Package: cograph (via r-universe)

October 8, 2026

Title Analysis and Visualization of Complex Networks

Version 2.7.6

Author Mohammed Saqr [aut, cph], Sonsoles López-Pernas [aut, cre, cph]

Maintainer Sonsoles López-Pernas <sonsoles.lopez@uef.fi>

Description Provides tools for the analysis, visualization, and manipulation of dynamical, social (Saqr et al. (2024) <doi:10.1007/978-3-031-54464-4_10>) and complex networks (Saqr et al. (2025) <doi:10.1145/3706468.3706513>). The package supports multiple network formats and offers flexible tools for heterogeneous, multi-layer, and hierarchical network analysis with simple syntax and extensive toolset.

License MIT + file LICENSE

URL https://sonsoles.me/cograph/, https://github.com/sonsoleslp/cograph

BugReports https://github.com/sonsoleslp/cograph/issues

Depends R (>= 4.1.0)

Imports ggplot2 (>= 3.4.0), grDevices, grid, parallel, R6, stats, utils

Suggests Matrix, backbone, brainGraph, centiserve, colorspace, digest, dplyr, gifski, gridExtra, grImport2, igraph, influenceR, jsonlite, keyplayer, knitr, Nestimate, netrankr, network, qgraph, RColorBrewer, reticulate, rmarkdown, rsvg, sna, testthat (>= 3.0.0), tidygraph, tna, tnet, viridisLite

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData true

Repository https://sonsoleslp.r-universe.dev

Date/Publication 2026-10-08 05:24:23 UTC

RemoteUrl https://github.com/sonsoleslp/cograph

RemoteRef HEAD

RemoteSha f23abcfcdfae840db888566ae0bea0530d196349

Contents

abbrev_label . . . 9
add_edges . . . 10
add_nodes . . . 11
aggregate_layers . . . 12
aggregate_weights . . . 13
as_cograph . . . 13
as_mcml . . . 15
as_tna . . . 17
assortativity . . . 18
assortativity_attribute . . . 20
binarize . . . 22
bind_networks . . . 23
centrality . . . 24
centrality_access_information . . . 41
centrality_adaptive_leaderrank . . . 42
centrality_alpha . . . 43
centrality_authority . . . 44
centrality_average_distance . . . 45
centrality_barycenter . . . 46
centrality_beta_measure . . . 47
centrality_betweenness . . . 48
centrality_bottleneck . . . 49
centrality_bridging . . . 50
centrality_bridging_capital . . . 51
centrality_brokerage_coordinator . . . 52
centrality_brokerage_gatekeeper . . . 53
centrality_brokerage_itinerant . . . 54
centrality_brokerage_liaison . . . 55
centrality_brokerage_representative . . . 56
centrality_cda . . . 57
centrality_centroid . . . 58
centrality_closeness . . . 59
centrality_closeness_vitality . . . 61
centrality_clusterrank . . . 62
centrality_coleman_theil . . . 63
centrality_communicability . . . 64
centrality_communicability_betweenness . . . 65
centrality_community_based . . . 66

centrality_community_hub_bridge . . . 67
centrality_constraint . . . 68
centrality_controlrank . . . 69
centrality_coreness . . . 70
centrality_cross_clique . . . 71
centrality_current_flow_betweenness . . . 72
centrality_current_flow_closeness . . . 73
centrality_dangalchev . . . 74
centrality_decay . . . 75
centrality_degree . . . 76
centrality_degree_discount . . . 77
centrality_diffusion . . . 78
centrality_diffusion_centrality . . . 79
centrality_dil . . . 80
centrality_distance_entropy . . . 81
centrality_diversity . . . 82
centrality_dkgm . . . 83
centrality_dmnc . . . 84
centrality_dynamical_importance . . . 85
centrality_dynamics_sensitive . . . 87
centrality_eccentricity . . . 88
centrality_effective_size . . . 89
centrality_ehcc . . . 90
centrality_eigenvector . . . 91
centrality_entropy . . . 92
centrality_entropy_variation . . . 93
centrality_exogenous . . . 94
centrality_expected . . . 95
centrality_expected_force . . . 96
centrality_expected_influence_1 . . . 97
centrality_expected_influence_2 . . . 98
centrality_extended_coreness . . . 99
centrality_extended_gravity . . . 100
centrality_extended_local_bridging . . . 101
centrality_extended_mixed_gravity . . . 102
centrality_flow_betweenness . . . 103
centrality_gateway . . . 104
centrality_generalized_closeness . . . 105
centrality_gilschmidt . . . 106
centrality_global_structure . . . 107
centrality_graph_regularization . . . 108
centrality_gravity . . . 109
centrality_harary . . . 110
centrality_harmonic . . . 111
centrality_hcc . . . 112
centrality_heatmap . . . 113
centrality_hubbell . . . 115
centrality_hybrid_global_structure . . . 116

centrality_iec . . . 117
centrality_iira . . . 118
centrality_improved_closeness . . . 119
centrality_improved_global_structure . . . 121
centrality_information . . . 122
centrality_integration . . . 123
centrality_ira . . . 124
centrality_katz . . . 125
centrality_ked . . . 126
centrality_kreach . . . 127
centrality_lac . . . 128
centrality_laplacian . . . 129
centrality_leaderrank . . . 130
centrality_length_scaled_betweenness . . . 131
centrality_leverage . . . 132
centrality_lhc . . . 133
centrality_lin . . . 134
centrality_linerank . . . 135
centrality_lnc . . . 137
centrality_load . . . 138
centrality_lobby . . . 139
centrality_local_bridging . . . 140
centrality_local_dimension . . . 141
centrality_local_dimension_fixed . . . 142
centrality_local_efficiency . . . 143
centrality_local_information_dimension . . . 145
centrality_localized_bridging . . . 146
centrality_malatya . . . 147
centrality_map_equation . . . 148
centrality_markov . . . 149
centrality_mcc . . . 150
centrality_mcgm . . . 151
centrality_mixed_gravity . . . 152
centrality_mnc . . . 153
centrality_modified_expected_force . . . 154
centrality_modularity_vitality . . . 155
centrality_ncvoterank . . . 156
centrality_neighbor_distance . . . 157
centrality_neighborhood_connectivity . . . 159
centrality_ninl . . . 160
centrality_node_contraction . . . 161
centrality_pagerank . . . 162
centrality_pairwisedis . . . 163
centrality_participation . . . 164
centrality_percolation . . . 165
centrality_power . . . 166
centrality_prestige_domain . . . 167
centrality_prestige_domain_proximity . . . 168

centrality_proximal_betweenness . . . 169
centrality_radiality . . . 170
centrality_random_walk . . . 171
centrality_random_walk_decay . . . 172
centrality_reaching_local . . . 173
centrality_relative_entropy . . . 174
centrality_residual_closeness . . . 176
centrality_resistance_curvature . . . 177
centrality_rsp_betweenness . . . 178
centrality_rumor . . . 179
centrality_s_shell . . . 180
centrality_salsa . . . 181
centrality_semilocal . . . 182
centrality_shapley_game1 . . . 183
centrality_spectralrank . . . 184
centrality_strength . . . 185
centrality_stress . . . 186
centrality_subgraph . . . 187
centrality_topological_coefficient . . . 188
centrality_transitivity . . . 189
centrality_truss . . . 190
centrality_trust_pagerank . . . 192
centrality_two_way_rw . . . 193
centrality_volume . . . 194
centrality_voterank . . . 195
centrality_weighted_kshell . . . 196
centrality_weighted_leaderrank . . . 197
centrality_wiener . . . 198
centrality_within_module_z . . . 199
centrality_wvoterank . . . 200
centrality_x_degree . . . 202
centralization . . . 203
cluster_quality . . . 204
cluster_significance . . . 205
cograph . . . 207
CographLayout . . . 208
CographNetwork . . . 210
CographTheme . . . 215
color_communities . . . 217
communities . . . 218
community_consensus . . . 220
community_edge_betweenness . . . 222
community_fast_greedy . . . 223
community_fluid . . . 224
community_infomap . . . 225
community_label_propagation . . . 227
community_leading_eigenvector . . . 228
community_leiden . . . 229

community_louvain . . . 231
community_optimal . . . 232
community_sizes . . . 233
community_spinglass . . . 233
community_walktrap . . . 235
compare_communities . . . 236
complement_network . . . 237
contract_nodes . . . 238
core_periphery . . . 239
csum . . . 241
degree_distribution . . . 244
detect_communities . . . 246
disparity_filter . . . 247
dispersion . . . 248
dyad_census . . . 249
edge_centrality . . . 251
edge_reciprocity . . . 252
ego_networks . . . 253
estrada_index . . . 255
extract_motifs . . . 256
extract_triads . . . 259
filter_edges . . . 260
filter_nodes . . . 261
fit_degree_distribution . . . 263
from_qgraph . . . 265
from_tna . . . 267
get_edge_list . . . 268
get_nodes . . . 269
ggplot_robustness . . . 272
group_centrality . . . 273
hai_datasets . . . 275
invert_weights . . . 276
is_bipartite . . . 277
is_tna_network . . . 278
k_shortest_paths . . . 278
layer_degree_correlation . . . 280
layer_similarity . . . 280
layer_similarity_matrix . . . 281
layout_circle . . . 282
layout_groups . . . 283
layout_oval . . . 284
layout_registry . . . 285
layout_saqr . . . 286
layout_spring . . . 287
layout_target . . . 288
list_centralities . . . 289
membership . . . 290
motif_census . . . 291

motifs . . . 292
mutate_edges . . . 296
mutate_nodes . . . 297
n_communities . . . 298
neighborhood_overlap . . . 298
network_bridges . . . 299
network_clique_size . . . 300
network_cut_vertices . . . 300
network_girth . . . 301
network_global_efficiency . . . 302
network_local_efficiency . . . 303
network_radius . . . 304
network_rich_club . . . 304
network_small_world . . . 305
network_summary . . . 306
network_vertex_connectivity . . . 308
network_wrangling . . . 309
normalize_weights . . . 311
overlay_communities . . . 312
palettes . . . 313
panel_layout . . . 314
plot-results . . . 315
plot_alluvial . . . 324
plot_bootstrap_forest . . . 327
plot_centrality . . . 331
plot_centrality_compare . . . 333
plot_centrality_distribution . . . 335
plot_centrality_heatmap . . . 336
plot_chord . . . 337
plot_compare . . . 340
plot_comparison_heatmap . . . 340
plot_degree_correlation . . . 342
plot_difference . . . 343
plot_edge_diff_forest . . . 345
plot_edge_weights . . . 347
plot_heatmap . . . 348
plot_htna . . . 351
plot_mcml . . . 355
plot_mixed_network . . . 362
plot_ml_heatmap . . . 364
plot_mlna . . . 366
plot_mtna . . . 369
plot_network_evolution . . . 372
plot_robustness . . . 374
plot_simplicial . . . 375
plot_temporal . . . 378
plot_tna . . . 380
plot_trajectories . . . 383

plot_transitions . . . 386
project_bipartite . . . 391
reaching_global . . . 392
regulation_net . . . 393
remove_edges . . . 394
remove_isolates . . . 395
remove_nodes . . . 396
rename_nodes . . . 396
render-ggplot . . . 397
render-grid . . . 398
reorder_nodes . . . 398
reverse_edges . . . 399
rich_club . . . 400
rich_club_local . . . 402
robustness . . . 403
robustness_auc . . . 405
robustness_summary . . . 406
select_bridges . . . 406
select_component . . . 407
select_edges . . . 408
select_edges_between . . . 410
select_edges_involving . . . 411
select_k_core . . . 412
select_neighbors . . . 413
select_nodes . . . 414
select_top . . . 416
select_top_edges . . . 417
shapes . . . 418
shortest_paths . . . 419
simmelian_strength . . . 420
simplify . . . 421
sn_edges . . . 423
sn_ggplot . . . 427
sn_layout . . . 428
sn_nodes . . . 429
sn_palette . . . 433
sn_save . . . 434
sn_save_ggplot . . . 434
sn_theme . . . 435
soplot . . . 436
spanning_tree . . . 444
split_components . . . 446
splot.net_bootstrap . . . 447
student_interactions . . . 462
subgraphs . . . 463
summarize_clusters . . . 464
summarize_network . . . 465
supra_adjacency . . . 467

supra_interlayer 468
supra_layer 469
symmetrize 469
themes 471
threshold_edges 472
to_data_frame 473
to_directed 474
to_igraph 475
to_matrix 476
to_network 477
to_undirected 477
triad_census 478
trophic_incoherence 479
verify_with_igraph 480
vulnerability 481

Index **484**

`abbrev_label` *Abbreviate Labels*

Description

Abbreviates labels to a maximum length, adding ellipsis if truncated.

Usage

```
abbrev_label(label, abbrev = NULL, n_labels = NULL)

label_abbrev(label, abbrev = NULL, n_labels = NULL)
```

Arguments

`label`	Character vector of labels to abbreviate.
`abbrev`	Abbreviation control: • NULL: No abbreviation (return labels unchanged) • Integer: Maximum character length (truncate + ellipsis) • "auto": Length chosen from the label count. Up to 5 labels are kept whole, and up to 8, 12, 20 or more labels are cut to 15, 10, 6 or 4 characters.
`n_labels`	Number of labels (used for "auto" mode). If NULL, uses length(label).

Value

Character vector of (possibly abbreviated) labels.

Examples

```
abbrev_label(colnames(regulation_net), abbrev = 4)
```

add_edges *Add Edges to a Network*

Description

Appends edges between existing nodes. An endpoint that is not a node of the network raises a `cograph_bad_selection` error.

Usage

```
add_edges(x, from, to, weight = 1, ..., keep_format = FALSE, directed = NULL)
```

Arguments

`x`	Network input.
`from`	Source nodes, by label or index.
`to`	Target nodes, by label or index. The same length as `from`.
`weight`	Numeric weight for the new edges, length 1 or `length(from)`. Default 1.
`...`	Named vectors of extra edge attributes, length 1 or `length(from)`.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with the new edges, or the input format when `keep_format = TRUE`. An edge that already exists has its weight replaced, and a `cograph_edges_replaced` warning says how many.

Note

When the igraph package is attached it masks this function with `igraph::add_edges()`, which takes an igraph object. Use `cograph::add_edges()` to be explicit.

See Also

`remove_edges`, `add_nodes`, `bind_networks`

Examples

```
add_edges(regulation_net, from = "Share", to = "Explore", weight = 0.5)
```

add_nodes *Add Nodes to a Network*

Description

Appends new nodes, identified by label, to a network. A label that already exists raises a `cograph_bad_selection` error.

Usage

```
add_nodes(x, labels, ..., keep_format = FALSE, directed = NULL)
```

Arguments

`x`	Network input.
`labels`	Character vector of labels for the new nodes.
`...`	Named vectors of node attributes for the new nodes, each of length 1 (recycled) or `length(labels)`. Columns the network does not already have are created and filled with `NA` for the existing nodes.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with the new nodes appended (isolated until edges are added), or the input format when `keep_format = TRUE`.

See Also

`remove_nodes`, `add_edges`, `mutate_nodes`

Examples

```
add_nodes(regulation_net, labels = "Revise")
```

`aggregate_layers` *Aggregate Layers*

Description

Combines multiple network layers into a single network.

Usage

```
aggregate_layers(
  layers,
  method = c("sum", "mean", "max", "min", "union", "intersection"),
  weights = NULL
)

lagg(
  layers,
  method = c("sum", "mean", "max", "min", "union", "intersection"),
  weights = NULL
)
```

Arguments

`layers`	List of adjacency matrices of the same dimensions
`method`	Aggregation: "sum" (default), "mean", "max", "min", "union" or "intersection". `"union"` and `"intersection"` return a binary matrix of the cells with a positive weight in any or in every layer.
`weights`	Optional numeric vector of layer weights, one per layer, used only by `method = "sum"` to compute a weighted sum.

Value

The aggregated adjacency matrix, with the dimnames of the first layer. A list with a single layer is aggregated the same way, so `"union"` and `"intersection"` binarize it and `"sum"` multiplies it by its layer weight.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
aggregate_layers(layers, method = "mean")
```

aggregate_weights *Aggregate Edge Weights*

Description

Aggregates a vector of edge weights into a single value. The method names follow those of igraph's `edge.attr.comb` argument.

Usage

```
aggregate_weights(w, method = "sum", n_possible = NULL)

wagg(w, method = "sum", n_possible = NULL)
```

Arguments

`w`	Numeric vector of edge weights. `NA` and zero entries are dropped before aggregation.
`method`	Aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", "geomean". Default "sum". Any other value is an error. `"geomean"` uses only the positive weights.
`n_possible`	Number of possible edges (used only by `method = "density"`; when NULL or not positive, the number of surviving weights is used as the denominator instead).

Value

A single numeric value, or 0 when no non-zero, non-NA weight remains.

Examples

```
aggregate_weights(regulation_net, method = "mean")
```

as_cograph *Convert to Cograph Network*

Description

`as_cograph()` creates a `cograph_network` object from a matrix, an edge list, or a network object of another package. `to_cograph()` is an alias. The object is a named list that every cograph function accepts.

Usage

```
as_cograph(x, directed = NULL, simplify = FALSE, ...)

to_cograph(x, directed = NULL, ...)

