas CSV Files (*nc_merge*)*

Description

The nc_merge function combines CSV files exported from Network Canvas, a popular tool for egocentric data capture. It is designed to address issues that may be encountered by nc_read when Network Canvas exports separate CSV files for individual responses.

Usage

```
nc_merge(path, export_path)
```

Arguments

path	A character value indicating the directory in which Network Canvas CSVs are located. nc_read will read in all CSV files located in this directory and process them.
export_path	A character value indicating the directory to which merged CSV files should be exported. This should not be the same directory as path, and this function will return an error if it detects that path and export_path are equivalent.

Value

nc_merge always writes two CSV files to the directory specified in export_path: an ego list and an alters list. If CSV files containing alter-alter ties are detected, it also writes a third merged CSV of these ties.

nc_read *Reading and Reshaping Network Canvas Data (nc_read)*

Description

The nc_read function reads in and processes CSV files produced by Network Canvas, a popular tool for egocentric data capture. nc_read produces three dataframes optimized for use with ego_netwrite.

Usage

```
nc_read(path, protocol = NULL, cat.to.factor = TRUE)
```

Arguments

path	A character value indicating the directory in which Network Canvas CSVs are located. nc_read will read in all CSV files located in this directory and process them.
protocol	A character value indicating the pathname of the Network Canvas protocol file corresponding to the data being read. Reading in the protocol is optional but recommended for accurate encoding of categorical variables.
cat.to.factor	A logical value indicating whether categorical variables, originally stored as a series of TRUE/FALSE columns, should be converted into a single factor column.

Value

nc_read returns a list containing three items: an ego list, an ego-alter edgelist, and an alter-alter edgelist. If multiple edge types exist for ego-alter and/or alter-alter ties, edgelists for each type of tie will be stored as individual data frames as elements in a list. All data frames are optimized for use with ego_netwrite.

Note that in the alters data frame(s), column node_type reflects the "node type" assigned to a given alter as specified in a Network Canvas protocol. Values in node_type are not necessarily those which should be fed into the alter_types argument in ego_netwrite.

netread *Reading Network Data Files and Initial Cleaning (netread)*

Description

The netread function reads in various files storing relational data converts them into edgelists that ensure their compatibility with other ideanet functions.

Usage

```
netread(
  path = NULL,
  filetype = NULL,
  sheet = NULL,
  nodelist = NULL,
  node_sheet = NULL,
  object = NULL,
  col_names = TRUE,
  row_names = FALSE,
  format = NULL,
  net_name = "network",
  missing_code = 99999,
  i_elements = NULL,
  j_elements = NULL
)
```

Arguments

path	A character value indicating the path of the file which the data are to be read from. If netread is converting igraph or network objects, no file path is needed.
filetype	A character value indicating the type of file being read. Valid arguments are "csv", "excel" (.xls, .xlsx), "igraph" (for igraph objects), "network" or "sna" (for network objects), "pajek" (for Pajek files), and "ucinet" (for UCINet files).
sheet	If reading in an Excel file with multiple sheets, a character value indicating the name of the sheet on which the core relational data are stored.
nodelist	If the relational data being read have a corresponding file for node-level information, a character value indicating the path of the file which this data are to be read from.
node_sheet	If reading in an Excel file with multiple sheets, a character value indicating the name of the sheet on which the node-level information is store.
object	If converting an igraph or network object, the object to be converted.
col_names	For reading CSV and Excel files, a logical value indicating whether the first row in the file serves as the file's header and contains the names of each column.
row_names	For reading CSV and Excel files, a logical value indicating whether the first column in the file contains ID values for each row and should not be treated as part of the core data.
format	For reading CSV and Excel files, a character value indicating the format in which relational data are structured in the file. Valid arguments include "edgelist", "adjacency_matrix", and "adjacency_list".
net_name	A character value indicating the name of the network being read from the file(s). This name will be used as a prefix for both outputs created by netread.

missing_code	A numeric value indicating "missing" values in the data being read. Such "missing" values are sometimes included to identify the presence of isolated nodes in an edgelist when a corresponding nodelist is unavailable.
i_elements	If format is set to edgelist, a character value indicating the name of the column containing the sender of ties in the edgelist. If not specified, netread assumes the first column of the data represents tie senders.
j_elements	If format is set to edgelist, a character value indicating the name of the column containing the receiver of ties in the edgelist. If not specified, netread assumes the second column of the data represents tie receivers

Value

A list containing an edgelist and a nodelist, both of which are formatted to be compatible with the netwrite function.

