Package 'memnet'

November 3, 2018

Type Package

Title Network Tools for Memory Research

Description Efficient implementations of network science tools
to facilitate research into human (semantic) memory. In its current version, the package contains several methods to infer networks from verbal fluency data, various network growth models, diverse (switcher-) random walk processes, and tools to analyze and visualize networks. To deliver maximum performance the majority of the code is written in C++. For an application see: Wulff, D. U., Hills, T., & Mata, R. (2018) <doi:10.31234/osf.io/s73dp>.

Version 0.1.0

Maintainer Dirk U. Wulff <dirk.wulff@gmail.com>

License GPL-3

Encoding UTF-8

Depends R (>= 3.1.0), Rcpp (>= 0.12.5)

Imports igraph

LinkingTo Rcpp, BH

RoxygenNote 6.1.0

Suggests knitr, rmarkdown

VignetteBuilder knitr

LazyData true

NeedsCompilation yes

Author Dirk U. Wulff [aut, cre]

Repository CRAN

Date/Publication 2018-11-03 16:20:06 UTC

R topics documented:

`alc` *Average local clustering (alc) coefficient*

Description

Computes the uncorrected or corrected average local clustering coffiecient.

Usage

```
alc(adj, types = "uncorrected", weights = NULL, mode = "undirected")
```

Arguments

adj	numeric matrix representing the adjacency matrix.
types	character. Either "uncorrected" or "corrected", or a vector containing both.
weights	numeric vector of edge weights. Optional.
mode	character, either "directed" or "undirected", specifying whether the network should be interepeted as directed or undirected. Defaults to "undirected".

Details

The uncorrected clustering coefficent is computed according to Watts & Strogatz (1998). The corrected clustering coefficient normalizes the uncorrected one by the `average degree / n nodes`, i.e., the expected average local clustering for an Erdös-Renyi random graph.

Value

the corrected local clustering coefficient and/or the uncorrected clustering coefficient.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list = threshold_graph(animal_fluency)

# get adjacency matrices
adj = edg_to_adj(edge_list)

# get local average clustering coefficient
alc(adj)

# get corrected local average clustering coefficient
alc(adj, types = 'corrected')
```

---

animal_fluency *Animal fluency data.*

Description

A dataset containing the animal fluency sequences of a lifespan sample of 201 individuals collected at Stanford university. See references.

Usage

```
animal_fluency
```

Format

A list of 201 animal fluency sequences named by the participants age.

Source

data obtained from published papers. See referneces.

References

Hills, T. T., Mata, R., Wilke, A., & Samanez-Larkin, G. R. (2013). Mechanisms of age-related decline in memory search across the adult life span. Developmental psychology, 49(12), 2396. Wulff, D. U., Hills, T. T., Lachman, M., & Mata, R. (2016). The Aging Lexicon: Differences in the Semantic Networks of Younger and Older Adults. Cognitive Science Society.

`aspl` *Average shortest path length (aspl)*

Description

Computes the average shortest path length (aspl) using `igraph`'s automatic method.

Usage

```
aspl(adj, types = "uncorrected", weights = NULL, mode = "undirected")
```

Arguments

`adj`	numeric matrix representing the adjacency matrix.
`types`	character. Either `"uncorrected"` or `"corrected"`, or a vector containing both.
`weights`	numeric vector of edge weights. Optional.
`mode`	character, either `"directed"` or `"undirected"`, specifying whether the network should be interepeted as directed or undirected. Defaults to `"undirected"`.

Details

Per default the uncorrected apsl is computed. Otherwise, the uncorrected aspl is normalized by `log(n nodes)/log(average degree)`, i.e., the expected average shortest path length for an Erdös-Renyi random graph.

Value

Local clustering coefficient.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list = threshold_graph(animal_fluency)

# get adjacency matrices
adj = edg_to_adj(edge_list)

# get average shortest path length
aspl(adj)

# get corrected average shortest path length
aspl(adj, types = 'corrected')
```

---

cmix *Fast general purpose color mixer*

Description

Mixes two colors or matching vectors of colors according to some relative weight and exports the result either in rgb or hex format.

Usage

```
cmix(col_1, col_2, weight, format = "hex")
```

Arguments

`col_1, col_2`	character vector of length one or of matching length containing colors either as a color name (see colors), rgb format (see rgb), or hex format.
`weight`	numeric between 0 and 1 specifying the relative mixing weight for color one. E.g., `weight = .8` means that final color is composed of 80 percent `col_2` and 20 percent `col_1`.
`format`	character string specifying the output format. Either `"hex"` or `"rgb"`.