## S3 method for class 'cograph_network'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  what = c("edges", "nodes")
)
```

Arguments

x	Network input. One of a square numeric weight matrix, a data frame edge list, an igraph object, a statnet network object, a qgraph object, a tna object, or an existing cograph_network, which is returned as it is. In an edge list the endpoint columns are found by name, ignoring case (from, source, src, v1, node1 or i, and to, target, tgt, v2, node2 or j), and otherwise the first two columns are used. An optional weight column is found by the names weight, w, value or strength. For as.data.frame(), a cograph_network.
directed	Logical. Forces a directed or undirected interpretation. NULL (default) detects it from the input.
simplify	Logical or character. If FALSE (default), every transition from tna sequence data is a separate edge. If TRUE (equivalent to "sum") or one of "sum", "mean", "max", "min", duplicate transitions are aggregated with that function. Other inputs are not affected.
...	Passed from to_cograph() to as_cograph(); otherwise unused.
row.names	NULL or a character vector of row names, as for as.data.frame.
optional	Logical, as for as.data.frame. It is ignored, and the column names are always the documented ones.
what	Which table as.data.frame() returns, "edges" (default) or "nodes".

Details

A cograph_network prints a short description of its nodes, edges and source, summary() reports counts and edge weight statistics, plot() plots it with sn_render, and as.data.frame() returns its edge or node table. The accessor and setter functions are documented in get_nodes.

Producer packages may attach plotting hints under meta$splot. The recognized fields are renderer (the cograph renderer to use), weight (the edge column or matrix plotted as weight), and defaults (a named list of renderer arguments). Entries in defaults are defaults only, and arguments passed to splot override them. renderer and weight define which view is plotted and are not overridden by plot arguments.

Value

`as_cograph()` and `to_cograph()` return a `cograph_network` object, a named list with components:

`nodes` Data frame with `id`, `label`, and optional layout or metadata columns.

`edges` Data frame with integer `from` and `to` columns (row numbers into `nodes`), `weight`, and extra columns such as `session` and `time` for tna input. Repeated rows of an edge list are kept as separate edges.

`directed` Logical. Whether the network is directed.

`weights` The n x n weight matrix for matrix and tna input, or `NULL` (for example for an edge list).

`data` The original estimation data (sequence data, edge list), or `NULL`.

`meta` Metadata list with `source` (input type), a `tna` entry for tna input (type, group name and group index) and, optionally, `splot` (rendering hints read by `splot`).

`node_groups` Optional data frame of node groupings.

`as.data.frame()` returns a base data frame. For `what = "edges"` it has one row per edge with columns `from` and `to` (node labels), `weight`, and any extra edge columns the network carries, such as `session` or columns computed by `mutate_edges`. For `what = "nodes"` it has one row per node with the node table columns (`id`, `label`, layout coordinates and any custom columns). `to_df` returns only `from`, `to` and `weight`.

See Also

`get_nodes`, `splot`, `to_df`

Examples

```
net <- as_cograph(regulation_net)
as.data.frame(net, what = "edges")
```

---

as_mcml *Convert to mcml*

Description

Converts an object to the `mcml` class, a representation of a multi-cluster network that does not depend on the tna package.

Usage

```
as_mcml(x, ...)

## S3 method for class 'cluster_summary'
as_mcml(x, ...)

## S3 method for class 'group_tna'
```

```
as_mcml(x, clusters = NULL, method = "sum", type = "tna", directed = TRUE, ...)

## S3 method for class 'mcml'
as_mcml(x, ...)

## Default S3 method:
as_mcml(x, ...)
```

Arguments

x	A cluster_summary, group_tna or mcml object. Any other input is an error; summarize_clusters builds an mcml object from raw data.
...	Additional arguments passed to methods.
clusters	Integer or character vector of row-to-group assignments. Required when the group_tna has the same labels across all groups (row-level clustering from tna::group_model(cluster_data(...))).
method	Aggregation method for macro weights (default "sum").
type	Transition type (default "tna").
directed	Logical; whether the network is directed (default TRUE).

Value

An mcml object with components macro, clusters, cluster_members, and meta.

An mcml object.

An mcml object. When clusters is provided, macro$data contains the cluster assignments and macro$weights is NULL.

The input mcml object unchanged.

See Also

summarize_clusters, as_tna

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
as_mcml(csum(regulation_net, clusters = clusters, type = "tna"))
```

as_tna *Convert cluster_summary to tna Objects*

Description

Converts a `cluster_summary` or `mcml` object to tna models that can be used with the functions of the tna package. The result holds a macro (cluster-level) model and one model of the internal transitions of each cluster, as a flat `group_tna` object.

Usage

```
as_tna(x)

## S3 method for class 'cluster_summary'
as_tna(x)

## S3 method for class 'mcml'
as_tna(x)

## Default S3 method:
as_tna(x)
```

Arguments

`x`	A `cluster_summary` object created by `csum`, or an `mcml` object created by `summarize_clusters`. A tna object is returned unchanged, and any other input is an error. The weights are passed to `tna::tna()`, which row-normalizes them, so a summary computed with `type = "raw"` gives the same transition probabilities as one computed with `type = "tna"`.

Details

Requirements:

The tna package must be installed. Without it, the function raises an error.

Excluded Clusters:

A per-cluster tna cannot be created when:

- The cluster has only 1 node (no internal transitions possible)
- Some nodes in the cluster have no outgoing edges (row sums to 0)

These clusters are left out of the result with a warning of class `cograph_cluster_dropped`, which names each cluster and the nodes that have no transition within it. The macro (cluster-level) model still includes all clusters.

Value

A group_tna object, a flat named list of tna objects. The first element is named "macro" and holds the cluster-level transitions. The remaining elements are named by cluster and hold the internal transitions of each cluster.

macro A tna object of cluster-level transitions, with weights (k x k transition matrix), inits (initial distribution) and labels (cluster names).

<cluster_name> One tna object per cluster, with weights (n_i x n_i matrix), inits (initial distribution) and labels (node labels). A cluster that cannot become a tna model is left out with a warning (see Excluded Clusters).

A group_tna object (flat list of tna objects: macro + per-cluster).

A group_tna object (flat list of tna objects: macro + per-cluster).

A tna object constructed from the input.

See Also

csum to create the input object, plot_mcml for visualization without conversion, tna::tna for the underlying tna constructor

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
as_tna(csum(regulation_net, clusters = clusters, type = "tna"))
```

---

assortativity *Degree Assortativity Coefficient*

Description

Computes the degree assortativity coefficient, measuring the tendency of nodes to connect to other nodes with similar degree. Positive values indicate assortative mixing (high-degree nodes connect to high-degree nodes), negative values indicate disassortative mixing.

Usage

```
assortativity(x, directed = NULL, type = NULL, digits = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
type	Character string specifying which degree correlation to compute, or NULL (default) to choose automatically: "out-in" for directed networks and "degree" for undirected ones. For a directed network the accepted values are "out-in", "in-in", "out-out" and "in-out"; for an undirected network the only accepted value is "degree". Any other value raises an error.
digits	Integer or NULL. Round result to this many decimal places. Default NULL (no rounding).
...	Not used. Any argument supplied here raises an "unused argument" error.

Details

The degree assortativity coefficient is defined as the Pearson correlation coefficient between the degrees of nodes at either end of each edge (Newman 2002):

$$r = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}$$

where e_{jk} is the fraction of edges connecting degree-j to degree-k vertices, q_k is the excess degree distribution, and σ_q^2 its variance.

Because the Pearson correlation is invariant to subtracting a constant, the implementation computes the correlation of the raw degrees at the two ends of each edge, counting every undirected edge in both orientations. This is numerically identical to the formula above. Degrees are unweighted counts of edges, so edge weights do not enter the coefficient.

For directed networks, the coefficient is the Pearson correlation between the source-end and target-end degrees over each edge in its stored orientation, with the degree mode at each end chosen by type (Foster et al. 2010).

The coefficient is NA when the network has no edges or when either degree vector has zero variance.

Value

An object of class "cograph_assortativity" with components:

coefficient Numeric scalar: the assortativity coefficient in $[-1, 1]$.

type Character: the degree type used.

directed Logical: whether the network was treated as directed.

n_nodes Numeric: number of nodes.

n_edges Numeric: number of edges.

network The original input network.

References

Newman, M.E.J. (2002). Assortative mixing in networks. *Physical Review Letters*, 89(20), 208701. doi:10.1103/PhysRevLett.89.208701

Foster, J.G., Foster, D.V., Grassberger, P., & Paczuski, M. (2010). Edge direction and the structure of networks. *PNAS*, 107(24), 10815-10820. doi:10.1073/pnas.0912671107

See Also

`assortativity_attribute`, `centrality`, `network_summary`

Examples

```
cograph::assortativity(regulation_net)
```

`assortativity_attribute` *Attribute Assortativity (Homophily)*

Description

Computes assortativity with respect to a node attribute, measuring the tendency of nodes to connect to others with similar attribute values. For categorical attributes, this computes the modularity-based nominal assortativity. For numeric attributes, this computes the Pearson correlation between attribute values at edge endpoints.

Usage

```
assortativity_attribute(x, values, directed = NULL, digits = NULL, ...)

homophily(x, values, directed = NULL, digits = NULL, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`values`	Named vector of attribute values whose names cover every node name, or an unnamed vector of length equal to the number of nodes, in node order. A numeric vector is treated as scalar and any other vector as nominal.
`directed`	Logical or NULL. If NULL (default), auto-detect.
`digits`	Integer or NULL. Round the coefficient to this many decimal places. Default NULL (no rounding).
`...`	Not used. Any argument supplied here raises an `"unused argument"` error.

Details

For categorical (nominal) attributes, the coefficient is:

$$r = \frac{\text{tr}(\mathbf{e}) - \|\mathbf{e}^2\|}{1 - \|\mathbf{e}^2\|}$$

where $\mathbf{e}$ is the mixing matrix with e_{ij} = fraction of edges connecting type i to type j.

For numeric (scalar) attributes, the coefficient is the Pearson correlation between attribute values at edge endpoints (computed over both orientations of every edge when the network is undirected). Any non-numeric `values` vector (character or factor) is treated as nominal.

The coefficient is `NA` when the network has no edges, when a nominal attribute has a single category, or when either value vector has zero variance.

Value

An object of class `"cograph_assortativity"` with components:

coefficient Numeric scalar: assortativity coefficient.

type Character: `"nominal"` or `"scalar"`.

directed Logical: whether the network was treated as directed.

n_nodes Numeric: number of nodes.

n_edges Numeric: number of edges.

attribute_values The attribute values used, named by node and in node order.

network The original input network.

References

Newman, M.E.J. (2003). Mixing patterns in networks. *Physical Review E*, 67(2), 026126. doi:10.1103/PhysRevE.67.026126

See Also

`assortativity`, `detect_communities`

Examples

```
cograph::assortativity_attribute(regulation_net, values = rep(c("self", "social"), each = 5))
```

`binarize` *Binarize Edge Weights*

Description

Replaces every surviving weight with 1 and drops edges whose weight is at or below the threshold. The operation corresponds to `sna::event2dichot()` with an absolute threshold.

Usage

```
binarize(
  x,
  threshold = 0,
  absolute = TRUE,
  signed = FALSE,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`threshold`	Numeric. Edges whose weight exceeds this value are kept and set to 1. Default 0, which keeps every existing edge.
`absolute`	Logical. Compare `abs(weight)`. Default TRUE, so a correlation network keeps its strong negative edges.
`signed`	Logical. If TRUE, negative edges become `-1`, which preserves the sign of the association. Default FALSE.
`keep_isolates`	Logical. Keep nodes that end up with no edges? Default TRUE.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` whose weights are all `1` (or, when `signed = TRUE`, `1` for a positive edge and `-1` for a negative one), or the input format when `keep_format = TRUE`. Nodes left without edges are kept and reported in a `cograph_isolates_created` warning, unless `keep_isolates = FALSE`.

References

Butts, C. T. (2008). Social network analysis with sna. *Journal of Statistical Software*, 24(6), 1–51.

See Also

`threshold_edges`, `normalize_weights`

Examples

```
binarize(regulation_net, threshold = 0.1)
```

`bind_networks` *Combine Two Networks*

Description

Aligns two networks on node labels and combines their edges.

Usage

```
bind_networks(
  x,
  y,
  method = c("union", "intersection", "difference"),
  weight = c("sum", "mean", "max", "min", "first"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`, `y`	Network inputs.
`method`	How to combine the edge sets:
	`"union"` (default) every edge of either network, over the union of the node sets
	`"intersection"` only edges present in both, over the nodes common to both
	`"difference"` edges of `x` that are not in `y`, over the nodes of `x`
`weight`	How to combine the weights of an edge present in both: `"sum"` (default), `"mean"`, `"max"`, `"min"`, or `"first"` (keep `x`'s weight).
`keep_format`	Logical. Return `x`'s format when TRUE.
`directed`	Logical or NULL. If NULL (default), the result is directed when either input is.

Value

A `cograph_network` over the combined node set, or `x`'s format when `keep_format = TRUE`. Nodes are ordered with `x`'s first, then any node only `y` has.

See Also

`add_edges`, `plot_difference`

Examples

```
bind_networks(regulation_net, t(regulation_net))
```

centrality *Calculate Network Centrality Measures*

Description

Computes centrality measures for nodes in a network and returns a tidy data frame. Accepts matrices, edge-list data frames, igraph objects, cograph_network, or tna objects.

Usage

```
centrality(
  x,
  type = c("basic", "extended", "all"),
  measures = NULL,
  include = NULL,
  mode = "all",
  normalized = FALSE,
  weighted = TRUE,
  directed = NULL,
  loops = TRUE,
  simplify = "sum",
  digits = NULL,
  sort_by = NULL,
  cutoff = -1,
  invert_weights = NULL,
  alpha = 1,
  damping = 0.85,
  personalized = NULL,
  transitivity_type = "local",
  isolates = "nan",
  lambda = 1,
  diffusion_method = NULL,
  k = 3,
  states = NULL,
  decay_parameter = 0.5,
  dmnc_epsilon = 1.7,
  membership = NULL,
  katz_alpha = 0.1,
  hubbell_weight = 0.5,
  shapley_k = 2,
  shapley_cutoff = 2,
  s_shell_a = 0.5,
  discount_p = 0.01,
  ncvote_theta = 0.5,
  comm_r = "max_intra",
  ld_radius = 2,
  enrenew_depth = 2,
```

```
    voterank_lambda = 0.1,
    contraction_rho = 5,
    wks_alpha = 1,
    wks_beta = 1,
    renewed_threshold = 2,
    kpath_k = 3,
    kpath_len = 3,
    epc_threshold = 0.5,
    epc_runs = 1000,
    epc_seed = NULL,
    betweenness_delta = 1,
    closeness_delta = 1,
    gravity_mass = "kshell",
    gravity_radius = 3,
    mdd_lambda = 0.7,
    volume_radius = 2,
    diffusion_q = 1,
    diffusion_steps = 3,
    ds_beta = 0.1,
    ds_mu = 1,
    ds_steps = 5,
    cda_alpha = 0.5,
    icc_alpha = 0.2,
    exogenous_base = "reverse_closeness",
    wlr_alpha = 1,
    alr_h_mode = "all",
    grc_gamma = 1,
    rwd_decay = 0.5,
    rwd_node_weights = NULL,
    linerank_aggregation = "probability",
    bridging_steps = 2,
    bridging_values = NULL,
    proximal_variant = "source",
    exf_alpha = 2,
    beta_direction = "positive",
    ninl_order = 3,
    ninl_radius = NULL,
    map_flow = "unrecorded",
    map_convention = "paper",
    sr_prior = 0,
    mcgm_radius = 2,
    mcgm_alpha = NULL,
    dkgm_radius = 2,
    nd_order = 2,
    nd_decay = 0.2,
    nd_mass = "degree",
    ira_mass = "coreness",
    ira_alpha = 1,
```