---

netwrite *Network Cleaning and Variable Calculation (*netwrite*)*

Description

The netwrite function reads in relational data of several formats and processes them into a set of standardized outputs. These outputs include sets of commonly calculated measures at the individual node and network-wide levels.

Usage

```
netwrite(
  data_type = c("edgelist"),
  adjacency_matrix = FALSE,
  adjacency_list = FALSE,
  nodelist = FALSE,
  node_id = NULL,
  node_netid = NULL,
  edgelist = FALSE,
  i_elements = FALSE,
  j_elements = FALSE,
  edge_netid = NULL,
  fix_nodelist = TRUE,
  weights = NULL,
  type = NULL,
  remove_loops = FALSE,
  missing_code = 99999,
  weight_type = "frequency",
  directed = FALSE,
  net_name = "network",
  shiny = FALSE,
```

```
  output = c("graph", "largest_bi_component", "largest_component", "node_measure_plot",
     "nodelist", "edgelist", "system_level_measures", "system_measure_plot"),
   message = TRUE
)
```

Arguments

data_type	A character value indicating the type of relational data being entered into netwrite. Available options are edgelist, adjacency_matrix, and adjacency_list.
adjacency_matrix	If data_type is set to adjacency_matrix, a matrix object containing the adjacency matrix for the network being processed.
adjacency_list	If data_type is set to adjacency_list, a data frame containing the adjacency list for the network being processed.
nodelist	Either a vector of values indicating unique node/vertex IDs, or a data frame including all information about nodes in the network. If the latter, a value for node_id must be specified.
node_id	If a data frame is entered for the nodelist argument, node_id should be a character value indicating the name of the column in the node-level data frame containing unique node identifiers.
node_netid	If a data frame is entered for the nodelist argument, node_netid should be a character value indicating the name of the column in the node-level data frame containing unique network identifiers. This argument should be specified if a value is given for edge_netid.
edgelist	A data frame including all ties in the network. If this argument is specified, i_elements, j_elements, edge_netid (if applicable), weights (if applicable), and type (if applicable), must be specified as single character values indicating the names of their respective columns.
i_elements	If data_type is set to "edgelist", a vector of identifiers indicating the senders of ties in the edgelist, or a single character value indicating the name of the column in edgelist containing these identifiers.
j_elements	If data_type is set to "edgelist", a vector of identifiers indicating the receivers of ties in the edgelist, or a single character value indicating the name of the column in edgelist containing these identifiers.
edge_netid	If data_type is set to "edgelist", a vector of identifiers indicating the specific network to which a particular tie in the edgelist belongs, or a single character value indicating the name of the column in edgelist containing network identifiers.
fix_nodelist	If data_type is set to "edgelist" and user inputs a vector or data frame into nodelist, a logical value indicating whether to include node IDs that do not appear in the nodelist but do appear in the edgelist in the nodelist used when processing network data. By default, fix_nodelist is set to FALSE to identify potential inconsistencies between the nodelist and edgelist to the user.
weights	If data_type is set to "edgelist", a numeric vector indicating the weight of ties in the edgelist, or a single character value indicating the name of the column in edgelist containing tie weights. netwrite requires that all edge weights be positive values.