Value

A vector of length `max(length(col_1), length(col_2))` containing the mixed colors in the specified format.

Examples

```
# mix blue and red with more weight on blue
cmix('blue', 'red', .2)

# mix blue and red with more weight on red
cmix('blue', 'red', .8)

# mix blue and red and return as rgb
cmix('blue', 'red', .8, format = 'rgb')
```

---

common_subgraphs *Get common subgraph*

Description

Function identifies and returns the common subgraphs of two networks.

Usage

```
common_subgraphs(adj_1, adj_2, weights_1 = NULL, weights_2 = NULL,
  mode_1 = "undirected", mode_2 = "undirected")
```

Arguments

adj_1	numeric matrix representing the adjacency matrix of graph 1.
adj_2	numeric matrix representing the adjacency matrix of graph 2.
weights_1	numeric vector of edge weights for network 1 and 2, respectively. Optional.
weights_2	numeric vector of edge weights for network 1 and 2, respectively. Optional.
mode_1	character, either "directed" or "undirected", specifying whether network 1 and 2 should be interepeted as directed or undirected, respectively. Defaults to "undirected".
mode_2	character, either "directed" or "undirected", specifying whether network 1 and 2 should be interepeted as directed or undirected, respectively. Defaults to "undirected".

Value

List containing the two subgraphs.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list_1 = threshold_graph(animal_fluency[1:100])
edge_list_2 = threshold_graph(animal_fluency[101:200])

# get adjacency matrices
adj_1 = edg_to_adj(edge_list_1)
adj_2 = edg_to_adj(edge_list_2)

# get common subgraph
common_subgraphs(adj_1, adj_2)
```

---

common_subgraph_stats *Common subgraph statistics*

Description

Function identifies the common subgraphs of two networks and returns the networks statistics using network_stats.

Usage

```
common_subgraph_stats(adj_1, adj_2, giant = FALSE, weights_1 = NULL,
  weights_2 = NULL, mode_1 = "undirected", mode_2 = "undirected")
```

Arguments

adj_1	numeric matrix representing the adjacency matrix of graph 1.
adj_2	numeric matrix representing the adjacency matrix of graph 2.
giant	logical specifying whether the should be computed for the largest component.
weights_1	numeric vector of edge weights for network 1 and 2, respectively. Optional.
weights_2	numeric vector of edge weights for network 1 and 2, respectively. Optional.
mode_1	character, either "directed" or "undirected", specifying whether network 1 and 2 should be interepeted as directed or undirected, respectively. Defaults to "undirected".
mode_2	character, either "directed" or "undirected", specifying whether network 1 and 2 should be interepeted as directed or undirected, respectively. Defaults to "undirected".

Value

List containing the two subgraphs.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list_1 = threshold_graph(animal_fluency[1:100])
edge_list_2 = threshold_graph(animal_fluency[101:200])

# get adjacency matrices
adj_1 = edg_to_adj(edge_list_1)
adj_2 = edg_to_adj(edge_list_2)

# get structural overview of both networks
common_subgraph_stats(adj_1, adj_2)
```

---

community_graph *Create community graph*

Description

Create a graph from verbal fluency data by adding edges for words that occur within a window size `l` and retaining those that occur more frequently than `min_cooc` and the expectations number of chance productions co- occurences based on `1-crit`.

Usage

```
community_graph(dat, l = 3L, min_cooc = 2L, crit = 0.05)
```

Arguments

`dat`	list of character vectors containing the fluency productions.
`l`	an integer specifying the window size. The internal upper limit of `l` is the number of productions.
`min_cooc`	integer specifying the minimum number of times two words have to coocur within a window size of `l` to consider including an edge between them.
`crit`	a numeric within `[0,1]` specifiying the type-1 error rate of including an edge between unconnected words.

Value

A matrix

References

Goni, J., Arrondo, G., Sepulcre, J., Martincorena, I., de Mendizábal, N. V., Corominas-Murtra, B., ... & Villoslada, P. (2011). The semantic organization of the animal category: evidence from semantic verbal fluency and network theory. Cognitive processing, 12(2), 183-196.

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Examples

```
# get animal fluency data
data(animal_fluency)

# infer influence network
inferred_network = community_graph(animal_fluency)

# Simulate -----

# generate watts strogatz graph
network = grow_ws(n = 200, k = 10, p = .5)

# generate fluency data
# sets string equal TRUE as community_graph expects mode character
fluency_data = fluency(get_adjlist(network), rep(10, 100), string = TRUE)

# infer fluency network
inferred_network = community_graph(fluency_data)
```

---

edg_to_adj *Edge list to adjacency matrix*

Description

Transforms an edge list as returned by community_graph into an adjacency matrix or `adjlist`.