```
  ira_tol = 1e-06,
  ira_max_iter = 1000,
  iira_beta = 0.2,
  iira_steps = 50,
  hcc_delta = 0.5,
  lhc_radius = 2,
  tpr_alpha = 0.85,
  tpr_k = 0.85,
  tpr_decay = 1,
  tpr_tol = 1e-14,
  tpr_max_iter = 1000,
  rsp_beta = 0.01,
  rsp_cost = c("inverse", "weight"),
  re_indexes = c("degree", "closeness", "betweenness", "constraint"),
  re_negative = NULL,
  tna_network = NULL,
  psych_network = NULL,
  ...
)
```

Arguments

x	Network input (matrix, edge-list data frame, igraph, network, cograph_network, tna object).
type	Character scalar selecting a curated tier of measures when `measures` is not supplied. One of: `"basic"` The default. Six measures: `degree`, `strength`, `closeness`, `betweenness`, `eigenvector`, `pagerank`. `"extended"` The basic tier plus harmonic, coreness, eccentricity, radiality, lin, decay, load, stress, katz, alpha, power, authority, leverage, constraint, effective_size, bridging, transitivity, subgraph, diffusion, laplacian, kreach, current_flow_betweenness and current_flow_closeness. `"all"` Every measure except the costly ones (see `include` and `list_centralities`). Passing `measures` explicitly overrides `type`.
measures	Character vector of measure names to compute. When `NULL` (default) the tier selected by `type` is used. `"all"` is a shortcut for `type = "all"`. Unknown names raise an error. `list_centralities` returns every valid name together with its orientation, mode support, partition requirement, weight use and cost. The measures fall into these groups: **Core** "degree", "strength", "betweenness", "closeness", "eigenvector", "pagerank", "authority", "hub", "eccentricity", "coreness", "constraint", "transitivity", "harmonic", "alpha", "power", "subgraph". **Flow and spreading** "diffusion", "leverage", "kreach", "laplacian", "load", "current_flow_closeness", "current_flow_betweenness", "voterank", "percolation".

Distance-based "radiality", "lin", "decay", "residual_closeness", "dangalchev", "generalized_closeness", "harary", "average_distance", "barycenter", "wiener", "closeness_vitality".

Spectral and walk-based "communicability", "communicability_betweenness", "random_walk".

Path-based "stress", "flow_betweenness".

Local and neighborhood "lobby", "entropy", "semilocal", "clusterrank", "bottleneck", "centroid", "mnc", "dmnc", "lac", "topological_coefficient", "bridging", "local_bridging", "effective_size", "diversity", "cross_clique", "markov".

Influence "integration", "expected", "gilschmidt".

Directed only "salsa", "leaderrank", "trophic_level", "pairwisedis", "prestige_domain", "prestige_domain_proximity".

Community-aware (require `membership`) "participation", "within_module_z", "gateway", "modularity_vitality" (Magelinski et al. 2021), "community_hub_bridge" (Ghalmane et al. 2019), "community_based" (Zhao et al. 2015), "comm_centrality" (Gupta et al. 2016), "community_mediator" (Tulu et al. 2018), and the Gould-Fernandez roles "brokerage_coordinator", "brokerage_itinerant", "brokerage_representative", "brokerage_gatekeeper", "brokerage_liaison", which also require a directed graph. See `centrality_modularity_vitality` and `centrality_brokerage_coordinator`.

Spreader identification "gravity", "collective_influence", "local_hindex", "hindex_strength", "onion", "second_order", "infection", "nonbacktracking", "spanning_tree". See `centrality_gravity`.

Classical "katz" (Katz 1953), "hubbell" (Hubbell 1965), "information" (Stephenson and Zelen 1989), "reaching_local" (Mones et al. 2012). See `centrality_katz`, `centrality_hubbell`, `centrality_information`, `centrality_pairwisedis` and `centrality_reaching_local`.

Psychometric "expected_influence_1", "expected_influence_2" (Robinaugh, Millner and McNally 2016). Expected influence keeps the sign of each edge, which matters in networks with negative edges such as partial-correlation and glasso networks.

Scaling and dimension "distance_entropy" (Stella and De Domenico 2018), "local_dimension" (Pu et al. 2014), "local_information_dimension" (Wen and Deng 2020), "neighborhood_connectivity" (Maslov and Sneppen 2002), "local_dimension_fixed" (Silva and Costa 2013), "fuzzy_local_dimension" (Wen and Jiang 2019), "local_volume_dimension" (Li and Deng 2021). See `centrality_distance_entropy`, `centrality_local_dimension`, `centrality_local_informat` `centrality_neighborhood_connectivity` and `centrality_local_dimension_fixed`.

Games, information and seed selection "shapley_game1", "shapley_game2", "shapley_game3" (Michalak et al. 2013), "access_information", "hide_information" (Rosvall et al. 2005), "rumor" (Shah and Zaman 2011), "entropy_variation_degree", "entropy_variation_betweenness" (Ai 2017), "s_shell" (Liu et al. 2017), "degree_discount", "single_discount" (Chen, Wang and Yang 2009). See `centrality_shapley_game1`, `centrality_access_information`, `centrality_rumor`, `centrality_community_hub_bridge`, `centrality_entropy_variation`, `centrality_s_shell`, `centrality_degree_discount` and `centrality_community_based`.

VoteRank family "ncvoterank" (Kumar and Panda 2020), "wvoterank" (Sun et al. 2019), "enrenew" (Guo et al. 2020), "voterank_plus" (Liu et al. 2021). See centrality_ncvoterank and centrality_wvoterank.

Contraction, walks and local structure "node_contraction", "node_contraction_improved" (Tan et al. 2006; Wang et al. 2011), "two_way_rw" (Curado et al. 2022), "heatmap" (Duron 2020), "flow_coefficient" (Honey et al. 2007), "local_entropy" (Nie et al. 2016), "weighted_h_index" (Gao et al. 2019), "redundancy" (Burt 1992), "weighted_kshell" (Garas et al. 2012), "renewed_coreness" (Liu et al. 2015), "geodesic_kpath" (Borgatti and Everett 2006). See centrality_node_contractic centrality_two_way_rw, centrality_heatmap and centrality_weighted_kshell.

Efficiency, cores and percolation "local_efficiency" (Latora and Marchiori 2001), "s_core" (Eidsaa and Almaas 2013), "fragmentation" (Borgatti 2006), "kpath" (Sade 1989), "epc" (Lin et al. 2008). See centrality_local_efficiency.

Betweenness and closeness variants "length_scaled_betweenness" (Brandes 2008), "delta_betweenness" and "delta_closeness" (Agneessens et al. 2017), "ego_betweenness" (Everett and Borgatti 2005). Bounded-distance (k-) betweenness is betweenness with cutoff = k. See centrality_length_scaled_betweenness.

The remaining measures are described under Details.

include — Character vector of costly measures to add to a tier, or "costly" for all of them. type = "all" holds back the measures whose cost grows steeply with network size (the costly column of list_centralities). A measure named in measures is always computed, whatever its cost. Unknown names raise a cograph_unknown_measure error. Default NULL.

mode — For directed networks: "all" (default), "in", or "out". Affects the mode-aware measures, whose output columns carry a mode suffix. These include degree, strength, closeness, eccentricity, coreness, harmonic, diffusion, leverage, kreach, the distance-based measures, most community-aware measures, and expected influence (the mode_aware column of list_centralities).

normalized — Logical. If TRUE, each measure is divided by its maximum, which scales non-negative measures to 0-1. A measure whose maximum is not positive is left unchanged. Closeness follows igraph's normalization instead, multiplying each value by the number of nodes the node reaches, and harmonic centrality is divided by $n-1$ only. Under psych_network = TRUE expected influence is divided by its maximum absolute value and keeps its sign. Default FALSE.

weighted — Logical. Use edge weights if available. Default TRUE. With FALSE every measure works on the binary adjacency, every edge with weight one.

directed — Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

loops — Logical. If TRUE (default), keep self-loops. Set to FALSE to remove them before calculation. tna_network = TRUE changes the default to FALSE.

simplify — How to combine multiple edges between the same node pair (possible only from edge-list, cograph_network or igraph input). Options: "sum" (default), "mean", "max", "min". FALSE and "none" also sum them, because the network is held as a dense weight matrix, which cannot carry parallel edges.

digits — Integer or NULL. Round all numeric columns to this many decimal places. Default NULL (no rounding).

`sort_by` Character or NULL. Column name to sort results by (descending order). Default NULL (original node order). A name that is not a column of the result raises an error.

`cutoff` Maximum path length to consider for betweenness, closeness, harmonic centrality and the distance-based closeness variants (radiality, lin, decay, residual_closeness, dangalchev, generalized_closeness, harary, average_distance, barycenter, wiener, centroid, closeness_vitality, delta_closeness). Default -1 (no limit). A positive value ignores longer paths, which shortens computation on large networks and changes the values.

`invert_weights` Logical or NULL. Whether path- and distance-based measures (for example betweenness, closeness, harmonic, k-reach, radiality, decay, stress, flow betweenness and related variants) invert the weights, so that higher weights mean shorter paths. A message reports the inversion. Eccentricity always reads the raw edge weights as distances. The default `NULL` is TRUE for tna objects (transition probabilities) and FALSE otherwise, as in igraph and sna. TRUE suits strength or frequency weights (the qgraph convention) and FALSE suits distance or cost weights.

`alpha` Numeric. Exponent for weight transformation when `invert_weights = TRUE`. Distance is computed as `1 / weight^alpha`. Default 1. Higher values increase the influence of weight differences on path lengths.

`damping` PageRank damping factor. Default 0.85. Must be between 0 and 1.

`personalized` Non-negative numeric vector of reset probabilities for personalized PageRank, one value per node. A named vector is matched to the node names, an unnamed one is taken in node order. The vector is rescaled to sum to 1. Default NULL (standard PageRank).

`transitivity_type`

Type of transitivity to calculate: "local" (default), "global", "undirected", "localundirected", "barrat" (weighted), "weighted", or "onnela". The first six follow the conventions of `igraph::transitivity()`. `"global"` and `"undirected"` give one graph-level value, repeated on every row. `"onnela"` computes the weighted clustering coefficient of Zhang and Horvath (2005) on the symmetrized matrix `x + t(x)`, the form tna uses under that name, and matches `tna::centralities(x, "Clustering")`. `tna_network = TRUE` changes the default to `"onnela"`.

`isolates` Value of local transitivity at nodes where it is undefined: "nan" (default) returns NaN, "zero" returns 0.

`lambda` Diffusion scaling factor for diffusion centrality. Default 1. Only used when `diffusion_method = "kandhway_kuri"`.

`diffusion_method`

Character or NULL. Selects the diffusion-centrality formula. `"kandhway_kuri"` (Kandhway & Kuri, 2014) computes the 1-hop binary-degree neighborhood sum $\lambda d_v + \lambda \sum_{u \in N(v)} d_u$. `"power_series"` computes the matrix power series $\mathrm{rowSums}(P + P^2 + \ldots + P^n)$ on the (optionally diagonal-zeroed) weighted matrix and matches `tna::centralities(., measures = "Diffusion")` when `loops = FALSE`. Default NULL auto-detects: `"power_series"` for tna objects (transition probabilities), `"kandhway_kuri"` otherwise.

`k` Distance bound for `"kreach"`, the number of nodes reachable within `k` steps. Default 3.

`states` Numeric vector of percolation states for percolation centrality, one per node. Each value represents how "activated" or "infected" a node is. A named vector is matched to the node labels, missing nodes get state 1, and values are clipped to 0-1. Default NULL gives every node state 1, which makes percolation equal to betweenness divided by $(n-1)(n-2)$.

`decay_parameter` Numeric. Decay parameter for decay and generalized closeness centrality, strictly between 0 and 1; smaller values discount distant nodes more. Other values raise a `cograph_bad_parameter` error when one of these measures is requested. Default 0.5.

`dmnc_epsilon` Numeric. Epsilon exponent for DMNC (Density of Maximum Neighborhood Component), a single positive number. Default 1.7, as recommended by Lin et al. (2008). centiserve uses 1.67. Other values raise a `cograph_bad_parameter` error when `"dmnc"` is requested.

`membership` Integer vector of community assignments (one per node) for the community-aware measures listed under `measures`. Default NULL. Without it those measures return `NA` with a warning of classes `cograph_bad_membership` and `cograph_undefined_measure`. `"map_equation"` uses it when supplied.

`katz_alpha` Attenuation factor for Katz centrality. The Katz series converges only for $\alpha < 1/\rho(A)$. Otherwise the measure raises a `cograph_katz_diverged` warning. Default 0.1, the centiserve and NetworkX convention. Only used when `"katz"` is in `measures`.

`hubbell_weight` Weight factor w for Hubbell centrality. Must be positive and satisfy $w \cdot \rho(W) < 1$ for solvability; otherwise the measure warns and returns `NA`. Default 0.5. Only used when `"hubbell"` is in `measures`.

`shapley_k` Neighbor threshold k for `"shapley_game2"`. Default 2. See `centrality_shapley_game2`.

`shapley_cutoff` Hop cutoff for `"shapley_game3"`. Default 2. See `centrality_shapley_game3`.

`s_shell_a` Exponent of the asymmetric link weights for `"s_shell"`. A single non-negative number; default 0.5. See `centrality_s_shell`.

`discount_p` Propagation probability for `"degree_discount"`. Default 0.01. See `centrality_degree_discount`.

`ncvote_theta` Weight of the plain vote in `"ncvoterank"`. Default 0.5. See `centrality_ncvoterank`.

`comm_r` Scale R of `"comm_centrality"`: `"max_intra"` (default) or a single positive number.

`ld_radius` Radius for `"local_dimension_fixed"`, in hops. A single number of at least 1; default 2.

`enrenew_depth` Renewal radius for `"enrenew"`. Default 2.

`voterank_lambda` Suppression factor for `"voterank_plus"`. Default 0.1.

`contraction_rho` α/β for `"node_contraction_improved"`. Default 5.

`wks_alpha`, `wks_beta` Degree and strength exponents for `"weighted_kshell"`. Default 1 and 1.

renewed_threshold
: Diffusion-importance threshold for "renewed_coreness". Default 2.

kpath_k
: Maximum path length for "geodesic_kpath". Default 3.

kpath_len
: Maximum path length for "kpath". Default 3; the enumeration is exhaustive, so cost grows with the branching factor to this power.

epc_threshold
: Edge removal probability for "epc". Default 0.5.

epc_runs
: Number of percolation realizations for "epc". Default 1000.

epc_seed
: Random seed for "epc". Default NULL draws from the session's random number stream, so the estimate varies between calls. A seed makes the estimate reproducible, and the caller's random number state is restored afterwards.

betweenness_delta
: Decay exponent for "delta_betweenness". Default 1; 0 gives ordinary betweenness.

closeness_delta
: Distance exponent for "delta_closeness". Default 1, which is harmonic over $n - 1$.

gravity_mass
: Mass in "gravity": "kshell" (default, Ma et al. 2016), "degree" (Li et al. 2019) or "legacy", which uses a unit focal mass and a partner mass equal to degree times k-shell index.

gravity_radius
: Largest distance each gravity source reaches in "gravity", "extended_gravity", "mixed_gravity" or "extended_mixed_gravity": a number (default 3), "auto" for half the mean distance, or NULL for the whole graph. The automatic radius averages the finite positive distances, rounds half to even, and is at least 1.

mdd_lambda
: Exhausted-degree weight for "mdd", between 0 and 1. Default 0.7. See centrality_truss.

volume_radius
: Closed neighborhood radius for "volume": a nonnegative integer or Inf, default 2. Degrees are measured in the full simple undirected graph. See centrality_volume.

diffusion_q
: Multiplier between 0 and 1 for "diffusion_centrality", default 1. It is separate from lambda, which scales "diffusion".

diffusion_steps
: Nonnegative integer horizon for "diffusion_centrality", default 3. See centrality_diffusion_centrality for its weighted-walk definition, direction, probability interpretation and precision limits.

ds_beta
: Spreading rate for "dynamics_sensitive", between zero and one, default 0.1.

ds_mu
: Recovery rate for "dynamics_sensitive", between zero and one, default 1. Zero selects the SI case.

ds_steps
: Nonnegative integer horizon for "dynamics_sensitive", default 5. See centrality_dynamics_sensit

cda_alpha
: Degree-versus-strength weight for "cda", between zero and one; default 0.5. See centrality_cda.

icc_alpha
: Shortest-path multiplicity exponent for "improved_closeness", between zero and one; default 0.2.

exogenous_base
: Base for "exogenous": reverse_closeness (default), betweenness or degree. See centrality_exogenous.

wlr_alpha
: Finite in-degree exponent for "weighted_leaderrank", default one. See centrality_weighted_leade

`alr_h_mode` H-index convention for `"adaptive_leaderrank"`: all (default), out or in. See `centrality_adaptive_leaderrank`.

`grc_gamma` Finite nonnegative regularization strength for `"graph_regularization"`, default one. See `centrality_graph_regularization`.

`rwd_decay` Finite first-arrival discount in $[0, 1)$ for `"random_walk_decay"`, default 0.5.

`rwd_node_weights` Nonnegative starting weights for `"random_walk_decay"`; NULL means ones. See `centrality_random_walk_decay`.

`linerank_aggregation` LineRank endpoint aggregation: probability (default) or weight. See `centrality_linerank`.

`bridging_steps` Nonnegative bridging-capital walk horizon, default two.

`bridging_values` Optional source-destination value matrix for `centrality_bridging_capital`; NULL uses ones.

`proximal_variant` Proximal betweenness role: source (default), target, sum, or union. See `centrality_proximal_betweenn`

`exf_alpha` Modified Expected Force degree factor, default two, finite and greater than one.

`beta_direction` BG-index orientation, positive (default) or negative. See `centrality_beta_measure`.

`ninl_order` Nonnegative NINL iteration count, default three.

`ninl_radius` NINL hop radius, NULL for ceiling of mean path length. See `centrality_ninl` for disconnected graphs and overrides.

`map_flow` Map equation flow model, unrecorded (default) or recorded.

`map_convention` Map equation coding convention, paper (default) or infomap. See `centrality_map_equation`.

`sr_prior` SpectralRank diagonal prior, default zero; scalar or one value per node. See `centrality_spectralrank`.

`mcgm_radius` MCGM hop cutoff, default two; NULL includes all reachable nodes.

`mcgm_alpha` MCGM coefficient, NULL for the published adaptive rule. See `centrality_mcgm` for disconnected-graph conventions.

`dkgm_radius` DKGM hop cutoff, default two as in the paper's printed example; NULL or infinity includes all reachable nodes and "auto" applies the paper's half-mean-distance rule with cograph rounding. See `centrality_dkgm`.

`nd_order` Steps of neighbors summed by `"neighbor_distance"`, a nonnegative whole number, default two; zero returns `nd_mass`. See `centrality_neighbor_distance`.

`nd_decay` Per-step decay for `"neighbor_distance"`, a finite number, default 0.2 as in the source.

`nd_mass` Benchmark centrality summed by `"neighbor_distance"`: degree (default) or coreness.

`ira_mass` Node centrality allocated by `"ira"` and `"iira"`: coreness (default, the k-shell index both sources use in their worked examples) or degree. See `centrality_ira`.

`ira_alpha` Exponent on the `"ira"` mass, a finite number, default one as in the source.

`ira_tol` Stopping tolerance for `"ira"` on the largest absolute change between iterates, a positive finite number, default `1e-6` as in the source.

`ira_max_iter` Iteration bound for `"ira"`, a whole number of at least one, default 1000. Reaching it raises a `cograph_no_converge` warning, which always happens on a bipartite component with unequal vertex classes. See `centrality_ira`.

`iira_beta` Spreading rate for `"iira"`, a number in $(0, 1]$, default 0.2 as in the source.

`iira_steps` Iterations for `"iira"`, a nonnegative whole number, default 50 as in the source; zero returns the initial unit resource. See `centrality_iira`.

`hcc_delta` Weight on a node's own degree in the extended degree used by `"hcc"` and `"ehcc"`, a single number in $[0, 1]$, default 0.5 as in the source. A value of one gives the classical degree and zero drops the node's own degree. Values outside $[0, 1]$ raise an error. See `centrality_hcc`.

`lhc_radius` Radius of the ball $\Phi(v)$ summed over by `"lhc"`, the d of the source's equation (1); a single whole number of at least one, default 2 as in the source, which reports 2-3 as optimal. At one the ball contains only the neighbors. At or above the diameter the score no longer changes. Values below one and non-integers raise an error. See `centrality_lhc`.

`tpr_alpha` Jump probability of the trust-PageRank iteration used by `"trust_pagerank"`, a single number strictly between zero and one, default 0.85 as the source sets it below its equation (7). See `centrality_trust_pagerank`.

`tpr_k` Weight of the degree ratio in the trust value of `"trust_pagerank"`, the k of the source's equation (6). The similarity ratio receives weight $1 - k$. A single number in $[0, 1]$, default 0.85, the value selected in section 3.3 of the source. One drops the similarity and zero drops the degree.

`tpr_decay` Attenuation factor of the similarity recursion used by `"trust_pagerank"`, the C of the source's equation (4); a single number in $(0, 1]$, default 1 as in the source. C changes the result. See `centrality_trust_pagerank`.

`tpr_tol` Convergence tolerance on the largest relative change of either trust-PageRank recursion, a single positive number, default `1e-14`. The tolerance is relative because the similarities on one graph span many orders of magnitude. See `centrality_trust_pagerank`.

`tpr_max_iter` Iteration bound for both trust-PageRank recursions, a whole number of at least one, default 1000. Reaching it raises a `cograph_no_converge` warning.