type	If data_type is set to "edgelist", a numeric or character vector indicating the types of relationships represented in the edgelist, or a single character value indicating the name of the column in edgelist containing tie types. If type is specified, netwrite will treat network(s) as multi-relational and produce additional outputs reflecting the different types of ties appearing in the data.
remove_loops	A logical value indicating whether "self-loops" (ties directed toward oneself) should be considered valid ties in the network being processed.
missing_code	A numeric value indicating "missing" values in an edgelist. Such "missing" values are sometimes included to identify the presence of isolated nodes in an edgelist when a corresponding nodelist is unavailable.
weight_type	A character value indicating whether edge weights should be treated as frequencies or distances. Available options are "frequency", indicating that higher values represent stronger ties, and "distance", indicating that higher values represent weaker ties. Note: some underlying functions assume that edges represent distances. If weight_type is set to "frequency", these functions will use the reciprocal of weights as distance values in calculation.
directed	A logical value indicating whether edges should be treated as a directed or undirected when constructing the network.
net_name	A character value indicating the name to which network/igraph objects should be given.
shiny	A logical value indicating whether netwrite is being used in conjunction with IDEANet's Shiny-based visualization app. shiny should also be set to TRUE when using ideanet in an R Markdown file that users expect to knit into a document.
output	A character vector indicating the kinds of objects netwrite should assign to the global environment. netwrite produces several outputs that may not all be necessary to a user's needs. Users can specify which outputs they specifically want in order to minimize the number of objects appearing in the global environment. Potential outputs include igraph object(s) ("graph"), subgraph(s) of only nodes that appear in the largest component and/or bicomponent of the network ("largest_component", "largest_bi_component"), data frame(s) containing node-level measures ("node_measure_plot"), a processed edgelist of the network ("edgelist"), a data frame indicating network-level summaries ("system_level_measures"), and summary visualizations for node- and network-level measures ("node_measure_plot", "system_measure_plot").
message	A logical value indicating whether warning messages should be displayed in the R console during processing.

Value

netwrite returns a list containing several output objects. Users may find it easier to access and work with outputs by applying list2env to this list, which will separate outputs and store them in the R Global Environment. Note, however, that this risks overwriting existing objects in the Global Environment should those objects share names with objects in netwrite's output. Depending on the values assigned to the output argument, netwrite will produce any or all of the following:

If output contains graph, netwrite will return an igraph object of the network represented in the original data. If a vector is entered into the type argument, netwrite also produces a list containing

igraph objects for each unique relation type as well as the overall network. These output objects are named according to the value specified in the net_name argument.

If output contains "nodelist", netwrite will return a dataframe containing individual-level information for each node in the network. This dataframe contains a set of frequently used node-level measures for each node in the network. If a vector is entered into the type argument, netwrite will produce these node-level measures for each unique relation type.

If output contains "edgelist", netwrite will return a formatted edgelist for the network represented in the original data. If a vector is entered into the type argument, netwrite also produces a list containing edgelists for each unique relation type as well as the overall network.

If output contains "system_level_measures", netwrite will return a data frame providing network-level summary information.

If output contains "node_measure_plot", netwrite will return a plot summarizing the distribution of frequently used node-level measures across all nodes in the network. If a vector is entered into the type argument, netwrite also produces a list containing node-level summary plots for each unique relation type as well as the overall network.

If output contains "system_measure_plot", netwrite will return a plot summarizing the distribution of frequently used network-level measures. If a vector is entered into the type argument, netwrite also produces a list containing network-level summary plots for each unique relation type as well as the overall network.

If output contains "largest_bi_component", netwrite will return an igraph object of the largest bicomponent in the network represented in the original data. If a vector is entered into the type argument, netwrite also produces a list containing the largest bicomponent for each unique relation type as well as the overall network.

If output contains "largest_bi_component", netwrite will return an igraph object of the largest main component in the network represented in the original data. If a vector is entered into the type argument, netwrite also produces a list containing the largest main component for each unique relation type as well as the overall network.