Usage

```
edg_to_adj(edg, weight = NULL, directed = FALSE, adjlist = FALSE)
```

Arguments

`edg`	character matrix with two columns containing the from and to nodes of the edges.
`weight`	optional numeric vector specifiying the weights associated with each edge.
`directed`	logical specifying whether edges should only be included for from-to or also fro to-from.
`adjlist`	logical specifying whether to export the adjancy matrix as an `adjlist` as required by fluency.

Value

a numeric n x n adjacency matrix with n being the number of unique entries in edg.

Examples

```
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency[1:3])

# transfrom to adjacency matrix
edg_to_adj(edge_list)
```

---

edg_to_adjlist *Edge list to adjlist*

Description

Transforms an edge list as returned by community_graph into an adjlist as required by, e.g., fluency.

Usage

```
edg_to_adjlist(edg)
```

Arguments

edg	character matrix with two columns containing the from and to nodes of the edges.

Value

a numeric n x n adjacency matrix with n being the number of unique entries in edg.

Examples

```
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency[1:3])

# transfrom to adjacency matrix
edg_to_adjlist(edge_list)
```

`ffluency` *Fast verbal fluency generator*

Description

Generates multiple verbal fluency sequences using one_ffluency.

Usage

```
ffluency(adjlist, n, pjump = 0, type = 0L, string = FALSE)
```

Arguments

`adjlist`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`n`	integer vector specifying for each sequence the maximum numbers of productions. Function may return fewer than `n`.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.
`string`	logical specifying whether the output should be of mode character.

Details

For details see one_ffluency.

Value

List of character vectors containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# create verbal fluency sequences
ffluency(get_adjlist(network), c(10, 10))

# create verbal fluency sequence
# with high jump probability
ffluency(get_adjlist(network), c(10, 10), pjump = .5)
```

`fluency` *Repeated verbal fluency generator.*

Description

Generates multiple verbal fluency sequences using one_fluency.

Usage

```
fluency(adjlist, n, pjump = 0, type = 0L, string = FALSE)
```

Arguments

`adjlist`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`n`	integer vector specifying for each sequence the number of unique productions.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.
`string`	logical specifying whether the output should be of mode character.

Details

For details see one_fluency.

Value

List of character vectors containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 3)

# create verbal fluency sequences
fluency(get_adjlist(network), c(10, 10))

# create verbal fluency sequence
# with high jump probability
fluency(get_adjlist(network), c(10, 10), pjump = .5)
```

`fluency_steps` *Verbal fluency step counter*

Description

Repeatedly generates verbal fluency data using one_fluency_steps and counts the number of steps required to produce `n` unique responses.

Usage

```
fluency_steps(adjlist, n, pjump = 0, type = 0L)
```

Arguments

`adjlist`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`n`	integer vector specifying the numbers of production.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.

Details

For details see one_fluency_steps.

Value

List of character vectors containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# count number of steps needed to create sequence
fluency_steps(get_adjlist(network), c(10, 10))

# count number of steps needed to create sequence
# with high jump probability
fluency_steps(get_adjlist(network), c(10, 10), pjump = .5)
```

---

get_adjlist *Get adjacency list*

Description

Get list containing adjacent, i.e., neighboring, nodes for each node in the graph. Nodes are returned as their row indices of the adjacency matrix.

Usage

```
get_adjlist(adj)
```

Arguments

adj	numeric matrix specifying the adjacency matrix.

Value

A list of vectors containing the indices of adjacent nodes.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 6, k = 2, p = .5)

# transform to adjlist
get_adjlist(network)
```

`get_kneighbors` *Get vector of neighbors exactly k steps away*

Description

Function iterates over graph to identify for a given node all nodes that are exactly k steps apart.

Usage

```
get_kneighbors(adj, start, k = 1L)
```

Arguments

`adj`	numeric matrix specifying the adjacency matrix.
`start`	integer specifying the row index of the start node in the adjacency matrix.
`k`	integer specifying the exact distance to the start node.

Details

`k < 1` will be set to `k = 1`.