`rsp_beta` Inverse temperature of the randomized-shortest-paths model used by `"rsp_betweenness"`, a single finite number strictly above zero, default 0.01, the value `NetworkToolbox::rspbc()` recommends. Small values approach the random-walk limit. Values of 1 and above move the measure towards shortest paths. See `centrality_rsp_betweenness`.

`rsp_cost` How an edge weight becomes a traversal cost for `"rsp_betweenness"`: `"inverse"` (default) for $C = 1/w$, reading a weight as an affinity, or `"weight"` for $C = w$, reading it as a distance. Both settings give unit cost per arc on a binary graph. See `centrality_rsp_betweenness`.

`re_indexes` Constituent indexes integrated by `"relative_entropy"`, default the source's four distinctiveness indexes; the vocabulary also holds `"n_components"` and `"largest_component"`. See `centrality_relative_entropy`.

`re_negative` Which of `re_indexes` are negative indexes, NULL for the source's own declarations. See `centrality_relative_entropy`.

`tna_network` Logical or NULL. Umbrella switch that forces tna-style conventions across all measures. `NULL` (default) is TRUE when `x` is a `tna`, `group_tna`, `ctna`, `ftna` or `atna` object (or a group of these) and FALSE otherwise. `TRUE` applies the tna conventions to any input, setting `invert_weights = TRUE`, `loops = FALSE`, `diffusion_method = "power_series"` and `transitivity_type = "onnela"`. `FALSE` keeps the cograph defaults for tna inputs as well. An argument passed explicitly always takes precedence over `tna_network`.

`psych_network` Logical or NULL. Switch for signed psychometric network conventions. `NULL` (default) auto-detects TRUE when a signed weighted network is evaluated with expected-influence measures. When `TRUE`, normalized expected influence is divided by the maximum absolute expected-influence value, which keeps its sign and bounds it between -1 and 1. `FALSE` keeps the generic cograph normalization.

`...` Additional arguments (currently unused)

Details

The following centrality measures are available:

degree Count of edges (supports mode: in/out/all)

strength Weighted degree (supports mode: in/out/all)

betweenness Shortest path centrality

closeness Inverse distance centrality (supports mode: in/out/all)

eigenvector Influence-based centrality

pagerank Random walk centrality (supports damping and personalization)

authority HITS authority score

hub HITS hub score

eccentricity Maximum distance to other nodes (supports mode). Distances use the raw edge weights whatever `invert_weights` says, and hop counts with `weighted = FALSE`.

coreness K-core membership (supports mode: in/out/all)

constraint Burt's constraint (structural holes)

transitivity Local clustering coefficient (supports multiple types)

harmonic Harmonic centrality, the sum of inverse distances. It stays finite on disconnected graphs (supports mode: in/out/all)

diffusion Diffusion degree centrality. With `diffusion_method = "kandhway_kuri"` it is the scaled degree of the node plus the scaled degrees of its neighbors (supports mode: in/out/all, lambda scaling)

leverage Leverage centrality. Influence over neighbors based on relative degree differences (supports mode: in/out/all)

kreach K-reach centrality. Number of nodes reachable within `k` steps (supports mode: in/out/all)

alpha Alpha centrality. Influence through paths, attenuated by length, with a unit exogenous contribution at every node (supports mode: in/out/all)

power Bonacich power centrality. Influence based on connections to other influential nodes (supports mode: in/out/all)

subgraph Subgraph centrality. Participation in closed walks, with shorter walks weighted more heavily

laplacian Laplacian centrality with the local formula of Qi et al. (2012). Matches NetworkX and `centiserve::laplacian()`

load Load centrality. Share of shortest-path load passing through the node, with load split evenly at each branching point

current_flow_closeness Information centrality. Closeness based on electrical current flow (requires a connected graph)

current_flow_betweenness Random-walk betweenness. Betweenness based on electrical current flow (requires a connected graph)

voterank VoteRank. Influential spreaders selected by iterative voting. The value is the election order rescaled so that the first elected node scores 1 and the last scores $1/n$

percolation Percolation centrality. Shortest-path betweenness weighted by the node states in `states`. With equal states it is betweenness divided by $(n-1)(n-2)$

radiality Radiality centrality (centiserve). Sum of (diam + 1 - d) normalized by n-1.

lin Lin's centrality. Reachable nodes squared divided by sum of distances.

decay Decay centrality. Sum of δ^d over all nodes, the node itself included, with δ = `decay_parameter`.

residual_closeness Residual closeness. Sum of $1/2^d$ over all nodes, the node itself included.

dangalchev Dangalchev closeness. Same values as `residual_closeness`.

generalized_closeness Generalized closeness. Same formula as `decay`, using `decay_parameter`.

harary Harary centrality. Sum of 1/d^2 for all reachable pairs.

average_distance Average distance (centiserve). Sum of distances / (n+1).

barycenter Barycenter centrality. 1 / sum of distances.

wiener Wiener index. Total sum of shortest path distances from node.

closeness_vitality Closeness vitality. Drop in Wiener index when node removed.

communicability Total communicability. Row sums of matrix exponential.

communicability_betweenness Communicability betweenness. Fraction of communicability through each node.

random_walk Random walk centrality. Inverse sum of random walk distances (requires connected graph).

stress Stress centrality. Number of shortest paths through node.

flow_betweenness Flow betweenness. Max-flow based betweenness.

lobby Lobby index (h-index of neighborhood).

entropy Graph entropy centrality. Entropy change on node removal.

semilocal Semi-local centrality. Triple-nested neighborhood sum.

clusterrank ClusterRank. Clustering coefficient times neighbor degree sum.

bottleneck Bottleneck centrality. Count of shortest path trees where node is critical.

centroid Centroid value. Minimum f(v,i) across all nodes.

mnc Maximum Neighborhood Component size.

dmnc Density of Maximum Neighborhood Component.

topological_coefficient Topological coefficient. Shared neighbor ratio.

bridging Bridging centrality. Betweenness times bridging coefficient.

local_bridging Local bridging. (1/degree) times bridging coefficient.

effective_size Burt's effective size. Degree minus redundancy.

diversity Diversity centrality. Shannon entropy of edge weight distribution.

cross_clique Cross-clique connectivity. Count of cliques containing node.

markov Markov centrality. Inverse mean first passage time (requires connected graph).

integration Integration centrality. Distance-based influence.

expected Expected centrality. Sum of neighbor degrees.

gilschmidt Gil-Schmidt power index. Sum of 1/d normalized by n-1.

salsa SALSA authority scores (directed graphs only).

leaderrank LeaderRank. PageRank with ground node (directed graphs only).

participation Participation coefficient. Diversity of inter-community connections (requires `membership`).

within_module_z Within-module degree z-score. Intra-community connectivity (requires `membership`).

gateway Gateway coefficient. Inter-community brokerage weighted by centrality (requires `membership`).

distance_entropy Normalized Shannon entropy of a node's hop-distance profile. It is 1 when the distances are spread evenly and 0 when all lie at one distance.

local_dimension Growth exponent of the ball around a node (slope of $\ln B_i(r)$ on $\ln r$). Lower values mark more influential nodes.

local_information_dimension Entropy-weighted local dimension over boxes up to half the node's eccentricity. Higher values mark more influential nodes.

neighborhood_connectivity Mean degree of a node's neighbors (average neighbor degree); isolates score 0.

modularity_vitality Drop in modularity when the node is removed under a fixed partition. Positive values mark community hubs and negative values mark bridges (requires `membership`).

shapley_game1, shapley_game2, shapley_game3 Shapley value of the node in the coverage games of Michalak et al. (2013): one-hop coverage, `shapley_k`-neighbor coverage, and coverage within `shapley_cutoff` hops. Values sum to the node count.

access_information Mean bits needed to reach every other node along shortest paths without a map. Low values mark well-connected nodes.

hide_information Mean bits others need to find the node. High values mark hidden nodes.

rumor Log rumor centrality on the node's BFS tree: log of the number of spreading orders that could start there.

community_hub_bridge Community size times intra-community degree plus number of other communities touched times inter-community degree (requires `membership`).

entropy_variation_degree, entropy_variation_betweenness Drop in the Shannon entropy of the degree (by `mode`) or betweenness distribution when the node is deleted; signed, nats.

s_shell Shell index of the strength-based peeling with asymmetric topological link weights, exponent `s_shell_a`.

degree_discount, single_discount Greedy seed-selection order under degree discounting (`discount_p`) or unit discounting, scored 1 for the first selected down to 1/n.

ncvoterank VoteRank with voters weighted by normalized neighborhood coreness (`ncvote_theta`); election order scored like `voterank`.

community_based, comm_centrality, community_mediator Links weighted by the size of the community they reach; Gupta's scaled intra/inter-degree combination (`comm_r`); base-2 entropy of the link distribution over communities times degree share (all require `membership`).

local_dimension_fixed, fuzzy_local_dimension, local_volume_dimension Silva-Costa estimator at `ld_radius`; slope of the fuzzy ball, where higher values mark more influential nodes; slope of the degree volume, where lower values mark more important nodes.

wvoterank, enrenew, voterank_plus Election orders of the weighted, entropy-based (`enrenew_depth`) and degree-weighted (`voterank_lambda`) VoteRank variants, scored like `voterank`.

node_contraction, node_contraction_improved One minus the agglomeration ratio after contracting the node with its neighbors; the improved form adds the same score of its edges on the line graph (`contraction_rho`).

two_way_rw Number of node pairs whose most likely two-way random-walk route passes through the node.

heatmap Farness minus mean neighbor farness. Lower values mark more central nodes.

flow_coefficient Share of neighbor pairs linked through the node but not directly.

local_entropy $-\sum_{j \in N(i)} k_j \ln k_j$. Lower values mark more central nodes.

weighted_h_index h-index over topological link weights $k_i k_j$ repeated k_j times.

redundancy Mean degree of the neighbors inside the ego network; degree minus effective size.

weighted_kshell k-shell on $(k^\alpha s^\beta)^{1/(\alpha+\beta)}$ after Garas' weight normalization (`wks_alpha`, `wks_beta`).

renewed_coreness k-core of the graph after removing links whose diffusion importance is below `renewed_threshold`.

geodesic_kpath Number of shortest paths of length at most `kpath_k` starting at the node.

local_efficiency Global efficiency of the subgraph induced on the node's neighbors, the node itself removed. This differs from `igraph::local_efficiency()`, which measures the distances between those neighbors through the rest of the network.

s_core Largest strength threshold whose s-core still contains the node; the k-core number when weights are absent.

fragmentation Distance-weighted fragmentation of the network after deleting the node. Higher means a more disruptive removal.

kpath Number of simple paths of length at most `kpath_len` that the node lies on, endpoints included.

epc Edge percolated component: mean size of the node's component over `epc_runs` bond-percolation realizations, as a share of the network. A Monte Carlo estimate; `epc_seed` makes it reproducible.

length_scaled_betweenness Betweenness with each separated pair weighted by $1/d(s,t)$.

delta_betweenness Betweenness with the pair weight $(h(s,t)-1)^{-\delta}$ (`betweenness_delta`), where $h(s,t)$ is the number of edges on a shortest path.

ego_betweenness Betweenness inside the node's own ego network.

delta_closeness $\sum_j d_{ij}^{-\delta}/(n-1)$ (`closeness_delta`).

truss, mdd Node truss number (k-2 triangles convention) and mixed-degree shell threshold (`mdd_lambda`). Both use the simple undirected skeleton; see `centrality_truss`.

bridging_coefficient, godfather, support Reciprocal-degree ratio, count of unconnected neighbor pairs, and count of triangle-supported relationships on the simple undirected skeleton.

volume Sum of degrees in the closed `volume_radius`-hop neighborhood on the simple undirected skeleton.

mcc Maximal clique centrality: sum of $(|C|-1)!$ over incident maximal cliques of size at least two. Costly; see `centrality_mcc` for isolate and precision conventions.

diffusion_centrality Finite-horizon weighted outgoing walks: $\sum_{t=1}^{T}(qA)^t\mathbf{1}$, with `diffusion_q` and `diffusion_steps`. Distinct from diffusion degree.

dynamical_importance Relative spectral-radius loss on vertex deletion, evaluated by repeated eigendecomposition. Costly; see `centrality_dynamical_importance` for zero-radius graphs.

dynamics_sensitive Finite-time spreading score including `ds_beta`, `ds_mu` and `ds_steps`; uses the simple undirected skeleton.

malatya Sum of focal-to-neighbor degree ratios on the simple undirected skeleton; the reciprocal of the bridging coefficient on nonisolated vertices.

resistance_curvature One minus half the incident conductance times effective-resistance sum. Weighted, componentwise and costly; see `centrality_resistance_curvature`.

extended_coreness Sum of neighbors' neighborhood coreness; equivalently the squared simple adjacency times core numbers.

dkgm Gravity with the degree k-shell index as the mass at both ends, default radius two; see `centrality_dkgm`.

neighbor_distance Benchmark centrality plus its decayed sums over non-backtracking walks of up to `nd_order` steps. See `centrality_neighbor_distance`.

ira Steady state of a unit resource repeatedly reallocated to neighbors in proportion to their `ira_mass`; conserved, so the scores of a component sum to its size. Warns `cograph_no_converge` where no steady state exists. See `centrality_ira`.

iira The same recursion with each share scaled by $1-(1-\beta)^{k_i}$ for the `iira_beta` spreading rate, run `iira_steps` times. Decays geometrically, so only the order is meaningful. See `centrality_iira`.

lnc Local neighbor contribution: the cubed degree times the binomial own-contribution factor $(1-1/d_i)^{d_i-1}$ times the neighbors' degree sum over $n-1$. Parameter-free; raw scores depend on the whole graph's order. See `centrality_lnc`.

ked KED method: the degree times one plus the normalized entropy of the neighbors' degrees times $\exp(K_i/N)$ for the neighbor-degree sum K_i and the whole graph's order N. Parameter-free. See `centrality_ked`.

hcc Hybrid characteristic centrality: the extended degree $\delta k_i + (1-\delta)\sum_{j\in N(i)} k_j$ over its maximum, plus the E-shell peeling round in which the node leaves over the number of rounds. Raw scores lie in $[0,2]$ and are not component-local. See `centrality_hcc`.

ehcc Extended hybrid characteristic centrality: the closed-neighborhood sum of `hcc`, the focal node counted once. See `centrality_ehcc`.

lhc Lhc index: the degree-and-triangle-share influence $C(v) = \sum_{u \in \Phi(v)} k_u(1 + TP(u))/d^2(uv)$ over the ball of radius `lhc_radius`, summed over the open neighborhood. The triangle share is normalized by $TNTS = \sum_u NTS(u)$, three times the number of distinct triangles, and is written as zero on a triangle-free graph. Raw scores are not component-local. See `centrality_lhc`.

iec Immediate effects centrality: the reciprocal mean length of the influence sequences that end at a node, $(n-1)/\sum_{i \neq j} m_{ij}$ for the mean first passage times $M = (I - Z + EZ_{dg})\mathrm{diag}(1/c)$ of the influence chain $W = A/\mathrm{rowSums}(A)$ built with $a_{ii} = 1$. Direction-sensitive and costly (one eigenproblem and two dense solves). `NA` at every node when the chain is reducible or the graph has one node. Not the same measure as `markov`. See `centrality_iec`.

dil Degree and importance of lines: the degree plus the share of each incident line's importance $I_e = (k_m - p - 1)(k_n - p - 1)/(p/2 + 1)$ that the node's own degree claims, $k_i + \sum_{j \in \Gamma_i} I_{e_{ij}}(k_i - 1)/(k_i + k_j - 2)$, with p the number of triangles on the line. Two-hop local and component-local; never below the node's degree. See `centrality_dil`.

trust_pagerank Trust-PageRank: a damped PageRank whose split of a node's score among its neighbors is the column-stochastic trust-value $T(i,j) = (1-k)s(i,j)/\sum_{l \in N_j} s(j,l) + k\, d_i/\sum_{l \in N_j} d_l$, with s the fixed point of SimRank restricted to the lines of the graph. Scores sum to one when no node is isolated. `NA` at every node of a component that has lines but no triangle, where the similarity vanishes and the ratio is undefined. Costly (two fixed-point recursions over dense matrices). See `centrality_trust_pagerank`.

rsp_betweenness Simple randomized shortest paths betweenness: the expected number of visits a node receives over the Boltzmann distribution on absorbing walks, summed over every ordered source-target pair. `rsp_beta` interpolates between the random-walk and shortest-path readings. Direction-sensitive, component-local, and costly (one dense inverse). See `centrality_rsp_betweenness`.

relative_entropy Normalized geometric mean of several index distributions, the minimum-relative-entropy integration of `re_indexes`; sums to one. See `centrality_relative_entropy`.

mixed_gravity Gravity with focal core-number and partner-degree masses, default radius three.

extended_mixed_gravity Sum of immediate neighbors' raw mixed gravitational centralities.

extended_gravity Sum of neighbors' raw k-shell gravity scores, with `gravity_radius` applied around each neighbor.

cda Weighted degree and strength, adjusted by Barrat clustering, plus weighted neighbor contributions; uses `cda_alpha`.

improved_closeness Closeness using distances divided by the number of shortest paths raised to `icc_alpha`.

exogenous Contribution to all other nodes' base centrality, measured by deletion. Selects a base using `exogenous_base`.

global_structure Exponential focal coreness times distance-discounted partner coreness (GSM).

hybrid_global_structure Exponential degree-coreness influences with an adaptive distance exponent (H-GSM).

improved_global_structure Exponential focal degree with partner degrees discounted by a global mean-degree distance exponent (IGSM).

weighted_leaderrank Stationary scores with ground-node outgoing weights determined by original in-degree and `wlr_alpha`.

linerank PageRank on the line graph, aggregated at endpoints; uses `damping` and `linerank_aggregation`.

expected_force Entropy of onward boundary degrees over all two-event transmission sequences.

mcgm Multi-characteristics gravity with degree, coreness and eigenvector masses; default radius two.

spectralrank Outgoing Perron eigenvector with a unit-linked ground node; `sr_prior` supplies optional diagonal information.

controlrank Smallest eigenvalue of each grounded symmetric row-Laplacian; see `centrality_controlrank`.

map_equation Codelength saving on silencing a node, conditional on the supplied partition, flow model and coding convention.

ninl Finite neighbor propagation of closed-neighborhood degree volume; uses `ninl_order` and `ninl_radius`.

beta_measure BG power shared by successors among predecessors; `beta_direction` selects positive or negative orientation.

localized_bridging, extended_local_bridging Betweenness in one-hop or two-hop ego networks times the original bridging coefficient.

modified_expected_force Expected Force multiplied by log degree with the scaling parameter `exf_alpha`.

proximal_betweenness First/last shortest-path intermediaries; uses `proximal_variant` on the directed unweighted skeleton.

x_degree Counts four-edge nonbacktracking walks with each node at the middle, using original neighbor excess degrees.

coleman_theil Concentration of dyadic Burt constraints across contacts; isolates zero and single-contact nodes one.

bridging_capital Information-walk loss under single-entry deletion; uses `bridging_steps` and `bridging_values`.

random_walk_decay Weighted sum of discounted first arrivals from random walks; uses `rwd_decay` and `rwd_node_weights`.

graph_regularization Reciprocal diagonal of the inverse regularized weighted Laplacian, using `grc_gamma`.

adaptive_leaderrank Stationary scores with destination weights determined by original H-indices using `alr_h_mode`.

Value

A base `data.frame` with one row per node, in the input's node order unless `sort_by` is given, and the columns:

- `node`: character, the node labels (the index as a string when the input carried no names)
- One numeric column per requested measure, with a mode suffix for the mode-aware measures (e.g., `degree_in`, `closeness_all`); see `list_centralities` for which measures carry a suffix. A measure that a tier supplied but that has no value on this input is an all-`NA` column.

Measures without a value on a given input

The community-aware measures warn and return an all-NA column when `membership` is missing, and the brokerage roles do the same on an undirected graph. `"relative_entropy"` has no value when one of its constituent indexes is zero at every node. Named in `measures` or `include`, it then raises a `cograph_undefined_index` error. Supplied by a tier, it gives a `cograph_undefined_measure` warning and an all-NA column, and the other measures of the tier are still computed. `"flow_betweenness"` requires the igraph package. Without igraph, naming it raises a `cograph_needs_igraph` error, and a tier gives the same warning and all-NA column.

Examples