If users are working with data containing multiple independent networks, netwrite will return a list containing the above outputs for each network in their data, provided that users have passed a vector of network identifiers to the edge_netid argument. Each network's output will be labeled according to its corresponding value in edge_netid.

Examples

```
# Use netwrite on an edgelist
nw_fauxmesa <- netwrite(nodelist = fauxmesa_nodes,
                        node_id = "id",
                        i_elements = fauxmesa_edges$from,
                        j_elements = fauxmesa_edges$to,
                        directed = TRUE,
                        net_name = "faux_mesa")

### Inspect updated edgelist
head(nw_fauxmesa$edgelist)

### Inspect data frame of node-level measures
head(nw_fauxmesa$node_measures)
```

```
### Inspect system-level summary
head(nw_fauxmesa$system_level_measures)

### Plot sociogram of network
plot(nw_fauxmesa$faux_mesa)

### View node-level summary visualization
nw_fauxmesa$node_measure_plot

### View system-level summary visualization
nw_fauxmesa$system_measure_plot



# Run netwrite on an adjacency matrix

nw_triad <- netwrite(data_type = "adjacency_matrix",
                     adjacency_matrix = triad,
                     directed = TRUE,
                     net_name = "triad_igraph")
```

---

ngq_aa *Ego Networks Elicited from the "Important Matters" Name Generator Question (Alter-Alter Edgelist)*

Description

This dataset contains a simplified subset of 20 ego networks elicited using the "important matters" name generator question (NGQ), which is frequently used to capture an individual's close personal ties. These networks were collected as part of an experiment illustrating how networks generated by this question may vary depending on the topics covered in preceding survey items. Networks were collected using an online survey deployed via Amazon Mechanical Turk.

Usage

```
ngq_aa
```

Format

A data frame with 123 rows and 5 columns:

ego_id Unique identifier for ego providing network

alter1 Within-network unique identifier for Alter 1 in alter-alter edgelist.

alter2 Within-network unique identifier for Alter 2 in alter-alter edgelist.

type A character indicating the type of relationship that Alter 1 and Alter 2 have with one another. Note that each dyad-type combination has its own unique row in this dataset, so more than one row may correspond to a single dyad if the dyad involves multiple types of relationships.

freqtalk A numeric indicating how frequently ego believes Alter 1 and Alter 2 talk with one another. 1 indicates "Never," 2 "Less than once a month," 3 "1-3 times a month," 4 "1-3 times a week," 5 "Daily or almost daily." ...

Source

Original Data, Collected by Danielle Montagne, Joseph Quinn, Liann Tucker, and Tom Wolff.

`ngq_alters` *Ego Networks Elicited from the "Important Matters" Name Generator Question (Alter List)*

Description

This dataset contains a simplifed subset of 20 ego networks elicited using the "important matters" name generator question (NGQ), which is frequently used to capture an individual's close personal ties. These networks were collected as part of an experiment illustrating how networks generated by this question may vary depending on the topics covered in preceding survey items. Networks were collected using an online survey deployed via Amazon Mechanical Turk.

Usage

```
ngq_alters
```

Format

A data frame with 67 rows and 14 columns:

ego_id Unique identifier for ego providing network

alter_id Within-network unique identifier for person nominated by ego (alter).

sex A numeric indicating alter's sex as reported by ego. 1 indicates male, 2 female.

race A character indicating a simplified characterization of alter's race/ethnicity as reported by ego. Values include `"White"`, `"Black"`, and `"Other"`.

black A logical indicating ego's perception of alter as "Black" or "African-American."

white A logical indicating ego's perception of alter as "White."

other_race A logical indicating ego's perception of alter as belonging to a racial/ethnic group other than "Black," "African-American," or "White."

pol A numeric indicating political orientation on a seven-point scale, as perceived by ego. 1 indicates "Extremely Liberal," 4 "Moderate," and 7 "Extremely Conservative."

family A logical indicating alter as ego's family member.

friend A logical indicating alter as ego's friend.

other_rel A logical indicating alter as have a relationship to ego other than one of the types of relationships listed above.