Value

A numeric vector containing node indices of nodes k or fewer steps away from `start`.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10, p = .5)

# get neighborhood of second node
get_kneighbors(network, 2)

# get 3-hop neighborhood of second node
get_kneighbors(network, 2, k = 3)
```

get_names *Get node names of memnet objects*

Description

Function extracts names from objects created from edgelists and adjacency matrices.

Usage

```
get_names(dat)
```

Arguments

dat	numeric or character matrix containing an edgelist or adjacency matrix.

Value

lists of character vectors.

Examples

```
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency)

# get names
get_names(edge_list)
```

get_neighborhood *Get neighbors* k *or fewer steps away*

Description

Function iterates over graph to identify for a given node all nodes that are no more than k steps apart.

Usage

```
get_neighborhood(adj, start, k = 1L)
```

Arguments

adj	numeric matrix specifying the adjacency matrix.
start	integer specifying the row of the start node in the adjacency matrix.
k	integer specifying the maximum distance to the start node.

Details

k < 1 will be set to k = 1.

Value

A numeric matrix of two columns containing nodes indices and the geodesic distance to the start nodes of all nodes k or fewer steps away from start.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10, p = .5)

# get neighborhood of second node
get_neighborhood(network, 2)

# get 3-hop neighborhood of second node
get_neighborhood(network, 2, k = 3)
```

---

grow_ba *Barabási & Albert (2002) network growth model*

Description

Grow networks using Barabási & Alberts's (2002) preferential attachment model.

Usage

```
grow_ba(n = 100L, m = 5L, power = 1)
```

Arguments

n	Integer. Number of nodes in the network.
m	Integer. Number of edges added for each incoming node.
power	Numeric. Controls the selection of nodes by raising the degree to this power.

Value

n x n adjacency matrix.

References

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. Science, 286(5439), 509-512.

Examples

```
# generate small graph
grow_ba(n = 6, m = 2)

## Not run:
# generate large graph, flat degree distribution
grow_ba(n = 100, m = 10, p = .1)

# generate large graph, steep degree distribution
grow_ba(n = 100, m = 10, p = 10)

## End(Not run)
```

---

grow_hk *Holme and Kim (2002) network growth model*

Description

Grow networks using Holme & Kim's (2002) model, which combines preferential attachment with tunable triad formation flexibly controlling the amount of clustering in the network via p.

Usage

```
grow_hk(n = 100L, m = 5L, p = 5)
```

Arguments

n	Integer. Number of nodes in the network.
m	Integer. Number of edges added for each incoming node.
p	Numeric. Proability that a triad formation step follows a preferential attachment step.

Value

n x n adjacency matrix.

References

Holme, P., & Kim, B. J. (2002). Growing scale-free networks with tunable clustering. Physical review E, 65(2), 026107.

Examples

```
# generate small graph
grow_hk(n = 6, m = 2, p = .1)

## Not run:
# generate large graph, low clustering
grow_hk(n = 100, m = 10, p = .1)

# generate large graph, high clustering
grow_hk(n = 100, m = 10, p = .9)

## End(Not run)
```

---

grow_lattice *Regular lattice network model*

Description

Grow regular lattice networks, in which every node is connected to m neighbors.

Usage

```
grow_lattice(n = 100L, k = 10L)
```

Arguments

n	Integer. Number of nodes in the network.
k	Integer. Number of edges added for each incoming node. Can only be even.

Value

n x n adjacency matrix.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. Nature, 393(6684), 440-442.

Examples

```
# generate small lattice
grow_lattice(n = 6, k = 2)

## Not run:
# generate large lattice
grow_lattice(n = 100, k = 10, p = .1)

## End(Not run)
```

grow_st *Steyvers and Tenenbaum (2004) network growth model*

Description

Grow networks using Steyvers and Tenenbaum (2004) model, which combines preferential attachment with a triad formation.

Usage

```
grow_st(n = 100L, m = 5L)
```

Arguments

n	Integer. Number of nodes in the network.
m	Integer. Number of edges added for each incoming node.

Value

n x n adjacency matrix.

References

Steyvers, M., & Tenenbaum, J. B. (2005). The large-scale structure of semantic networks: Statistical analyses and a model of semantic growth. Cognitive science, 29(1), 41-78.

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Examples

```
# generate small graph
grow_st(n = 6, m = 2)

## Not run:
# generate large graph
grow_st(n = 100, m = 10)

## End(Not run)
```

grow_ws *Watts & Strogatz (2002) network growth model*

Description

Grow networks using Watts & Strogatz (1999) growth model, which constructs in-between regular lattices and random networks by re-wiring edges of a regular lattice with probability p.