```
centrality(regulation_net)
```

---

centrality_access_information
Access and Hide Information

Description

Access and hide information (Rosvall et al. 2005; Sneppen, Trusina and Rosvall 2005) measure the number of bits a walker needs to follow a shortest path without a map. The search information from i to j sums over all shortest paths $p(i, j)$, with k_i the degree of the source and $k_l - 1$ the choices left at each intermediate node:

$$S(i \to j) = -\log_2 \sum_{p(i,j)} \frac{1}{k_i} \prod_{l \in p,\, l \neq i,j} \frac{1}{k_l - 1}.$$

Access information A_i averages $S(i \to j)$ over targets j, and hide information H_i averages $S(j \to i)$ over sources.

Usage

```
centrality_access_information(x, ...)

centrality_hide_information(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Distances are hop counts, so edge weights are ignored. On a directed network every step uses the out-degree. The average runs over the nodes a walker can reach, or be reached from, so values stay finite on a disconnected network and equal the source's $1/N$ average on a connected one. Hubs score high on access information and low on hide information. On a star with five leaves the hub has access information 1.93 bits and hide information 0, and each leaf has access information 1.33 bits. The Centrality Zoo states the star case in reverse, and the values above follow the formulas and the source papers.

Value

A named numeric vector with one score per node, in input node order, in bits.

References

Rosvall, M., Trusina, A., Minnhagen, P., & Sneppen, K. (2005). Networks and cities: An information perspective. Physical Review Letters, 94, 028701.

Sneppen, K., Trusina, A., & Rosvall, M. (2005). Hide-and-seek on complex networks. Europhysics Letters, 69(5), 853-859.

See Also

centrality_closeness, centrality.

Examples

```
centrality_access_information(regulation_net)
centrality_hide_information(regulation_net)
```

---

centrality_adaptive_leaderrank

Adaptive LeaderRank

Description

Adaptive LeaderRank (Xu and Wang 2017) adds a ground node with H-index 1 that is linked in both directions to every node, and weights each arc from j to i by the H-index h_i of its target. The H-index of a node is the largest h such that at least h of its neighbors have degree at least h. The score is the stationary resource of the random walk on the row-normalized weights, started with one unit on every node and none on the ground.

Usage

```
centrality_adaptive_leaderrank(x, alr_h_mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
alr_h_mode	Neighbors and degrees used for the H-index: "all" (default), "out" or "in". On an undirected network the three agree.
...	Further arguments to centrality, such as normalized.

Details

The random walk keeps the direction of the arcs, and an undirected edge counts as two opposite arcs. Edge weights, loops and parallel arcs are ignored, and mode has no effect. H-indices are computed once on the original network. With alr_h_mode = "all" they use the simple undirected skeleton, "out" uses the out-degrees of out-neighbors and "in" the in-degrees of in-neighbors. The returned scores omit the ground, so they sum to less than N. A node with H-index 0 scores 0, and when every H-index is 0 all scores are NaN with a cograph_undefined_measure warning.

Value

A named numeric vector with one score per node, in input node order.

References

Xu, S., & Wang, P. (2017). Identifying important nodes by adaptive LeaderRank. Physica A, 469, 654-664. doi:10.1016/j.physa.2016.11.034.

See Also

centrality_weighted_leaderrank, centrality_lobby, centrality.

Examples

```
centrality_adaptive_leaderrank(regulation_net)
```

---

centrality_alpha *Alpha Centrality*

Description

Alpha centrality (Bonacich and Lloyd 2001) gives every node an exogenous score of one plus the scores of the nodes that point to it:

$$x = \alpha A^T x + e, \qquad x = (I - \alpha A^T)^{-1} e.$$

The attenuation is fixed at $\alpha = 1$ and e is a vector of ones, the defaults of igraph::alpha_centrality().

Usage

```
centrality_alpha(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "in" or "out".
...	Further arguments to centrality, such as directed and normalized.

Details

A holds the edge weights, or ones with weighted = FALSE, with the diagonal set to zero. The alpha argument of centrality is the weight-inversion exponent, which this measure does not read. The scores are positive when the spectral radius of A is below one and can be negative otherwise. A singular system or a negative edge weight raises an error of class cograph_singular_system.

On a directed network mode = "in" sums over incoming ties as above and equals igraph::alpha_centrality(), mode = "out" uses A in place of A^T and so sums over outgoing ties, and mode = "all" (default) uses the symmetrized weights $A + A^T$. On an undirected network the three modes agree.

Value

A named numeric vector with one score per node, in input node order.

References

Bonacich, P., & Lloyd, P. (2001). Eigenvector-like measures of centrality for asymmetric relations. Social Networks, 23(3), 191-201. doi:10.1016/S03788733(01)000387.

See Also

centrality_katz, centrality_eigenvector, centrality.

Examples

```
centrality_alpha(regulation_net)
```

---

centrality_authority *HITS Authority and Hub Scores*

Description

The HITS algorithm (Kleinberg 1999) assigns each node an authority score from the hubs that point to it and a hub score from the authorities it points to:

$$a = \lambda^{-1} A^T h, \qquad h = \lambda^{-1} A a.$$

Authorities are the dominant eigenvector of $A^T A$ and hubs the dominant eigenvector of $A A^T$.

Usage

```
centrality_authority(x, ...)

centrality_hub(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as directed.

Details

A holds the edge weights, or ones with weighted = FALSE. Both scores are scaled to a maximum of one, so they lie between 0 and 1. On an undirected network both equal eigenvector centrality. A network without edges gives every node a score of one. centrality_hub() returns the hub scores.

Value

A named numeric vector with one score per node, in input node order.

References

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5), 604-632. doi:10.1145/324133.324140.

See Also

centrality_eigenvector, centrality_pagerank, centrality.

Examples

```
centrality_authority(regulation_net)
centrality_hub(regulation_net)
```

---

centrality_average_distance

Average Distance

Description

Average distance is the sum of the shortest-path distances from a node to every node, divided by $n + 1$ as in the centiserve package:

$$AD(v) = \frac{1}{n+1} \sum_{w} d(v, w).$$

Lower values mark more central nodes.

Usage

```
centrality_average_distance(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1) and cutoff (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. invert_weights = TRUE uses $1/w^{\alpha}$ as the length, and weighted = FALSE counts hops. mode sets the direction of the paths. A node that cannot reach every other node scores Inf, so on a disconnected network every score is Inf.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_barycenter, centrality_closeness, centrality.

Examples

```
centrality_average_distance(regulation_net)
```

---

centrality_barycenter *Barycenter Centrality*

Description

Barycenter centrality is the reciprocal of the total shortest-path distance from a node to the nodes it reaches:

$$BC(v) = \frac{1}{\sum_{w \neq v} d(v, w)}.$$

Unreachable nodes are left out of the sum.

Usage

```
centrality_barycenter(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1), cutoff (largest path length considered, default -1 for no limit) and normalized (divide by the maximum, default FALSE).

Details

Edge weights are read as path lengths. invert_weights = TRUE uses $1/w^{\alpha}$ as the length, and weighted = FALSE counts hops. mode sets the direction of the paths. The formula is the one centrality_closeness computes, and with the default weighted = TRUE the two agree on every node that reaches another node. A node that reaches no other node scores 0, where closeness returns NaN.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_closeness, centrality_average_distance, centrality.

Examples

```
centrality_barycenter(regulation_net)
```

---

centrality_beta_measure

Beta Measure

Description

The beta measure, or BG-index (van den Brink and Gilles 2000), gives node i the sum, over its successors j, of one divided by the in-degree of j. Each node with predecessors thus shares one unit of domination power equally among them:

$$\beta_i = \sum_{j : i \to j} \frac{1}{d_j^{in}}.$$

The negative variant applies the measure to the reversed network (Boldi and Vigna 2014).

Usage

```
centrality_beta_measure(x, beta_direction = "positive", ...)
```

Arguments

x	Network input accepted by centrality.
beta_direction	"positive" (default) credits the sources of arcs. "negative" credits their targets.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple unweighted network with direction kept, so weights, loops and parallel edges are ignored. An undirected edge is a pair of reciprocal arcs, so on an undirected network both variants equal the sum of the reciprocal degrees of the neighbors. A node without successors has positive score zero, and a node without predecessors has negative score zero. The positive scores sum to the number of nodes with nonzero in-degree, and the negative scores sum to the number of nodes with nonzero out-degree.

Value

A named numeric vector with one score per node, in input node order.

References

van den Brink, R. and Gilles, R. P. (2000). Measuring domination in directed networks. Social Networks, 22, 141-157. doi:10.1016/S03788733(00)000198.

Boldi, P. and Vigna, S. (2014). Axioms for centrality. Internet Mathematics, 10, 222-262. doi:10.1080/15427951.2013.865686.

See Also

centrality_indegree, centrality_prestige_domain, centrality.

Examples

```
centrality_beta_measure(regulation_net)
```

---

centrality_betweenness

Betweenness Centrality

Description

Betweenness centrality (Freeman 1977) sums, over pairs of other nodes, the fraction of shortest paths between them that pass through the node:

$$B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ the number of those through v.

Usage

```
centrality_betweenness(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1), cutoff (largest path length considered, default -1 for no limit) and normalized (default FALSE).

Details

On a directed network the sum runs over ordered pairs along the edge direction, and on an undirected network over unordered pairs. Edge weights are read as path lengths, and invert_weights = TRUE uses $1/w^{\alpha}$ instead. weighted = FALSE uses hop counts. cutoff drops paths longer than the given length. normalized = TRUE divides the scores by their maximum.

Value

A named numeric vector with one score per node, in input node order.

References

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. Sociometry, 40(1), 35-41. doi:10.2307/3033543.

See Also

centrality_load, centrality_stress, centrality.

Examples

```
centrality_betweenness(regulation_net)
```

---

centrality_bottleneck *Bottleneck Centrality*

Description

Bottleneck centrality (Przulj, Wigle and Jurisica 2004) counts the shortest-path trees in which a node is a bottleneck. For each source s, every shortest path from s to every reachable node is enumerated, and a node $v \neq s$ scores one for that source when it lies on more than $n/4$ of these paths.

Usage

```
centrality_bottleneck(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Distances are hop counts, so edge weights are ignored. mode sets the direction in which the paths leave the source. Paths that end at v count toward its total. A network with one node gives that node a score of one.

Value

A named integer vector with one score per node, in input node order.

References

Przulj, N., Wigle, D. A., & Jurisica, I. (2004). Functional topology in a network of protein interactions. Bioinformatics, 20(3), 340-348. doi:10.1093/bioinformatics/btg415.

See Also

centrality_stress, centrality_betweenness, centrality.

Examples

```
centrality_bottleneck(regulation_net)
```

---

centrality_bridging *Bridging Centrality*

Description

Bridging centrality (Hwang et al. 2008) is the product of betweenness $B(v)$ and the bridging coefficient, which compares the inverse degree of a node with the inverse degrees of its neighbors:

$$BrC(v) = B(v)\frac{1/k_v}{\sum_{u \in N(v)} 1/k_u}.$$

High values mark nodes that lie on many shortest paths and connect densely linked regions.

Usage

```
centrality_bridging(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as directed and normalized.

Details

The betweenness factor reads edge weights as path lengths and follows the edge direction of a directed network. weighted = FALSE uses hop counts, and invert_weights has no effect. The degrees are total degrees, and on a directed network a reciprocated neighbor enters the sum twice. An isolated node scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Hwang, W., Kim, T., Ramanathan, M., & Zhang, A. (2008). Bridging centrality: Graph mining from element level to group level. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 336-344). doi:10.1145/1401890.1401934.

See Also

centrality_local_bridging, centrality_betweenness, centrality.

Examples

```
centrality_bridging(regulation_net)
```

---

centrality_bridging_capital
Bridging Capital

Description

Bridging capital (Jackson 2020, section 3.3) credits node i with the expected number of walks of length at most T that are lost when one entry P_{ij} of the transmission matrix is deleted, summed over j and weighted by source-destination values v_{st}:

$$Brid_i = \sum_j \sum_{s,t} v_{st} \sum_{h=1}^{T} \left[P^h - (P - P_{ij}E_{ij})^h \right]_{st}.$$

Usage

```
centrality_bridging_capital(x, bridging_steps = 2, bridging_values = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
bridging_steps	Horizon T, a nonnegative integer. Default 2.
bridging_values	Nonnegative n by n matrix of source-destination values v. NULL (default) sets every value to one. Row and column names, when present, are matched to the node names.
...	Further arguments to centrality. The measure uses weighted (use edge weights, default TRUE) and normalized (divide by the maximum, default FALSE).

Details

Edge weights are the transmission probabilities in P and must lie between zero and one, and an unweighted edge has probability one. A weight above one raises an error. Direction and loops are kept, and the two entries of an undirected edge are deleted separately. Walks may repeat nodes and edges, and a walk that uses the deleted entry several times is counted once. Isolated nodes score zero, and bridging_steps = 0 gives zero scores. The function computes the expected walk count EInf of the source paper.

Value

A named numeric vector with one score per node, in input node order.

References

Jackson, M. O. (2020). A typology of social capital and associated network measures. Social Choice and Welfare, 54, 311-336. doi:10.1007/s00355019011893.

See Also

centrality_bridging, centrality_diffusion_centrality, centrality.

Examples

```
centrality_bridging_capital(regulation_net)
```

---

centrality_brokerage_coordinator

Coordinator Brokerage

Description

Coordinator brokerage (Gould and Fernandez 1989) counts the open two-paths $a \to v \to c$ through node v in which a, v and c all belong to the same group. A two-path is open when the network has no edge from a to c. This role is w_I in the notation of the source.

Usage

```
centrality_brokerage_coordinator(x, membership = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
membership	Group labels, one per node.
...	Further arguments to centrality, such as normalized.

Details

The measure is defined for directed networks. On an undirected network it returns NA with a cograph_undefined_measure warning, and the same happens when membership is missing, where the warning also has class cograph_bad_membership. A membership whose length differs from the number of nodes raises a cograph_bad_membership error. Group labels may be numbers or strings. Edge weights and self-loops are ignored. The other four roles are centrality_brokerage_itinerant, centrality_brokerage_representative, centrality_brokerage_gatekeeper and centrality_brokerage_liaison.

Value

A named integer vector with one count per node, in input node order. normalized = TRUE returns a numeric vector.

References

Gould, R. V., & Fernandez, R. M. (1989). Structures of mediation: A formal approach to brokerage in transaction networks. Sociological Methodology, 19, 89-126. doi:10.2307/270949.

See Also

centrality_brokerage_gatekeeper, centrality_gateway, centrality.

Examples

```
centrality_brokerage_coordinator(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_brokerage_gatekeeper

Gatekeeper Brokerage

Description

Gatekeeper brokerage (Gould and Fernandez 1989) counts the open two-paths $a \to v \to c$ through node v in which v and c belong to the same group and a to another group. The broker admits contact from outside to a member of its own group. A two-path is open when the network has no edge from a to c. This role is b_{OI} in the notation of the source.

Usage

```
centrality_brokerage_gatekeeper(x, membership = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
membership	Group labels, one per node.
...	Further arguments to centrality, such as normalized.

Details

The measure is defined for directed networks. On an undirected network it returns NA with a cograph_undefined_measure warning, and the same happens when membership is missing, where the warning also has class cograph_bad_membership. A membership whose length differs from the number of nodes raises a cograph_bad_membership error. Group labels may be numbers or strings. Edge weights and self-loops are ignored.

Value

A named integer vector with one count per node, in input node order. normalized = TRUE returns a numeric vector.

References

Gould, R. V., & Fernandez, R. M. (1989). Structures of mediation: A formal approach to brokerage in transaction networks. Sociological Methodology, 19, 89-126. doi:10.2307/270949.

See Also

centrality_brokerage_representative, centrality_brokerage_coordinator, centrality.

Examples

```
centrality_brokerage_gatekeeper(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_brokerage_itinerant

Itinerant Brokerage

Description

Itinerant brokerage (Gould and Fernandez 1989), also called the consultant role, counts the open two-paths $a \to v \to c$ through node v in which a and c belong to the same group and v to another group. A two-path is open when the network has no edge from a to c. This role is w_O in the notation of the source.

Usage

```
centrality_brokerage_itinerant(x, membership = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
membership	Group labels, one per node.
...	Further arguments to centrality, such as normalized.

Details

The measure is defined for directed networks. On an undirected network it returns NA with a cograph_undefined_measure warning, and the same happens when membership is missing, where the warning also has class cograph_bad_membership. A membership whose length differs from the number of nodes raises a cograph_bad_membership error. Group labels may be numbers or strings. Edge weights and self-loops are ignored.

Value

A named integer vector with one count per node, in input node order. normalized = TRUE returns a numeric vector.

References

Gould, R. V., & Fernandez, R. M. (1989). Structures of mediation: A formal approach to brokerage in transaction networks. Sociological Methodology, 19, 89-126. doi:10.2307/270949.

See Also

centrality_brokerage_coordinator, centrality_brokerage_liaison, centrality.

Examples

```
centrality_brokerage_itinerant(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_brokerage_liaison

Liaison Brokerage

Description

Liaison brokerage (Gould and Fernandez 1989) counts the open two-paths $a \to v \to c$ through node v in which a, v and c belong to three different groups. A two-path is open when the network has no edge from a to c. This role is b_O in the notation of the source.

Usage

```
centrality_brokerage_liaison(x, membership = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
membership	Group labels, one per node.
...	Further arguments to centrality, such as normalized.