face A numeric indicating how frequently ego and alter interact in person. 1 indicates "Never," 2 "Less than once a month," 3 "1-3 times a month," 4 "1-3 times a week," 5 "Daily or almost daily."

phone A numeric indicating how frequently ego and alter talk on the phone or via video chat. 1 indicates "Never," 2 "Less than once a month," 3 "1-3 times a month," 4 "1-3 times a week," 5 "Daily or almost daily."

text A numeric indicating how frequently ego and alter interact via electronic messaging (e.g. texting, email, social media). 1 indicates "Never," 2 "Less than once a month," 3 "1-3 times a month," 4 "1-3 times a week," 5 "Daily or almost daily." ...

Source

Original Data, Collected by Danielle Montagne, Joseph Quinn, Liann Tucker, and Tom Wolff.

`ngq_egos` *Ego Networks Elicited from the "Important Matters" Name Generator Question (Nodelist)*

Description

This dataset contains a simplified subset of 20 ego networks elicited using the "important matters" name generator question (NGQ), which is frequently used to capture an individual's close personal ties. These networks were collected as part of an experiment illustrating how networks generated by this question may vary depending on the topics covered in preceding survey items. Networks were collected using an online survey deployed via Amazon Mechanical Turk.

Usage

```
ngq_egos
```

Format

A data frame with 20 rows and 9 columns:

ego_id Unique identifier for ego providing network

age A numeric indicating ego's self-reported age

sex A numeric indicating ego's self-reported sex. 1 indicates male, 2 female.

race A character indicating a simplification of ego's self-reported race/ethnicity. Values include `"White"`, `"Black"`, and `"Other"`.

black A logical indicating ego's self-identification as "Black" or "African-American."

white A logical indicating ego's self-identification as "White."

other_race A logical indicating ego's self-identification with a race or ethnicity other than "Black," "African-American," or "White."

edu A numeric indicating ego's highest level of educational attainment. 1 indicates less than a high school diploma, 4 indicates a high school diploma or GED, 5 some college, 6 a college degree, and 7 a graduate or professional degree.

pol A numeric indicating ego's self-identified political orientation on a seven-point scale. 1 indicates "Extremely Liberal," 4 "Moderate," and 7 "Extremely Conservative." ...

Source

Original Data, Collected by Danielle Montagne, Joseph Quinn, Liann Tucker, and Tom Wolff.

`pearson_phi` *Pearson's Phi (*`pearson_phi`*)*

Description

The `pearson_phi` function identifies the underlying homophilous preference of ego based on the distribution of alter attributes in the population (Perry et al. 2018)

Usage

```
pearson_phi(
  ego_id,
  ego_measure,
  alter_ego,
  alter_measure,
  prefix = NULL,
  suffix = NULL
)
```

Arguments

`ego_id`	A vector of unique ego identifiers located in an ego dataframe. If using data objects created by `ego_netwrite`, this should be the data frame entitled `egos`.
`ego_measure`	A vector of attributes corresponding to each ego.
`alter_ego`	A vector of ego identifiers located in an alter dataframe. If using data objects created by `ego_netwrite`, this should be the data frame entitled `alters`.
`alter_measure`	A vector of attributes corresponding to each alter
`prefix`	A character value indicating the desired prefix for the calculated homophily measure.
`suffix`	A character value indicating the desired suffix for the calculated homophily measure.

Value

`pearson_phi` returns a dataframe of vectors that include the ego identifier and phi value of homophilous preference.