Usage

```
grow_ws(n = 100L, k = 10L, p = 0.2)
```

Arguments

n	Integer. Number of nodes in the network.
k	Integer. Number of edges added for each incoming node. Can only be even.
p	Numeric. Proability that an edge e_ij is rewired to e_ik with k being randomly drawn from the set of nodes.

Value

n x n adjacency matrix.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. Nature, 393(6684), 440-442.

Examples

```
# generate small, mildly random graph
grow_ws(n = 6, k = 2, p = .2)

## Not run:
# generate large, mildly random graph
grow_ws(n = 100, k = 10, p = .1)

# generate large, highly random graph
grow_ws(n = 100, k = 10, p = 10)

## End(Not run)
```

k_dist *Maximum difference between cumulative degree distribution*

Description

Determines the maximum difference between the cumulative degree distributions of two graphs

Usage

```
k_dist(adj_1, adj_2, weights_1 = NULL, weights_2 = NULL,
  mode_1 = "undirected", mode_2 = "undirected")
```

Arguments

adj_1	numeric matrix representing the adjacency matrix of graph 1.
adj_2	numeric matrix representing the adjacency matrix of graph 2.
weights_1, weights_2	numeric vector of edge weights for network 1 and 2, respectively. Optional.
mode_1, mode_2	character, either "directed" or "undirected", specifying whether network 1 and 2 should be interepeted as directed or undirected, respectively. Defaults to "undirected".

Value

Distance between cumulative degree distributions.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list_1 = threshold_graph(animal_fluency[1:100])
edge_list_2 = threshold_graph(animal_fluency[101:200])

# get adjacency matrices
adj_1 = edg_to_adj(edge_list_1)
adj_2 = edg_to_adj(edge_list_2)

# get max degree distance
k_dist(adj_1, adj_2)
```

`l_comp` *Retrieve largest component*

Description

Retrieves the largest component. In case of equally sized components the first is component is retrieved.

Usage

```
l_comp(adj, weights = NULL, mode = "undirected", igraph = FALSE)
```

Arguments

`adj`	numeric matrix representing the adjacency matrix.
`weights`	numeric vector of edge weights. Optional.
`mode`	character, either `"directed"` or `"undirected"`, specifying whether the network should be interepeted as directed or undirected. Defaults to `"undirected"`.
`igraph`	logical specifying whether the output should be of class `"igraph"`.

Value

A list containing the, now, named adjacency matrix and a numeric value indicating the size of the largest component relative to to the entire graph.

Examples

```
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency[1:10])

# get adjacency matrix
adj = edg_to_adj(edge_list)

# get largest component
l_comp(adj)
```

neighborhood_plot *Neighborhood plot*

Description

Plot k-Neighborhood of given node containing all nodes with distances larger than k from given node.

Usage

```
neighborhood_plot(adj, names = NULL, node, k = 2,
  nod_col = "#E0EED4", nod_shading = TRUE, ...)
```

Arguments

adj	numeric matrix representing the adjacency matrix. Can also be an object of class "igraph" or an edge list, i.e., a two-column matrix or data.frame containing specifying the edges start and end points.
names	optional character vector specifiying the node names.
node	integer specifying the row index (within the adjacency matrix) of the node whose the neighborhood should be plotted. Alternatively the node name.
k	integer specifying the size of the neighborhood. Specifically, the plot will contain all nodes that are k or fewer steps away from v.
nod_col	character vector of length 1 specifying the node colors.
nod_shading	logical specifying whether the node colors should be shaded as a function of the distance to node.
...	arguments to be passed to network_plot.

Value

nothing. A plot is created in dev.cur.

Examples

```
## Not run:
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency[1:40])

# get adjacency matrix
adj = edg_to_adj(edge_list)

# plot
```

```
neighborhood_plot(adj, node = 'dog', k = 2)

## End(Not run)
```

`network_plot` *Plot graph*

Description

Custom graph plot using `igraph`'s layout functions.

Usage