Details

The measure is defined for directed networks. On an undirected network it returns NA with a cograph_undefined_measure warning, and the same happens when membership is missing, where the warning also has class cograph_bad_membership. A membership whose length differs from the number of nodes raises a cograph_bad_membership error. Group labels may be numbers or strings. With fewer than three groups every count is 0. Edge weights and self-loops are ignored.

Value

A named integer vector with one count per node, in input node order. normalized = TRUE returns a numeric vector.

References

Gould, R. V., & Fernandez, R. M. (1989). Structures of mediation: A formal approach to brokerage in transaction networks. Sociological Methodology, 19, 89-126. doi:10.2307/270949.

See Also

centrality_brokerage_itinerant, centrality_brokerage_coordinator, centrality.

Examples

```
centrality_brokerage_liaison(regulation_net, membership = rep(1:3, length.out = 10))
```

---

centrality_brokerage_representative
Representative Brokerage

Description

Representative brokerage (Gould and Fernandez 1989) counts the open two-paths $a \to v \to c$ through node v in which a and v belong to the same group and c to another group. The broker passes contact from its own group to the outside. A two-path is open when the network has no edge from a to c. This role is b_{IO} in the notation of the source.

Usage

```
centrality_brokerage_representative(x, membership = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
membership	Group labels, one per node.
...	Further arguments to centrality, such as normalized.

Details

The measure is defined for directed networks. On an undirected network it returns NA with a cograph_undefined_measure warning, and the same happens when membership is missing, where the warning also has class cograph_bad_membership. A membership whose length differs from the number of nodes raises a cograph_bad_membership error. Group labels may be numbers or strings. Edge weights and self-loops are ignored.

Value

A named integer vector with one count per node, in input node order. normalized = TRUE returns a numeric vector.

References

Gould, R. V., & Fernandez, R. M. (1989). Structures of mediation: A formal approach to brokerage in transaction networks. Sociological Methodology, 19, 89-126. doi:10.2307/270949.

See Also

centrality_brokerage_gatekeeper, centrality_brokerage_coordinator, centrality.

Examples

```
centrality_brokerage_representative(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_cda *Clustering Degree Algorithm*

Description

The clustering degree algorithm (CDA; Wang et al. 2018) scores a node by its clustering degree $CD_i = [\alpha d_i + (1-\alpha) s_i] / [1 + \exp(-C_i^w)]$ plus the clustering degrees of its neighbors, each scaled by the edge weight:

$$PC_i = CD_i + \sum_{j \in N(i)} \frac{w_{ij}}{w_{\max}} CD_j .$$

Here d_i is the degree, s_i the strength, C_i^w the weighted clustering coefficient of Barrat et al., and $w_{\max}$ the largest edge weight in the network.

Usage

```
centrality_cda(x, cda_alpha = 0.5, ...)
```

Arguments

x	Network input accepted by centrality.
cda_alpha	Weight α of the degree against the strength, between 0 and 1. Default 0.5, the value of Wang et al. (2018). A value outside that range raises an error.
...	Further arguments to centrality. The measure uses weighted (default TRUE) and normalized (default FALSE).

Details

The measure works on an undirected weighted network. On a directed network the weights of the two arcs between a pair are added, and weighted = FALSE uses the simple undirected skeleton. Self-loops are removed, and mode and invert_weights have no effect. A node with fewer than two neighbors has clustering 0, and an isolated node scores 0. Negative or non-finite weights raise an error. With cda_alpha = 1 the scores are unchanged when every weight is multiplied by a constant, and with cda_alpha = 0 they scale by that constant. On an unweighted network cda_alpha has no effect.

Value

A named numeric vector with one score per node, in input node order.

References

Wang, Q., Ren, J., Wang, Y., Zhang, B., Cheng, Y., & Zhao, X. (2018). CDA: A Clustering Degree Based Influential Spreader Identification Algorithm in Weighted Complex Network. IEEE Access, 6, 19550-19559. doi:10.1109/ACCESS.2018.2822844.

See Also

centrality_strength, centrality_transitivity, centrality.

Examples

```
centrality_cda(regulation_net)
```

---

centrality_centroid *Centroid Value*

Description

The centroid value (Koschuetzki et al. 2005) compares a node with every other node by the number of nodes each one is closer to. With $\gamma(v, u)$ the number of nodes strictly closer to v than to u,

$$CV(v) = \min_{u} \left[\gamma(v, u) - \gamma(u, v)\right].$$

The minimum includes $u = v$, so the score is at most 0, and values closer to 0 mark more central nodes.

Usage

```
centrality_centroid(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1) and cutoff (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. invert_weights = TRUE uses $1/w^{\alpha}$ as the length, and weighted = FALSE counts hops. mode sets the direction of the paths.

Value

A named numeric vector with one score per node, in input node order.

References

Koschuetzki, D., Lehmann, K. A., Peeters, L., Richter, S., Tenfelde-Podehl, D., & Zlotowski, O. (2005). Centrality indices. In U. Brandes & T. Erlebach (Eds.), Network Analysis: Methodological Foundations (pp. 16-61). Springer. doi:10.1007/9783540319559_3.

See Also

centrality_closeness, centrality_closeness_vitality, centrality.

Examples

```
centrality_centroid(regulation_net)
```

---

centrality_closeness *Closeness Centrality*

Description

Closeness centrality (Sabidussi 1966) is the reciprocal of the total shortest-path distance from a node to the nodes it reaches:

$$C(v) = \frac{1}{\sum_{w \neq v} d(v, w)}.$$

Unreachable nodes are left out of the sum, as in igraph::closeness().

Usage

```
centrality_closeness(x, mode = "all", ...)

centrality_incloseness(x, ...)

centrality_outcloseness(x, ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1), cutoff (largest path length considered, default -1 for no limit) and normalized (default FALSE).

Details

Edge weights are read as path lengths, and invert_weights = TRUE uses $1/w^{\alpha}$ instead. weighted = FALSE uses hop counts. mode = "out" follows paths leaving the node and mode = "in" paths arriving at it. centrality_outcloseness() and centrality_incloseness() are these two forms. A node that reaches no other node returns NaN. normalized = TRUE multiplies each score by the number of other nodes the node reaches, as igraph does.

Value

A named numeric vector with one score per node, in input node order.

References

Sabidussi, G. (1966). The centrality index of a graph. Psychometrika, 31(4), 581-603. doi:10.1007/BF02289527.

See Also

centrality_harmonic, centrality_barycenter, centrality.

Examples

```
centrality_closeness(regulation_net)
```

`centrality_closeness_vitality` *Closeness Vitality*

Description

Closeness vitality (Koschuetzki et al. 2005) is the drop in the Wiener index when a node is removed:

$$CV(v) = W(G) - W(G - v), \qquad W(G) = \sum_{s \neq t} d(s, t).$$

The Wiener index sums the finite distances over ordered pairs, as in `networkx::closeness_vitality()`.

Usage

```
centrality_closeness_vitality(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which is `TRUE` for tna input), `alpha` (inversion exponent, default 1) and `cutoff` (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. `invert_weights = TRUE` uses $1/w^{\alpha}$ as the length, and `weighted = FALSE` counts hops. `mode` sets the direction of the paths. Pairs that are not connected contribute nothing to either index. An isolated node scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Koschuetzki, D., Lehmann, K. A., Peeters, L., Richter, S., Tenfelde-Podehl, D., & Zlotowski, O. (2005). Centrality indices. In U. Brandes & T. Erlebach (Eds.), Network Analysis: Methodological Foundations (pp. 16-61). Springer. doi:10.1007/9783540319559_3.

See Also

`centrality_wiener`, `centrality_centroid`, `centrality`.

Examples

```
centrality_closeness_vitality(regulation_net)
```

`centrality_clusterrank` *ClusterRank*

Description

ClusterRank (Chen et al. 2013) combines the local clustering coefficient c_v of a node with the degrees of its neighbors:

$$CR(v) = c_v \sum_{u \in N(v)} (k_u + 1).$$

Usage

```
centrality_clusterrank(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Edge weights are ignored. `mode` sets both the neighbor set and the degrees, and on a directed network a reciprocated neighbor enters the sum twice under `mode = "all"`. The clustering coefficient ignores edge direction. A node with fewer than two neighbors has no clustering coefficient and returns `NaN`. Chen et al. weight the sum by 10^{-c_v}, and the measure here multiplies by c_v itself, as the centiserve package does.

Value

A named numeric vector with one score per node, in input node order.

References

Chen, D.-B., Gao, H., Lu, L., & Zhou, T. (2013). Identifying influential nodes in large-scale directed networks: The role of clustering. PLoS ONE, 8(10), e77455. doi:10.1371/journal.pone.0077455.

See Also

`centrality_transitivity`, `centrality_expected`, `centrality`.

Examples

```
centrality_clusterrank(regulation_net)
```

`centrality_coleman_theil` *Coleman-Theil Hierarchy Index*

Description

The Coleman-Theil index (Burt 1992) measures how concentrated Burt's dyadic constraint is across the d_i contacts of node i. Let p_{ij} be the share of the mutual tie strength of i invested in j, $c_{ij} = (p_{ij} + \sum_q p_{iq} p_{qj})^2$ the constraint, and r_{ij} the constraint divided by its mean over the contacts. Then

$$H_i = \frac{\sum_{j \in N(i)} r_{ij} \log r_{ij}}{d_i \log d_i}.$$

Usage

```
centrality_coleman_theil(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (use edge weights, default `TRUE`).

Details

The mutual tie strength of a pair is the sum of the weights in both directions, so direction is combined and loops are removed. Edge weights must be finite and nonnegative, and `weighted = FALSE` gives every edge weight one before the directions are combined. The index lies between zero, for equal constraints, and one, for constraint concentrated on one contact. Following Burt's STRUCTURE 4.2 manual (pages 181-183), an isolated node scores zero and a node with one contact scores one. The organizational and oligopoly multipliers of STRUCTURE are fixed at one. A weight range beyond double precision raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Burt, R. S. (1992). Structural Holes: The Social Structure of Competition. Harvard University Press. doi:10.4159/9780674029095.

See Also

`centrality_constraint`, `centrality_effective_size`, `centrality`.

Examples

```
centrality_coleman_theil(regulation_net)
```

`centrality_communicability` *Communicability Centrality*

Description

Total communicability (Estrada and Hatano 2008; Benzi and Klymko 2013) sums the walks of every length that start at a node, a walk of length k weighted by $1/k!$:

$$TC(v) = \sum_{w} \left[e^{A}\right]_{vw} = \sum_{w} \sum_{k=0}^{\infty} \frac{(A^k)_{vw}}{k!}.$$

Usage

```
centrality_communicability(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `directed` and `normalized`.

Details

A is the binary adjacency matrix, so edge weights are ignored. On an undirected network the matrix exponential is formed from the eigendecomposition of the symmetric A. On a directed network it is computed by scaling and squaring with a Pade approximation (Moler and Van Loan 2003), and the scores are the row sums of e^A, the walks that leave each node. `directed = FALSE` gives the undirected reading.

Value

A named numeric vector with one score per node, in input node order.

References

Estrada, E., & Hatano, N. (2008). Communicability in complex networks. Physical Review E, 77(3), 036111. doi:10.1103/PhysRevE.77.036111.

Benzi, M., & Klymko, C. (2013). Total communicability as a centrality measure. Journal of Complex Networks, 1(2), 124-149. doi:10.1093/comnet/cnt007.

Moler, C., & Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Review, 45(1), 3-49. doi:10.1137/S00361445024180.

See Also

`centrality_subgraph`, `centrality_communicability_betweenness`, `centrality`.

Examples

```
centrality_communicability(regulation_net, directed = FALSE)
```

`centrality_communicability_betweenness` *Communicability Betweenness*

Description

Communicability betweenness (Estrada, Higham and Hatano 2009) is the share of the communicability between other pairs of nodes that is lost when the node is removed. With $G = e^A$ and $G^{(r)}$ the same exponential after the edges of r are deleted,

$$\omega_r = \frac{1}{(n-1)(n-2)} \sum_{p \neq q,\ p,q \neq r} \frac{G_{pq} - G^{(r)}_{pq}}{G_{pq}}.$$

Usage

```
centrality_communicability_betweenness(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `directed` and `normalized`.

Details

A is the binary adjacency matrix, so edge weights are ignored. The scores lie between 0 and 1. The measure is defined for undirected networks. On a directed network G and $G^{(r)}$ are computed by scaling and squaring with a Pade approximation, as in `centrality_communicability`, so the measure is defined there as well. `directed = FALSE` gives the undirected reading.

Value

A named numeric vector with one score per node, in input node order.

References

Estrada, E., Higham, D. J., & Hatano, N. (2009). Communicability betweenness in complex networks. Physica A, 388(5), 764-774. doi:10.1016/j.physa.2008.11.011.

See Also

`centrality_communicability`, `centrality_betweenness`, `centrality`.

Examples

```
centrality_communicability_betweenness(regulation_net, directed = FALSE)
```

`centrality_community_based` *Community-Based Centralities*

Description

Three measures that score a node from a partition `membership`. Community-based centrality (Zhao et al. 2015) is $CbC(i) = \sum_w d_{iw} S_w / N$, where d_{iw} counts the links of i into community w of size S_w. Comm centrality (Gupta, Singh and Cherifi 2016) combines intra-community degree k^{in} and inter-community degree k^{out}:

$$CC(i) = (1+\mu_C)\frac{k_i^{in}}{\max_{j\in C} k_j^{in}} R + (1-\mu_C)\left(\frac{k_i^{out}}{\max_{j\in C} k_j^{out}} R\right)^2,$$

with μ_C the mean inter-link fraction in the community of i. Community-based mediator centrality (Tulu, Hou and Younas 2018) is $CbM(i) = H_i\, d_i / \sum_j d_j$, with H_i the base-2 entropy of the links of i over the communities.

Usage

```
centrality_community_based(x, membership = NULL, mode = "all", ...)

centrality_comm_centrality(
  x,
  membership = NULL,
  mode = "all",
  comm_r = "max_intra",
  ...
)

centrality_community_mediator(x, membership = NULL, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`membership`	Community labels, one per node, for example from `detect_communities`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `normalized`.
`comm_r`	Scale R of Comm centrality: `"max_intra"` (default) or a single positive number applied to every community.

Details

Edge weights and self-loops are ignored. Under `mode = "out"` or `mode = "in"` only out-links or in-links count, and the default ignores direction. Higher values mark more central nodes in all three. The default `comm_r = "max_intra"` sets R to the largest intra-community degree of each

community, the choice the source recommends. The prose of Gupta et al. writes μ_C where their equation has $1 + \mu_C$, and the equation is implemented. Nodes linked to one community only score 0 on the mediator measure. Without `membership` each function raises a warning of classes `cograph_bad_membership` and `cograph_undefined_measure` and returns `NA`. A `membership` that is not one non-missing label per node raises an error of class `cograph_bad_membership`, and an invalid `comm_r` raises `cograph_bad_parameter`.

Value

A named numeric vector with one score per node, in input node order.

References

Zhao, Z., Wang, X., Zhang, W., & Zhu, Z. (2015). A community-based approach to identifying influential spreaders. Entropy, 17(4), 2228-2252.

Gupta, N., Singh, A., & Cherifi, H. (2016). Centrality measures for networks with community structure. Physica A, 452, 46-59.

Tulu, M. M., Hou, R., & Younas, T. (2018). Identifying influential nodes based on community structure to speed up the dissemination of information in complex network. IEEE Access, 6, 7390-7401.

See Also

`centrality_community_hub_bridge`, `centrality_participation`, `centrality`.

Examples

```
centrality_community_based(regulation_net, membership = rep(1:2, each = 5))
centrality_comm_centrality(regulation_net, membership = rep(1:2, each = 5))
centrality_community_mediator(regulation_net,
                              membership = rep(1:2, each = 5))
```

---

centrality_community_hub_bridge

Community Hub-Bridge Centrality

Description

Community hub-bridge centrality (Ghalmane, El Hassouni and Cherifi 2019) scores nodes that are hubs inside their community and bridges between communities:

$$CHB(i) = |C_i|\, k_i^{intra} + NNC_i\, k_i^{inter},$$

where $|C_i|$ is the size of the community of i, k_i^{intra} and k_i^{inter} its numbers of links inside and outside that community, and NNC_i the number of other communities it links to.

Usage

```
centrality_community_hub_bridge(x, membership = NULL, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
membership	Community labels, one per node, for example from detect_communities.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Edge weights and self-loops are ignored. Under mode = "out" or mode = "in" only out-links or in-links count, and the default ignores direction. This is the raw form of the original article. Later work by the same group uses a normalized variant with the same name. Without membership the function raises a warning of classes cograph_bad_membership and cograph_undefined_measure and returns NA for every node. A membership that is not one non-missing label per node raises an error of class cograph_bad_membership.

Value

A named numeric vector with one score per node, in input node order.

References

Ghalmane, Z., El Hassouni, M., & Cherifi, H. (2019). Immunization of networks with non-overlapping community structure. Social Network Analysis and Mining, 9, 45.

See Also

centrality_community_based, centrality_modularity_vitality, centrality.

Examples

```
centrality_community_hub_bridge(regulation_net,
                                membership = rep(1:2, each = 5))
```

---

centrality_constraint *Burt's Constraint*

Description

Burt's constraint measures how much a node's ties are concentrated in contacts that are themselves tied to each other. With p_{ij} the proportion of the tie strength of i invested in j,

$$C_i = \sum_{j \neq i} \left(p_{ij} + \sum_{q \neq i,j} p_{iq} p_{qj} \right)^2 .$$

Low constraint marks access to structural holes.

Usage

```
centrality_constraint(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

Ties are symmetrized as $w_{ij} + w_{ji}$ before the proportions are formed, so edge direction is ignored. With weighted = FALSE every tie has weight one. The values match igraph::constraint(). An isolated node returns NaN, and a node whose only tie is a self-loop scores 0.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_effective_size, centrality_bridging, centrality.

Examples

```
centrality_constraint(regulation_net)
```

---

centrality_controlrank

ControlRank Centrality

Description

ControlRank (Zhou, Yu and Lu 2019) is the smallest eigenvalue of the symmetric part of the graph Laplacian $L = D - A$ after the row and column of the node are deleted:

$$CR_i = \lambda_{\min}\left(\left(\frac{L + L^T}{2}\right)_{-i,-i}\right).$$

Larger values rank higher.