Examples

```
# Run `ego_netwrite`
ngq_nw <- ego_netwrite(egos = ngq_egos,
                       ego_id = ngq_egos$ego_id,

                       alters = ngq_alters,
                       alter_id = ngq_alters$alter_id,
                       alter_ego = ngq_alters$ego_id,

                       max_alters = 10,
                       alter_alter = ngq_aa,
                       aa_ego = ngq_aa$ego_id,
                       i_elements = ngq_aa$alter1,
                       j_elements = ngq_aa$alter2,
                       directed = FALSE)


race_pphi <- pearson_phi(ego_id = ngq_nw$egos$ego_id, ego_measure = ngq_nw$egos$race,
                     alter_ego = ngq_nw$alters$ego_id, alter_measure = ngq_nw$alters$race,
                         suffix = "race")

race_pphi
```

---

qap_run *Quadratic Assignment Procedure (*qap_run*).*

Description

The qap_run function is a wrapper around sna's Quadratic Assignment Procedure models sna::netlm and sna::netlogit. It expects a networks objects containing dependent and independent variables of interest. It is required to use the output from qap_setup.

Usage

```
qap_run(
  net,
  dependent = NULL,
  variables,
  directed = FALSE,
  family = "linear",
  reps = 500
)
```

Arguments

net	An igraph or network object.
dependent	A string naming the dependent variable of interest. By default, the probability of a tie. Can also be the output of qap_setup using prefixes "same_", "diff_" or "abs_diff_".

variables	A vector of strings naming the independent variables of interest. Must be the output of qap_setup using prefixes "same_", "diff_" and "abs_diff_", or suffixes "_ego" and "_alter".
directed	A logical statement identifying if the network should be treated as directed. Defaults to FALSE.
family	A string identifying the functional form. Options are "linear" and "binomial". Defauts to "linear".
reps	A numeric value indicating the number of draws. Defaults to 500.

Value

'qap_run' returns a list of elements that include:

- covs_df, a data frame containing term labels, estimates, standard errors and p-values

- mods_df, a data frame containing model-level information including the number of observations, AIC and BIC statistics.

Examples

```
flor <- netwrite(nodelist = florentine_nodes,
                 node_id = "id",
                 i_elements = florentine_edges$source,
                 j_elements = florentine_edges$target,
                 type = florentine_edges$type,
                 directed = FALSE,
                 net_name = "florentine_graph")

flor_setup <- qap_setup(flor$florentine_graph,
                        variables = c("total_degree"),
                        methods = c("difference"))

flor_qap <- qap_run(flor_setup$graph,
                    variables = c("diff_total_degree"))

# Inspect results
flor_qap$covs_df
```

---

qap_setup *Individual to Dyadic variable transformation (*qap_setup*).*

Description

The qap_setup function transform an individual level attributes into dyadic comparisons following a set of methods. Output can be used to compute QAP measurements using sister functions in ideanet.

Usage

```
qap_setup(
  net,
  variables = NULL,
  methods = NULL,
  directed = FALSE,
  additional_vars = NULL
)
```

Arguments

net	An igraph or network object.
variables	A vector of strings naming attributes to be transformed from individual-level to dyadic-level.
methods	A vector of strings naming methods to be applied to the variables vector. The methods vector must be the same length as the variables vector. Methods are applied in order (e.g, first method is applied to the first named attribute in variables). Possible methods are "reduced_category", "multi_category", "both", and "difference". For more information about methods, consult the included vignette.
directed	A logical statement identifying if the network should be treated as directed. Defaults to FALSE.
additional_vars	A data frame containing additional individual-level variables not contained in the primary network input. Additional dataframe must contain an id or label variables which matches network exactly.

Value

qap_setup returns a list of elements that include:

- graph, an updated igraph object containing the newly constructed dyadic variables and additional individual-level variables.

- nodes, a nodelist reflecting additional variables if included.

- edges, a nodelist reflecting new dyadic variables.

Examples

```
flor <- netwrite(nodelist = florentine_nodes,
                 node_id = "id",
                 i_elements = florentine_edges$source,
                 j_elements = florentine_edges$target,
                 type = florentine_edges$type,
                 directed = FALSE,
                 net_name = "florentine_graph")

flor_setup <- qap_setup(flor$florentine_graph,
                        variables = c("total_degree"),
```

```
methods = c("difference"))
```

`role_analysis` *Positional (Role) Analysis in Networks (*`role_analysis`*)*

Description

The `role_analysis` function takes networks processed by the `netwrite` function and performs positional analysis on them. Positional analysis methods allows users to infer distinct "roles" in networks from patterns in network activity. `role_analysis` currently supports the identification of roles using two methods: hierarchical clustering (cite) and convergence of correlations (CONCOR, Breiger 1975).