```
network_plot(adj, names = NULL,
  layout = igraph::layout.fruchterman.reingold, nod_col = "#E0EED4",
  nod_cex = 3, nod_shadow = T, edg_col = "grey25", edg_lwd = 1.5,
  lab_col = "black", lab_cex = 1, lab_lwd = 1, lab_lcol = "grey25",
  lab_grid_size = 48, lab_padding = c(3, 3))
```

Arguments

`adj`	numeric matrix representing the adjacency matrix. Can also be an object of class `"igraph"` or an edge list, i.e., a two-column `matrix` or `data.frame` containing specifying the edges start and end points.
`names`	optional character vector specifying the node names. Must be of appropriate length.
`layout`	layout function from the `igraph` package. Default is layout.fruchterman.reingold.
`nod_col`	character vector of length 1 or length |V| specifying the node colors.
`nod_cex`	numeric speciying the size of the node circles.
`nod_shadow`	logical specifiying whether nodes should have shadows. Node shodow color is created from darkening `node_col`.
`edg_col`	character vector of length 1 of length |E| specifying the edge line colors.
`edg_lwd`	numeric vector of length 1 of length |E| specifying the edge line widths.
`lab_col`	character vector of length 1 of length |V| specifying the text label colors.
`lab_cex`	numeric vector of length 1 of length |V| specifying the text label sizes.
`lab_lwd`	numeric vector of length 1 of length |V| specifying the width of the lines connecting the text label to the nodes.
`lab_lcol`	character vector of length 1 of length |E| specifying specifying the color of the lines connecting the text label to the nodes.
`lab_grid_size`	integer specifying the grid size used to place the node labels. Canvas is split in `lab_grid_size` and labels are placed into cells closest to the associated node.
`lab_padding`	numeric vector of length 2 specifying the spacing among labels and between labels and nodes on the x and y dimension.

Value

nothing. A plot is created in dev.cur.

Examples

```
## Not run:
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency[1:20])

# get adjacency matrix
adj = edg_to_adj(edge_list)

# plot
network_plot(adj)

## End(Not run)
```

`network_stats` *Network statistics*

Description

Function computes various network measures including the size of the number of nodes (|V|), the number of edges (|E|), the average degree (k), uncorrected and corrected average local clustering (C, Cc), uncorrected and corrected average shortest path lengths (L, Lc), the small-world index (S) from Humphries & Gurney (2008), assortivity (A), and the size of largest component (p) relative to the entire network.

Usage

```
network_stats(adj, giant = FALSE, weights = NULL,
  mode = "undirected")
```

Arguments

`adj`	numeric matrix representing the adjacency matrix.
`giant`	logical specifying whether the should be computed for the largest component.
`weights`	numeric vector of edge weights. Optional.
`mode`	character, either `"directed"` or `"undirected"`, specifying whether the network should be interepeted as directed or undirected. Defaults to `"undirected"`.

Value

A named numeric vector containing the computed network statistics.

Examples

```
# get fluency data
data(animal_fluency)

# edge lists of fluency graphs
edge_list = threshold_graph(animal_fluency)

# get adjacency matrices
adj = edg_to_adj(edge_list)

# get structural overview
network_stats(adj)
```

---

one_ffluency *Fast verbal fluency generator*

Description

Generates verbal fluency data using a switcher-random walk process.

Usage

```
one_ffluency(adj_list, n, pjump = 0, type = 0L)
```

Arguments

adj_list	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
n	integer specifying the maximum number of productions. Function may return fewer than n.
pjump	numeric specifying the probability of a jump.
type	integer controlling network start and jump nodes. For type = 0 the process selects the start node and any jump nodes proportional to their degree. For type = 1 the process selects a random node to serve both as the start node and the jump node. For type = 2 the process selects the start and any jump nodes uniformly at random.

Details

Function produces verbal fluency data via a switcher random walk process that traverses the network by selecting a neighbor with probability `1-pjump` or jumps to a random place in the network with probability `pjump`. Where the random walk process enters the network and where it jumps to is further controlled by `type`. Neighbors are always selected uniformly.

In contrast to fluency, this function does not check at every step whether the sampled neighbor is already in the list of productions. Instead, `ffluency` simply returns the list of unique productions. This means that if repetitions occur `ffluency` will produce sequences of length `min(n * 3 - k, n)` where k is the number of repeptitions.

Value

Integer vector containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# create verbal fluency sequences
one_ffluency(get_adjlist(network),  10)

# create verbal fluency sequence
# with high jump probability
one_ffluency(get_adjlist(network), 10, pjump = .5)
```

---

one_fluency *Verbal fluency generator*

Description

Generates verbal fluency data using a switcher-random walk process.

Usage

```
one_fluency(adj_list, n, pjump = 0, type = 0L)
```

Arguments

adj_list	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
n	integer specifying the number of unique productions.
pjump	numeric specifying the probability of a jump.
type	integer controlling network start and jump nodes. For type = 0 the process selects the start node and any jump nodes proportional to their degree. For type = 1 the process selects a random node to serve both as the start node and the jump node. For type = 2 the process selects the start and any jump nodes uniformly at random.