Usage

```
centrality_controlrank(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses weighted (use edge weights, default TRUE) and normalized (divide by the maximum, default FALSE).

Details

Edge weights must be finite and nonnegative, `weighted = FALSE` gives every edge weight one, and loops are removed. In a directed network A_{ij} is the arc from i to j, D holds the out-strengths and scores can be negative. On a connected undirected network with at least two nodes all scores are positive, and on a disconnected undirected network every score is zero. A single node scores zero. With `normalized = TRUE` the scores are divided by their maximum when it is positive, so negative directed scores stay negative. For matrix input with very small weights, set `directed = TRUE` explicitly, because symmetry detection is approximate. A weight range beyond double precision or an unresolved spectrum raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Zhou, J., Yu, X. and Lu, J.-A. (2019). Node Importance in Controlled Complex Networks. IEEE Transactions on Circuits and Systems II: Express Briefs, 66(3), 437-441. doi:10.1109/TCSII.2018.2845940.

See Also

`centrality_laplacian`, `centrality_spectralrank`, `centrality`.

Examples

```
centrality_controlrank(regulation_net)
```

`centrality_coreness` *Coreness*

Description

Coreness (Seidman 1983) is the largest k for which a node belongs to the k-core, the maximal subnetwork in which every node has degree at least k. It is found by repeatedly removing the nodes of lowest degree.

Usage

```
centrality_coreness(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `loops` and `normalized`.

Details

Edge weights are ignored. `mode = "in"` and `mode = "out"` peel by in-degree and out-degree, and `mode = "all"` by total degree, in which a reciprocated tie counts twice. A self-loop adds a fixed amount to the degree of its node throughout the peeling. The values match `igraph::coreness()`.

Value

A named numeric vector with one score per node, in input node order.

References

Seidman, S. B. (1983). Network structure and minimum degree. Social Networks, 5(3), 269-287. doi:10.1016/03788733(83)90028X.

See Also

`centrality_degree`, `centrality_s_core`, `centrality`.

Examples

```
centrality_coreness(regulation_net)
```

---

centrality_cross_clique

Cross-Clique Connectivity

Description

Cross-clique connectivity (Faghani and Nguyen 2013) counts the cliques that contain a node. Every complete subnetwork counts, including the node itself and each of its edges, so a node of an isolated triangle scores 4.

Usage

```
centrality_cross_clique(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Edge direction, edge weights and self-loops are ignored.

Value

A named integer vector with one count per node, in input node order.

References

Faghani, M. R., & Nguyen, U. T. (2013). A study of XSS worm propagation and detection mechanisms in online social networks. IEEE Transactions on Information Forensics and Security, 8(11), 1815-1826. doi:10.1109/TIFS.2013.2280884.

See Also

centrality_coreness, centrality_transitivity, centrality.

Examples

```
centrality_cross_clique(regulation_net)
```

---

centrality_current_flow_betweenness

Current-Flow Betweenness

Description

Current-flow betweenness (Brandes and Fleischer 2005), also known as random-walk betweenness (Newman 2005), treats the network as an electrical circuit with the edge weights as conductances. One unit of current is sent between every pair s, t, and the score is the average current that flows through the node:

$$CFB(v) = \frac{2}{(n-1)(n-2)} \sum_{s<t} \frac{1}{2} \sum_{u} w_{vu} \left| p_v^{(st)} - p_u^{(st)} \right|,$$

where $p^{(st)}$ are the node potentials of that pair, taken from the pseudoinverse of the Laplacian, and the inner sum is set to zero for $v = s$ and $v = t$.

Usage

```
centrality_current_flow_betweenness(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as directed and normalized.

Details

The measure is defined for connected undirected networks. On a disconnected network every score is NA, with a cograph_undefined_measure warning. On a directed network the Laplacian is built from the asymmetric weight matrix, and directed = FALSE gives the undirected reading. The potentials and the currents come from the same weights, and weighted = FALSE gives every edge conductance one. The fixed factor $2/((n-1)(n-2))$ is the normalization of networkx::current_flow_betweenness_central

Value

A named numeric vector with one score per node, in input node order.

References

Brandes, U., & Fleischer, D. (2005). Centrality measures based on current flow. In STACS 2005, Lecture Notes in Computer Science, 3404 (pp. 533-544). Springer. doi:10.1007/978354031856-9_44.

Newman, M. E. J. (2005). A measure of betweenness centrality based on random walks. Social Networks, 27(1), 39-54. doi:10.1016/j.socnet.2004.11.009.

See Also

centrality_current_flow_closeness, centrality_betweenness, centrality.

Examples

```
centrality_current_flow_betweenness(regulation_net, directed = FALSE)
```

---

centrality_current_flow_closeness

Current-Flow Closeness

Description

Current-flow closeness (Brandes and Fleischer 2005), equal to the information centrality of Stephenson and Zelen (1989), replaces the shortest-path distance in closeness by the effective resistance R_{vw} of the network read as an electrical circuit with the edge weights as conductances:

$$CFC(v) = \frac{n-1}{\sum_{w \neq v} R_{vw}}, \qquad R_{vw} = L^{+}_{vv} + L^{+}_{ww} - 2L^{+}_{vw},$$

with L^{+} the pseudoinverse of the Laplacian.

Usage

```
centrality_current_flow_closeness(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as directed and normalized.

Details

The measure is defined for connected undirected networks. On a disconnected network every score is NA, with a cograph_undefined_measure warning. On a directed network the Laplacian is built from the asymmetric weight matrix, and directed = FALSE gives the undirected reading. Edge weights are conductances, and weighted = FALSE gives every edge conductance one.

Value

A named numeric vector with one score per node, in input node order.

References

Stephenson, K., & Zelen, M. (1989). Rethinking centrality: Methods and examples. Social Networks, 11(1), 1-37. doi:10.1016/03788733(89)900166.

Brandes, U., & Fleischer, D. (2005). Centrality measures based on current flow. In STACS 2005, Lecture Notes in Computer Science, 3404 (pp. 533-544). Springer. doi:10.1007/978354031856-9_44.

See Also

centrality_current_flow_betweenness, centrality_information, centrality.

Examples

```
centrality_current_flow_closeness(regulation_net, directed = FALSE)
```

---

centrality_dangalchev *Dangalchev Closeness*

Description

Dangalchev closeness, the residual closeness of Dangalchev (2006), sums an exponentially decaying function of the distance to every node:

$$D(v) = \sum_{w} 2^{-d(v,w)}.$$

The sum includes the node itself, which adds one to every score, as in the centiserve package. Unreachable nodes contribute 0.

Usage

```
centrality_dangalchev(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1) and cutoff (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. `invert_weights = TRUE` uses $1/w^{\alpha}$ as the length, and `weighted = FALSE` counts hops. `mode` sets the direction of the paths. The values equal those of `centrality_residual_closeness` and of `centrality_decay` with `decay_parameter = 0.5`.

Value

A named numeric vector with one score per node, in input node order.

References

Dangalchev, C. (2006). Residual closeness in networks. Physica A, 365(2), 556-564. doi:10.1016/j.physa.2005.12.020.

See Also

`centrality_residual_closeness`, `centrality_decay`, `centrality`.

Examples

```
centrality_dangalchev(regulation_net)
```

---

centrality_decay *Decay Centrality*

Description

Decay centrality sums a decay factor δ raised to the distance from the node to every node:

$$D(v) = \sum_{w} \delta^{d(v,w)}.$$

The sum includes the node itself, which adds one to every score, and unreachable nodes contribute 0. Values of δ between 0 and 1 discount distant nodes.

Usage

```
centrality_decay(x, mode = "all", decay_parameter = 0.5, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`decay_parameter`	Decay factor δ. Default 0.5.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which is `TRUE` for tna input), `alpha` (inversion exponent, default 1) and `cutoff` (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. `invert_weights = TRUE` uses $1/w^{\alpha}$ as the length, and `weighted = FALSE` counts hops. `mode` sets the direction of the paths. `decay_parameter` must lie strictly between 0 and 1, and other values raise a `cograph_bad_parameter` error. `centrality_generalized_closeness` computes the same quantity, and `decay_parameter = 0.5` gives `centrality_dangalchev`.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_generalized_closeness`, `centrality_dangalchev`, `centrality`.

Examples

```
centrality_decay(regulation_net)
```

---

centrality_degree *Degree Centrality*

Description

Degree centrality counts the edges incident to each node. With `mode = "in"` it counts incoming edges and with `mode = "out"` outgoing edges. On an undirected network the three modes agree.

Usage

```
centrality_degree(x, mode = "all", ...)

centrality_indegree(x, ...)

centrality_outdegree(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"in"` or `"out"`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Edge weights are ignored; `centrality_strength` sums them instead. `normalized = TRUE` divides the scores by their maximum. `centrality_indegree()` and `centrality_outdegree()` are the `mode = "in"` and `mode = "out"` forms.

Value

A named numeric vector with one score per node, in input node order.

References

Freeman, L. C. (1978). Centrality in social networks conceptual clarification. Social Networks, 1(3), 215-239. doi:10.1016/03788733(78)900217.

See Also

centrality_strength, centrality.

Examples

```
centrality_degree(regulation_net)
```

---

centrality_degree_discount

DegreeDiscountIC and SingleDiscount

Description

DegreeDiscountIC and SingleDiscount (Chen, Wang and Yang 2009) select spreaders for the independent-cascade model one at a time by the largest discounted degree. After each selection every unselected neighbor v of the new seed gains one selected neighbor t_v, and its DegreeDiscountIC degree becomes

$$dd_v = d_v - 2t_v - (d_v - t_v)\, t_v\, p,$$

with propagation probability p. SingleDiscount uses $d_v - t_v$.

Usage

```
centrality_degree_discount(x, discount_p = 0.01, ...)

centrality_single_discount(x, ...)
```

Arguments

x	Network input accepted by centrality.
discount_p	Propagation probability p for DegreeDiscountIC (default 0.01).
...	Further arguments to centrality, such as normalized.

Details

Every node is placed, and the selection order is returned as a score. The first node selected scores 1 and the last $1/n$, so scores lie in $(0, 1]$. Ties are broken by node order, which the source leaves open. Direction, edge weights and self-loops are ignored.

Value

A named numeric vector with one score per node, in input node order.

References

Chen, W., Wang, Y., & Yang, S. (2009). Efficient influence maximization in social networks. Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 199-208.

See Also

centrality_voterank, centrality_ncvoterank, centrality.

Examples

```
centrality_degree_discount(regulation_net)
centrality_single_discount(regulation_net)
```

---

centrality_diffusion *Diffusion Centrality*

Description

Diffusion centrality has two forms, chosen by `diffusion_method`. The `"kandhway_kuri"` form (Kandhway and Kuri 2014) adds the degrees of a node's neighbors to its own degree, both scaled by λ: $DC(v) = \lambda k_v + \lambda \sum_{u \in N(v)} k_u$. The `"power_series"` form sums the rows of the first n powers of the weight matrix:

$$DC(v) = \sum_{w} \left(W + W^2 + \cdots + W^n \right)_{vw}.$$

Usage

```
centrality_diffusion(x, mode = "all", lambda = 1, ...)
```

Arguments

`x`	Network input accepted by centrality.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`lambda`	Scale factor λ of the `"kandhway_kuri"` form. Default 1.
`...`	Further arguments to centrality. The measure uses `diffusion_method` (`"kandhway_kuri"` or `"power_series"`, default `NULL`, which picks by input type) and `loops` (default `TRUE`, `FALSE` for tna input).

Details

The default is `"kandhway_kuri"`, and `"power_series"` for tna input. The `"kandhway_kuri"` form uses binary degrees, so edge weights are ignored, and `mode` sets both the degrees and the neighbor set. On a directed network `mode = "all"` uses total degrees and the undirected neighbor set. `lambda` multiplies every score. The `"power_series"` form uses the edge weights, or ones with `weighted = FALSE`, and ignores `mode` and `lambda`. With `loops = FALSE` the diagonal of W is set to zero. The `"power_series"` values match `tna::centralities(measures = "Diffusion")`.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_expected`, `centrality_diffusion_centrality`, `centrality`.

Examples

```
centrality_diffusion(regulation_net)
```

---

centrality_diffusion_centrality

Diffusion Centrality

Description

Diffusion centrality (Banerjee et al. 2013) counts the walks of length 1 to T that start at a node, each discounted by q per step:

$$DC(A; q, T) = \sum_{t=1}^{T} (qA)^t \mathbf{1}.$$

Walks may revisit nodes and return to the source, so the score counts repeated hearings of a message.

Usage

```
centrality_diffusion_centrality(x, diffusion_q = 1, diffusion_steps = 3, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`diffusion_q`	Discount q, between 0 and 1. Default 1.
`diffusion_steps`	Horizon T, a nonnegative integer. Default 3.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `loops` (default `TRUE`) and `normalized` (default `FALSE`).

Details

A holds the edge weights, or ones with `weighted = FALSE`. Directed edges carry information from source to target, so the score uses row sums and `mode` has no effect. Transposing the input gives incoming walks. Self-loops follow `loops`. Negative or non-finite weights raise an error. $T = 0$ gives 0 and $T = 1$ gives q times the out-strength. When every entry of qA lies between 0 and 1 the score is the expected number of times the information is heard (Banerjee et al. 2013), and it can exceed the number of nodes. The defaults $q = 1$ and $T = 3$ are package choices. This measure differs from `centrality_diffusion`.

Value

A named numeric vector with one score per node, in input node order.

References

Banerjee, A., Chandrasekhar, A. G., Duflo, E., & Jackson, M. O. (2013). The Diffusion of Microfinance. Science, 341, 1236498. doi:10.1126/science.1236498.

Banerjee, A., Chandrasekhar, A. G., Duflo, E., & Jackson, M. O. (2019). Using Gossips to Spread Information: Theory and Evidence from Two Randomized Controlled Trials. Review of Economic Studies, 86, 2453-2490. doi:10.1093/restud/rdz008.

See Also

`centrality_dynamics_sensitive`, `centrality_diffusion`, `centrality`.

Examples

```
centrality_diffusion_centrality(regulation_net)
```

---

centrality_dil *Degree and Importance of Lines*

Description

The degree and importance of lines (DIL; Liu et al. 2016) adds to the degree of a node a share of the importance of every line incident to it. A line e_{ij} that lies on p triangles has importance $I_{ij} = (k_i - p - 1)(k_j - p - 1)/\lambda$ with $\lambda = p/2 + 1$, and this importance is split between the endpoints in proportion to their degrees minus one:

$$L_i = k_i + \sum_{j \in \Gamma_i} I_{ij} \frac{k_i - 1}{k_i + k_j - 2}.$$

Usage

```
centrality_dil(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Every score is at least the degree of the node, and isolated nodes score 0. The share is undefined only on an isolated edge, whose importance is zero. That share is set to zero, so both endpoints score 1. The text layer of the published article reads λ as $2p + 1$. The printed page and the worked example of the article give $p/2 + 1$, which is the value used here. normalized = TRUE divides the scores by their maximum.

Value

A named numeric vector with one score per node, in input node order.

References

Liu, J., Xiong, Q., Shi, W., Shi, X. and Wang, K. (2016). Evaluating the importance of nodes in complex networks. Physica A: Statistical Mechanics and its Applications, 452, 209-219. doi:10.1016/j.physa.2016.02.049.

See Also

centrality_lhc, centrality_hcc, centrality.

Examples

```
centrality_dil(regulation_net)
```

---

centrality_distance_entropy

Distance Entropy

Description

Distance entropy (Stella and De Domenico 2018) is the Shannon entropy of the distribution of hop distances from a node to the nodes it reaches, scaled by the logarithm of the number of distance values in its range. With p_k the share of reachable nodes at distance k, and m_i and M_i the smallest and largest distance,

$$h_i = -\frac{1}{\log(M_i - m_i + 1)} \sum_{k=m_i}^{M_i} p_k \log p_k.$$

Usage

```
centrality_distance_entropy(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Distances are hop counts, so edge weights are ignored. On a directed network mode sets the direction of the paths. Scores lie between 0 and 1. A node whose reachable nodes all lie at one distance scores 0, and a node that reaches no other node returns NaN. The source divides by $\log(M_i - m_i)$, which is zero when the range holds two distances. The implementation divides by $\log(M_i - m_i + 1)$, so a uniform distribution scores 1.

Value

A named numeric vector with one score per node, in input node order.

References

Stella, M., & De Domenico, M. (2018). Distance entropy cartography characterises centrality in complex networks. Entropy, 20(4), 268.

See Also

centrality_local_dimension, centrality_closeness, centrality.

Examples

```
centrality_distance_entropy(regulation_net)
```

---

centrality_diversity *Diversity Centrality*

Description

Diversity centrality (Eagle, Macy and Claxton 2010) is the Shannon entropy of the weights on the edges incident to a node, divided by its maximum $\log_2 k_v$:

$$D(v) = -\frac{\sum_j p_{vj} \log_2 p_{vj}}{\log_2 k_v}, \qquad p_{vj} = \frac{|w_{vj}|}{\sum_l |w_{vl}|}.$$

The score lies between 0 and 1 and reaches 1 when the weights are equal.

Usage

```
centrality_diversity(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as directed and normalized.

Details

On a directed network incoming and outgoing edges are separate entries, and k_v counts both. With weighted = FALSE, or on an unweighted input, every node with two or more edges scores 1. A node with fewer than two edges, or with edge weights summing to zero, scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Eagle, N., Macy, M., & Claxton, R. (2010). Network diversity and economic development. Science, 328(5981), 1029-1031. doi:10.1126/science.1186605.

See Also

centrality_entropy, centrality_strength, centrality.

Examples

```
centrality_diversity(regulation_net)
```

---

centrality_dkgm *DK-Based Gravity Model*

Description

The DK-based gravity model (Li and Huang 2021) sums, over the partners within a hop-distance radius R, the product of the two nodes' masses divided by their squared distance. The mass $DK(i) = k(i) + k_s^*(i)$ adds the degree to an improved k-shell index $k_s^*(i) = k_s(i) + p(i)/(\max_k q(k) + 1)$, where $p(i)$ is the peeling stage at which the node leaves its shell and $q(k)$ the number of stages shell k needs.

$$DKGM_i = \sum_{j \neq i,\ d(i,j) \leq R} \frac{DK(i)\, DK(j)}{d(i,j)^2}$$

Usage

```
centrality_dkgm(x, dkgm_radius = 2, ...)
```

Arguments

x	Network input accepted by centrality.
dkgm_radius	Hop-distance radius R, a nonnegative number (default 2, a value the source recommends). NULL or Inf includes every reachable partner. "auto" uses half the mean finite hop distance, rounded to the nearest integer and at least one, following the source's equation 4.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. The k-shell peeling removes nodes of degree at most k, which is the reading that terminates and reproduces the source's Tables 2 to 5, so an isolate falls in the one-shell. Unreachable partners contribute nothing, and isolates and a single-node graph score zero. The stage denominator $\max_k q(k) + 1$ is a global maximum, so adding a disconnected component can change every score. A dkgm_radius below one gives zero at every node.