Usage

```
role_analysis(
  graph,
  nodes,
  directed = NA,
  method = "cluster",
  min_partitions = NA,
  max_partitions = NA,
  min_partition_size = NA,
  backbone = 0.9,
  viz = FALSE,
  fast_triad = NULL,
  retain_variables = FALSE,
  cluster_summaries = FALSE,
  dendro_names = FALSE,
  self_ties = FALSE,
  cutoff = 0.999,
  max_iter = 50
)
```

Arguments

`graph`	An igraph object or a list of igraph objects produced as output from `netwrite`.
`nodes`	A data frame containing individual-level network measures for each node in the network. Ideally, the `node_measures` data frame produced by `netwrite` should be assigned to this argument.
`directed`	A logical value indicating whether network edges should be treated as directed.
`method`	A character value indicating the method used for positional analysis. Valid arguments are currently `"cluster"` for hierarchical clustering and `"concor"` for CONCOR.

`min_partitions` A numeric value indicating the number of minimum number of clusters or partitions to assign to nodes in the network. When using hierarchical clustering, this value reflects the minimum number of clusters produced by analysis. When using CONCOR, this value reflects the minimum number of partitions produced in analysis, such that a value of 1 results in a partitioning of two groups, a value of 2 results in four groups, and so on.

`max_partitions` A numeric value indicating the number of maximum number of clusters or partitions to assign to nodes in the network. When using hierarchical clustering, this value reflects the maximum number of clusters produced by analysis. When using CONCOR, this value reflects the maximum number of partitions produced in analysis, such that a value of 1 results in a partitioning of two groups, a value of 2 results in four groups, and so on.

`min_partition_size`
A numeric value indicating the minimum number of nodes required for inclusion in a cluster. If an inferred cluster or partition contains fewer nodes than the number assigned to `min_partition_size`, nodes in this cluster/partition will be labeled as members of a parent cluster/partition.

`backbone` A numeric value ranging from 0-1 indicating which edges in the similarity/correlation matrix should be kept when calculating modularity of cluster/partition assignments. When calculating optimal modularity, it helps to backbone the similarity/correlation matrix according to the nth percentile. Larger networks benefit from higher `backbone` values, while lower values generally benefit smaller networks.

`viz` A logical value indicating whether to produce summary visualizations of the positional analysis.

`fast_triad` (Hierarchical clustering method only.) A logical value indicating whether to use a faster method for counting individual nodes' positions in different types of triads. This faster method may lead to memory issues and should be avoided when working with larger networks.

`retain_variables`
(Hierarchical clustering method only.) A logical value indicating whether output should include a data frame of all node-level measures used in hierarchical clustering.

`cluster_summaries`
(Hierarchical clustering method only.) A logical value indicating whether output should includde a data frame containing by-cluster mean values of variables used in hierarchical clustering.

`dendro_names` (Hierarchical clustering method only.) A logical value indicating whether the cluster dendrogram visualization should display node labels rather than ID numbers.

`self_ties` (CONCOR only.) A logical value indicting whether to include self-loops (ties directed toward oneself) in CONCOR calculation.

`cutoff` (CONCOR only.) A numeric value ranging from 0 to 1 that indicates the correlation cutoff for detecting convergence in CONCOR calculation.

`max_iter` (CONCOR only.) A numeric value indicating the maximum number of iteractions allowed for CONCOR calculattion.

Value

The role_analysis returns a list of outputs that users can access to help interpret results. This contents of this list varies somewhat depending on the method being used for positional analysis.