Details

Function produces verbal fluency data via a switcher random walk process that traverses the network by selecting a neighbor with probability 1-pjump or jumps to a random place in the network with probability pjump. Where the random walk process enters the network and where it jumps to is further controlled by type. Neighbors are always selected uniformly.

Value

Integer vector containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# create verbal fluency sequence
one_fluency(get_adjlist(network), 10)

# create verbal fluency sequence
# with high jump probability
one_fluency(get_adjlist(network), 10, pjump = .5)
```

`one_fluency_steps` *Verbal fluency step counter*

Description

Generates verbal fluency data using a switcher-random walk process and counts the number of steps required to produce `n` unique responses.

Usage

```
one_fluency_steps(adj_list, n, pjump = 0, type = 0L)
```

Arguments

`adj_list`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`n`	integer specifying the number of productions.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.

Details

Function produces verbal fluency data via a switcher random walk process that traverses the network by selecting a neighbor with probability `1-pjump` or jumps to a random place in the network with probability `pjump`. Where the random walk process enters the network and where it jumps to is further controlled by `type`. Neighbors are always selected uniformly.

In contrast to fluency and `ffluency`, returns the number of steps required to produce a sequence of unique productions, rather than the productions itself.

Value

Integer vector containing the indices of the fluency productions. Indices refer to the row of the item in the original adjacency matrix. See get_adjlist.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# count number of steps needed to create sequence
one_fluency_steps(get_adjlist(network), 10)

# count number of steps needed to create sequence
# with high jump probability
one_fluency_steps(get_adjlist(network), 10, pjump = .5)
```

---

one_search *Search network using switcher-random walk process*

Description

Traverses a network using a switcher-random walk process and records the number of steps required to get from node `start` to node `observe`.

Usage

```
one_search(adj_list, start, observe, nmax = 1000L, pjump = 0,
  type = 0L)
```

Arguments

`adj_list`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`start`	integer vector of length 1 specifying the index of the start node.
`observe`	integer vector of length 1 or larger specifying the target end nodes.
`nmax`	integer specifying the maximum number of steps before search terminates.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.

Details

If a node specified in `observe` has never been visited then the function returns `nmax` for that node.

Value

Numeric, 3 column matrix containing in each row the start node, the end node, and the (minimum) number of steps it took to reach the end node from the start node.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# observe number of steps from node 2
# to nodes 3, 4, and 5
one_search(get_adjlist(network), 2, c(3, 4, 5))

# observe number of steps from node 2 to nodes 3, 4, and 5
# with high jump probability
one_search(get_adjlist(network), start = 2, observe = c(3, 4, 5), pjump = .5)
```

---

restore_names *Restore names of memnet objects*

Description

Function replaces the index vectors created by get_adjlist fluency, or search_rw by their original names.

Usage

```
restore_names(dat, names)
```

Arguments

dat	lists of indices that correspond to positions in a name vector or matrix with two initial columns containing node indices.
names	character vector giving the names.

Value

lists of character vectors.

Examples

```
# get fluency data
data(animal_fluency)

# edge list of fluency graph
edge_list = threshold_graph(animal_fluency)

# extract adjlist from community network
adjlist = edg_to_adjlist(edge_list)

f = fluency(adjlist, c(10, 14, 16, 18))
restore_names(f, get_names(edge_list))
```

---

rw_graph *Create random walk graph*

Description

Create a random walk graph from verbal fluency data that includes edges for words that occur within a window size of 1.

Usage

```
rw_graph(dat)
```

Arguments

dat	list of character vectors containing the fluency productions.

Value

A matrix

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Zemla, J. C., & Austerweil, J. L. (2018). Estimating semantic networks of groups and individuals from fluency data. Computational Brain & Behavior, 1-23.

Examples

```
# get animal fluency data
data(animal_fluency)

# infer influence network
inferred_network = rw_graph(animal_fluency)

# Simulate -----

# generate watts strogatz graph
network = grow_ws(n = 200, k = 10, p = .5)

# generate fluency data
# sets string equal TRUE as community_graph expects mode character
fluency_data = fluency(get_adjlist(network), rep(10, 100), string = TRUE)

# infer fluency network
inferred_network = rw_graph(fluency_data)
```

`search_rw` *Search network using switcher-random walk process*

Description

Traverses a network using a switcher-random walk process and records the number of steps required to get from node `start` to node `observe`.

Usage