When hierarchical clustering is used, the list contains the following: cluster_assignments is a data frame indicating each node's membership within inferred clusters at each level of partitioning. cluster_sociogram contains a visualization of the network wherein nodes are colored by their membership within clusters at the optimal level of partitioning. cluster_dendrogram is a visualization of the dendrogram produced from clustering nodes. Red boxes on the visualization indicate nodes' cluster memberships at the optimal level of partitioning. cluster_modularity is a visualization of the modularity scores of the matrix of similarity scores between nodes for each level of partitioning. This visualization helps identify the optimal level of partitioning inferred by the role_analysis function. cluster_summaries_cent contains one or more visualization representing how clusters inferred at the optimal level of partitioning differ from one another on several important node-level measures. cluster_summaries_triad contains one or more visualization representing how clusters inferred at the optimal level of partitioning differ from one another on in terms of their positions within certain kinds of triads in the network. cluster_relations_heatmaps is a list object containing several heatmap visualizations representing the extent to which nodes in one inferred cluster are connected to nodes in another cluster. cluster_relations_sociogram contains a network visualization representing the extent to which nodes in clusters inferred at the optimal level of partitioning are tied to one another. Nodes in this visualization represent inferred clusters in the aggregate.

When CONCOR is used, this list contains the following: concor_assignments is a data frame indicating each node's membership within inferred blocks at each level of partitioning. concor_sociogram contains a visualization of the network wherein nodes are colored by their membership within blocks at the optimal level of partitioning. concor_block_tree is a visualization representing how smaller blocks are derived from larger blocks at each level of partitioning using CONCOR. concor_modularity is a visualization of the modularity scores of the matrix of similarity scores between nodes for each level of partitioning. This visualization helps identify the optimal level of partitioning inferred by the role_analysis function. concor_relations_heatmaps is a list object containing several heatmap visualizations representing the extent to which nodes in one inferred block are connected to nodes in another block. concor_relations_sociogram contains a network visualization representing the extent to which nodes in blocks inferred at the optimal level of partitioning are tied to one another. Nodes in this visualization represent inferred blocks in the aggregate.

Examples

```
flor <- netwrite(nodelist = florentine_nodes,
                 node_id = "id",
                 i_elements = florentine_edges$source,
                 j_elements = florentine_edges$target,
                 type = florentine_edges$type,
                 directed = FALSE,
                 net_name = "florentine")

# Clustering method
flor_cluster <- role_analysis(graph = flor$igraph_list,
                              nodes = flor$node_measures,
                              directed = FALSE,
```

```
                              method = "cluster",
                              min_partitions = 2,
                              max_partitions = 8,
                              viz = TRUE)

### View cluster dendrogram
flor_cluster$cluster_dendrogram

### View modularity summary plot
flor_cluster$cluster_modularity

### View cluster assignments
head(flor_cluster$cluster_assignments)

### View centrality summary plot for aggregate network
flor_cluster$cluster_summaries_cent$summary_graph
### View cenrality summary plot for network of relation `business`
flor_cluster$cluster_summaries_cent$business

### View triad position summary plot for network of relation `marriage`
flor_cluster$cluster_summaries_triad$marriage


# CONCOR method
flor_concor <- role_analysis(graph = flor$igraph_list,
                             nodes = flor$node_measures,
                             directed = FALSE,
                             method = "concor",
                             min_partitions = 1,
                             max_partitions = 4,
                             viz = TRUE)

### View CONCOR tree
flor_concor$concor_block_tree

### View modularity summary plot
flor_concor$concor_modularity

### View cluster assignments
head(flor_concor$concor_assignments)

### View chi-squared heatmaps of relations between blocks
flor_concor$concor_relations_heatmaps$chisq
```

`triad` *A Small Network Containing all Triads and Motifs*

Description

An adjacency matrix representing a network of 9 nodes, the ties between which form all possible triads and 3-node motifs that can appear in a directed network.

Usage

```
triad
```

Format

A matrix with 9 rows and 9 columns

Index