```
search_rw(adjlist, start, observe, nmax = 1000L, pjump = 0,
  type = 0L)
```

Arguments

`adjlist`	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
`start`	integer vector of length 1 or larger specifying the index of the start node.
`observe`	integer vector of length 1 or larger specifying the target end nodes.
`nmax`	integer specifying the maximum number of steps before search terminates.
`pjump`	numeric specifying the probability of a jump.
`type`	integer controlling network start and jump nodes. For `type = 0` the process selects the start node and any jump nodes proportional to their degree. For `type = 1` the process selects a random node to serve both as the start node and the jump node. For `type = 2` the process selects the start and any jump nodes uniformly at random.

Details

If a node specified in `observe` has never been visited then the function returns `nmax` for that node.

Value

Numeric, 3 column matrix containing in each row the start node, the end node, and the (minimum) number of steps it took to reach the end node from the start node.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# observe number of steps from node 2 and 6
# to nodes 3, 4, and 5
search_rw(get_adjlist(network), c(2, 6), c(3, 4, 5))

# observe number of steps from node 2 and 6 to nodes 3, 4, and 5
# with high jump probability
search_rw(get_adjlist(network), start = c(2, 6), observe = c(3, 4, 5), pjump = .5)
```

`search_rw_mean` *Search network repeatedly using switcher-random walk process*

Description

Traverses a network using a switcher-random walk process repeatedly, records the earliest visit to nodes of interest and averages the result.

Usage

```
search_rw_mean(adjlist, start, observe, nmax = 1000L, pjump = 0,
  type = 0L, nrep = 100L)
```

Arguments

adjlist	a list containing row indices of nodes adjacent node to the ith node as created by get_adjlist.
start	integer vector of length 1 or larger specifying the index of the start node.
observe	integer vector specifying the nodes whose first visits should be recorded.
nmax	integer specifying the maximum number of steps.
pjump	numeric specifying the probability of a jump.
type	integer controlling network start and jump nodes. For type = 0 the process selects the start node and any jump nodes proportional to their degree. For type = 1 the process selects a random node to serve both as the start node and the jump node. For type = 2 the process selects the start and any jump nodes uniformly at random.
nrep	integer specifying the number of iterations across which aggregates are computed.

Details

Beginning at a given start node, function traverses a network using switcher random walk and records for each of a list of nodes of interest the index at which the respective nodes have been visited first.

If a node specified in observe has never been visited then the function returns nmax for that node.

Value

Numeric, 3 column matrix containing in each row the start node, the end node, and the (minimum) number of steps it took to reach the end node from the start node.

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Goni, J., Martincorena, I., Corominas-Murtra, B., Arrondo, G., Ardanza- Trevijano, S., & Villoslada, P. (2010). Switcher-random-walks: A cognitive- inspired mechanism for network exploration. International Journal of Bifurcation and Chaos, 20(03), 913-922.

Examples

```
# generate watts strogatz graph
network = grow_ws(n = 100, k = 10)

# determine mean number of steps from node 2 and 6
# to nodes 3, 4, and 5
search_rw_mean(get_adjlist(network), c(2, 6), c(3, 4, 5))

# determine mean number of steps from node 2 and 6 to nodes 3, 4, and 5
# with high jump probability
search_rw_mean(get_adjlist(network), start = c(2, 6), observe = c(3, 4, 5), pjump = .5)
```

threshold_graph *Create threshold graph*

Description

Create a graph from verbal fluency data by adding edges for words that occur adjacent to each other more frequently than min_cooc.

Usage

```
threshold_graph(dat, min_cooc = 2L)
```

Arguments

dat	list of character vectors containing the fluency productions.
min_cooc	integer specifying the minimum number of times two words are required to coocur one step apart from each other for an edge to connect those words.

Value

A matrix

References

Wulff, D. U., Hills, T., & Mata, R. (2018, October 29). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Zemla, J. C., & Austerweil, J. L. (2018). Estimating semantic networks of groups and individuals from fluency data. Computational Brain & Behavior, 1-23.

Examples

```
# get animal fluency data
data(animal_fluency)

# infer influence network
inferred_network = threshold_graph(animal_fluency)

# Simulate -----

# generate watts strogatz graph
network = grow_ws(n = 200, k = 10, p = .5)

# generate fluency data
# sets string equal TRUE as community_graph expects mode character
fluency_data = fluency(get_adjlist(network), rep(10, 100), string = TRUE)

# infer fluency network
```

```
inferred_network = threshold_graph(fluency_data)
```

Index