Value

A named numeric vector with one score per node, in input node order.

References

Li, Z. and Huang, X. (2021). Identifying influential spreaders in complex networks by an improved gravity model. Scientific Reports, 11, 22194. doi:10.1038/s41598021012181.

See Also

centrality_mcgm, centrality_mixed_gravity, centrality.

Examples

```
centrality_dkgm(regulation_net)
```

---

centrality_dmnc *Density of Maximum Neighborhood Component*

Description

The density of maximum neighborhood component (Lin et al. 2008) looks at the subnetwork induced by the neighbors of a node, the node itself left out, and takes its largest connected component with E edges and N nodes:

$$DMNC(v) = \frac{E}{N^{\varepsilon}}.$$

A node without neighbors scores 0.

Usage

```
centrality_dmnc(x, mode = "all", dmnc_epsilon = 1.7, ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
dmnc_epsilon	Exponent ε. Default 1.7, the value Lin et al. (2008) recommend. The centiserve package uses 1.67.
...	Further arguments to centrality, such as normalized.

Details

Edge weights are ignored, and mode sets the neighbor set. On an undirected network the result follows this definition. On a directed network the component is a strongly connected component of the induced subnetwork, E counts its directed edges, and a reciprocated neighbor enters the subnetwork once. When several components share the largest size, E counts the edges among all of their nodes. A dmnc_epsilon that is not a single positive finite number raises a cograph_bad_parameter error.

Value

A named numeric vector with one score per node, in input node order.

References

Lin, C.-Y., Chin, C.-H., Wu, H.-H., Chen, S.-H., Ho, C.-W., & Ko, M.-T. (2008). Hubba: Hub objects analyzer, a framework of interactome hubs identification for network biology. Nucleic Acids Research, 36(suppl 2), W438-W443. doi:10.1093/nar/gkn257.

See Also

centrality_mnc, centrality_mcc, centrality.

Examples

```
centrality_dmnc(regulation_net, directed = FALSE)
```

---

centrality_dynamical_importance

Dynamical Importance

Description

Dynamical importance (Restrepo et al. 2006) is the relative drop in the spectral radius ρ of the adjacency matrix when the node is removed:

$$I_i = \frac{\rho(A) - \rho(A_{-i})}{\rho(A)}.$$

The spectral radius is recomputed after each deletion, in place of the eigenvector approximation of the paper (eq. 5).

Usage

```
centrality_dynamical_importance(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses weighted (default TRUE) and normalized (default FALSE).

Details

A holds the edge weights, or ones with weighted = FALSE, and self-loops are removed. Negative or non-finite weights raise an error. The scores lie between 0 and 1 and are unchanged when every arc is reversed, so mode has no effect. A network with spectral radius 0, such as any directed acyclic network, gives NaN for every node with a cograph_undefined_measure warning. An isolated node in a network with positive spectral radius scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Restrepo, J. G., Ott, E., & Hunt, B. R. (2006). Characterizing the Dynamical Importance of Network Nodes and Links. Physical Review Letters, 97, 094102. doi:10.1103/PhysRevLett.97.094102.

See Also

centrality_eigenvector, centrality_resistance_curvature, centrality.

Examples

```
centrality_dynamical_importance(regulation_net)
```

`centrality_dynamics_sensitive` *Dynamics-Sensitive Centrality*

Description

Dynamics-sensitive centrality (Liu et al. 2016) is a linearized cumulative spreading score with spreading rate β and recovery rate μ over T steps:

$$S(T) = \sum_{r=0}^{T-1} \beta A \left[\beta A + (1-\mu) I\right]^r \mathbf{1}.$$

With $\mu = 1$ it reduces to $\sum_{t=1}^{T} (\beta A)^t \mathbf{1}$, the form listed in the Centrality Zoo, and $\mu = 0$ gives the susceptible-infected case.

Usage

```
centrality_dynamics_sensitive(x, ds_beta = 0.1, ds_mu = 1, ds_steps = 5, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`ds_beta`	Spreading rate β, between 0 and 1. Default 0.1.
`ds_mu`	Recovery rate μ, between 0 and 1. Default 1.
`ds_steps`	Horizon T, a nonnegative integer. Default 5.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Isolated nodes score 0. $T = 0$ or $\beta = 0$ gives 0, and $T = 1$ gives β times the degree. The score counts repeated walks and can exceed the number of nodes, so it is not an infection probability. The defaults $\beta = 0.1$, $\mu = 1$ and $T = 5$ are one setting studied by Liu et al. (2016). Parameters outside their ranges raise an error.

Value

A named numeric vector with one score per node, in input node order.

References

Liu, J. G., Lin, J. H., Guo, Q., & Zhou, T. (2016). Locating influential nodes via dynamics-sensitive centrality. Scientific Reports, 6, 21380. doi:10.1038/srep21380.

See Also

`centrality_diffusion_centrality`, `centrality`.

Examples

```
centrality_dynamics_sensitive(regulation_net)
```

`centrality_eccentricity` *Eccentricity*

Description

The eccentricity of a node (Hage and Harary 1995) is its largest shortest-path distance to a node it reaches:

$$e(v) = \max_{w:\, d(v,w)<\infty} d(v, w).$$

Lower values mark more central nodes.

Usage

```
centrality_eccentricity(x, mode = "all", ...)

centrality_ineccentricity(x, ...)

centrality_outeccentricity(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `directed` and `normalized`.

Details

Edge weights are read as path lengths, and `invert_weights` has no effect. `weighted = FALSE`, or an unweighted input, gives hop counts. `mode = "out"` follows paths leaving the node and `mode = "in"` paths arriving at it. `centrality_outeccentricity()` and `centrality_ineccentricity()` are these two forms. A node that reaches no other node scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Hage, P., & Harary, F. (1995). Eccentricity and centrality in networks. Social Networks, 17(1), 57-63. doi:10.1016/03788733(94)002489.

See Also

`centrality_closeness`, `centrality_radiality`, `centrality`.

Examples

```
centrality_eccentricity(regulation_net)
```

`centrality_effective_size` *Effective Size*

Description

Burt's effective size is the number of a node's contacts minus their redundancy, the average number of ties each contact has to the other contacts:

$$ES(v) = k_v - \frac{1}{k_v} \sum_{j \in N(v)} |N(v) \cap N(j)|.$$

On an undirected network this is $k_v - 2t_v/k_v$, with t_v the number of ties among the contacts.

Usage

```
centrality_effective_size(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `directed` and `normalized`.

Details

Edge weights are ignored. On a directed network each reciprocated neighbor enters the neighbor list twice, so k_v counts it twice and the result differs from that of the undirected skeleton. An isolated node scores 0.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_constraint`, `centrality_redundancy`, `centrality`.

Examples

```
centrality_effective_size(regulation_net)
```

`centrality_ehcc` *Extended Hybrid Characteristic Centrality*

Description

Extended hybrid characteristic centrality (Liu and Zheng 2023) adds to a node's `centrality_hcc` score the scores of its neighbors.

$$EHCC(u) = HCC(u) + \sum_{v \in \phi(u)} HCC(v)$$

Usage

```
centrality_ehcc(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. The measure uses `hcc_delta` (weight δ of the node's own degree in the extended degree, default 0.5).

Details

The measure inherits the conventions of `centrality_hcc`. It is computed on the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and a `hcc_delta` outside $[0, 1]$ raises a `cograph_bad_parameter` error. Scores lie in $[0, 2(1+k_{max})]$, and an isolate scores its own HCC.

Value

A named numeric vector with one score per node, in input node order.

References

Liu, J. and Zheng, J. (2023). Identifying important nodes in complex networks based on extended degree and E-shell hierarchy decomposition. Scientific Reports, 13, 3197. doi:10.1038/s41598023-303085.

See Also

`centrality_hcc`, `centrality`.

Examples

```
centrality_ehcc(regulation_net)
```

`centrality_eigenvector` *Eigenvector Centrality*

Description

Eigenvector centrality (Bonacich 1972) scores a node by the scores of the nodes that point to it. The scores form the dominant eigenvector of the transposed weight matrix:

$$\lambda x = A^T x.$$

The vector is scaled to a maximum of one.

Usage

```
centrality_eigenvector(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `directed`.

Details

A holds the edge weights, or ones with `weighted = FALSE`. On a directed network a node gains standing from its incoming edges. The scores lie between 0 and 1. A network without edges gives every node a score of one.

Value

A named numeric vector with one score per node, in input node order.

References

Bonacich, P. (1972). Factoring and weighting approaches to status scores and clique identification. Journal of Mathematical Sociology, 2(1), 113-120. doi:10.1080/0022250X.1972.9989806.

See Also

`centrality_pagerank`, `centrality_alpha`, `centrality_authority`, `centrality`.

Examples

```
centrality_eigenvector(regulation_net)
```

`centrality_entropy` *Entropy Centrality*

Description

Entropy centrality, in the form of the centiserve package, removes the node and measures how evenly reachability is spread over the remaining network. With r_j the number of nodes that node j reaches in $G - v$ and P half the total of the r_j,

$$H(v) = -\sum_j y_j \log_2 y_j, \qquad y_j = \frac{r_j}{P}.$$

Terms with $y_j = 0$ are dropped.

Usage

```
centrality_entropy(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Edge weights are ignored, and `mode` sets the direction of reachability. When the network stays strongly connected after the removal of any one node, every node scores $2\log_2((n-1)/2)$, as on `regulation_net`.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_distance_entropy`, `centrality_diversity`, `centrality`.

Examples

```
centrality_entropy(regulation_net)
```

`centrality_entropy_variation` *Entropy Variation*

Description

Entropy variation (Ai 2017) is the change in the Shannon entropy of a node-level distribution f when a node and its links are removed:

$$EnV_f(i) = I_f(G) - I_f(G - i), \qquad I_f(G) = -\sum_j p_j \ln p_j, \quad p_j = \frac{f(j)}{\sum_l f(l)}.$$

The distribution f is the degree or the betweenness. Higher values mark more important nodes.

Usage

```
centrality_entropy_variation(
  x,
  of = c("degree", "betweenness"),
  mode = "all",
  ...
)
```

Arguments

`x`	Network input accepted by `centrality`.
`of`	Distribution: `"degree"` (default) or `"betweenness"`.
`mode`	For the degree variant on directed networks: `"all"` (default, in plus out), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The degree variant uses `loops` (keep self-loops, default `TRUE`).

Details

The entropy uses the natural logarithm, as in the code of the author, so scores are in nats. The difference is signed, and a negative value means that removing the node evens out the distribution. Edge weights are ignored in both variants. For the degree variant `mode` selects the in-, out- or total degree on a directed network, and self-loops count toward the degree, and `loops = FALSE` drops them. The betweenness variant ignores `mode` and self-loops. A deletion that leaves every value of f at zero, such as betweenness on a clique, has entropy 0.

Value

A named numeric vector with one score per node, in input node order.

References

Ai, X. (2017). Node importance ranking of complex networks with entropy variation. Entropy, 19(7), 303.

See Also

centrality_betweenness, centrality_distance_entropy, centrality.

Examples

```
centrality_entropy_variation(regulation_net)
```

---

centrality_exogenous *Exogenous Centrality*

Description

Exogenous centrality (Everett and Borgatti 2010) is the total change in a base centrality of the other nodes when the node is deleted:

$$E(i) = \sum_{j \neq i} \left[C_G(j) - C_{G-i}(j) \right].$$

The default base is reverse closeness, $C_H(j) = \sum_{k \neq j} \max(N - d_H(j,k), 0)$, with N the number of nodes of the original network. The other bases are raw betweenness and degree.

Usage

```
centrality_exogenous(
  x,
  mode = "all",
  exogenous_base = "reverse_closeness",
  ...
)
```

Arguments

x	Network input accepted by centrality.
mode	Direction of the base measure: "all" (default), "out" or "in".
exogenous_base	Base centrality: "reverse_closeness" (default), "betweenness" or "degree".
...	Further arguments to centrality, such as normalized.

Details

The measure uses the simple binary network, so weights, loops and parallel edges are ignored. mode = "all" uses the undirected skeleton, and "out" and "in" use directed paths and degrees. On an undirected network the three modes agree and the degree base returns the degree. On a directed network the out-degree base returns the in-degree and the in-degree base the out-degree. Exogenous betweenness can be negative when a deletion raises the betweenness of the remaining nodes. An isolated node scores 0. normalized = TRUE divides by the largest score.

Value

A named numeric vector with one score per node, in input node order.

References

Everett, M. G., & Borgatti, S. P. (2010). Induced, endogenous and exogenous centrality. Social Networks, 32(4), 339-344. doi:10.1016/j.socnet.2010.06.004.

See Also

centrality_closeness_vitality, centrality_betweenness, centrality.

Examples

```
centrality_exogenous(regulation_net)
```

---

centrality_expected *Expected Centrality*

Description

Expected centrality is the sum of the degrees of a node's neighbors:

$$E(v) = \sum_{u \in N(v)} k_u.$$

Usage

```
centrality_expected(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Edge weights are ignored. mode sets both the degrees and the neighbor set. On a directed network mode = "all" uses total degrees and the undirected neighbor set. Adding the node's own degree gives the "kandhway_kuri" form of centrality_diffusion.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_diffusion, centrality_neighborhood_connectivity, centrality.

Examples

```
centrality_expected(regulation_net)
```

---

centrality_expected_force

Expected Force Centrality

Description

The Expected Force (Lawyer 2015, equation 1) is the entropy of the onward spreading potential after two transmission events from a seed node. Each ordered sequence of two transmissions gives an infected cluster of three nodes with D_k edges to susceptible nodes, and with natural logarithms

$$ExF_i = -\sum_k \frac{D_k}{\sum_l D_l} \log \frac{D_k}{\sum_l D_l}.$$

Usage

```
centrality_expected_force(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple unweighted network with direction kept, so weights, loops and parallel edges are ignored. In a directed network only outgoing arcs transmit and count toward D_k. Different orders of the same two infections count as distinct sequences. A node that cannot start two transmissions scores zero, so isolated nodes and nodes of components with at most three nodes score zero. When every cluster has no edge to a susceptible node the entropy is undefined, and the score is set to zero.

Value

A named numeric vector with one score per node, in input node order.

References

Lawyer, G. (2015). Understanding the influence of all nodes in a network. Scientific Reports, 5, 8665. doi:10.1038/srep08665.

See Also

centrality_modified_expected_force, centrality_expected, centrality.

Examples

```
centrality_expected_force(regulation_net)
```

---

centrality_expected_influence_1
One-Step Expected Influence

Description

One-step expected influence (Robinaugh, Millner and McNally 2016) sums the signed weights of a node's edges:

$$EI_1(i) = \sum_j w_{ij}.$$

Negative edges lower the score, which makes the measure suited to partial-correlation and other signed networks.

Usage

```
centrality_expected_influence_1(x, mode = "out", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "out" (default), "in" or "all".
...	Further arguments to centrality, such as normalized and psych_network.

Details

mode = "out" (the default here) sums the outgoing weights, mode = "in" the incoming weights and mode = "all" both, with a self-loop counted once. On an undirected network the three modes agree and each edge is counted once. With weighted = FALSE, or on an unweighted input, the score is the degree in the chosen mode. When the network has a negative edge, normalized = TRUE divides by the largest absolute score and keeps the sign.

Value

A named numeric vector with one score per node, in input node order.

References

Robinaugh, D. J., Millner, A. J., & McNally, R. J. (2016). Identifying highly influential nodes in the complicated grief network. Journal of Abnormal Psychology, 125(6), 747-757. doi:10.1037/abn0000181.

See Also

centrality_expected_influence_2, centrality_strength, centrality.

Examples

```
centrality_expected_influence_1(regulation_net)
```

---

centrality_expected_influence_2

Two-Step Expected Influence

Description

Two-step expected influence (Robinaugh, Millner and McNally 2016) adds to the one-step expected influence of a node the one-step expected influence of its neighbors, each weighted by the signed edge between them:

$$EI_2(i) = EI_1(i) + \sum_j w_{ij} EI_1(j).$$

Usage

```
centrality_expected_influence_2(x, mode = "out", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "out" (default), "in" or "all".
...	Further arguments to centrality, such as normalized and psych_network.

Details

mode = "out" (the default here) follows outgoing edges at both steps, mode = "in" incoming edges and mode = "all" both, with a self-loop counted once. On an undirected network the three modes agree and each edge is counted once. weighted = FALSE gives every edge weight one. When the network has a negative edge, normalized = TRUE divides by the largest absolute score and keeps the sign.

Value

A named numeric vector with one score per node, in input node order.

References

Robinaugh, D. J., Millner, A. J., & McNally, R. J. (2016). Identifying highly influential nodes in the complicated grief network. Journal of Abnormal Psychology, 125(6), 747-757. doi:10.1037/abn0000181.

See Also

centrality_expected_influence_1, centrality_strength, centrality.

Examples

```
centrality_expected_influence_2(regulation_net)
```

`centrality_extended_coreness`

Extended Neighborhood Coreness

Description

Extended neighborhood coreness (Bae and Kim 2014) sums the neighborhood coreness of every neighbor of a node:

$$C_{nc+}(i) = \sum_{j \in N(i)} \sum_{l \in N(j)} k_s(l),$$

where k_s is the core number in the whole network. The score equals $A^2 k_s$.

Usage

```
centrality_extended_coreness(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Every walk of length two adds the core number of its endpoint, including walks that return to the focal node. Isolated nodes score 0. On a tree the score equals the sum of the neighbors' degrees, and on a d-regular network it equals d^3.

Value

A named numeric vector with one score per node, in input node order.

References

Bae, J., & Kim, S. (2014). Identifying and ranking influential spreaders in complex networks by neighborhood coreness. Physica A, 395, 549-559. doi:10.1016/j.physa.2013.10.047.

See Also

centrality_coreness, centrality_extended_gravity, centrality.

Examples

```
centrality_extended_coreness(regulation_net)
```

`centrality_extended_gravity`
Extended Gravity Centrality

Description

Extended gravity centrality (Ma et al. 2016) is the sum of the gravity scores of a node's neighbors,

$$G^{+}(i) = \sum_{j \in N(i)} G(j), \qquad G(j) = \sum_{l:\, 0 < d_{jl} \le r} \frac{k_s(j)\, k_s(l)}{d_{jl}^2},$$

where k_s is the k-shell index and d the hop distance.

Usage

```
centrality_extended_gravity(x, gravity_radius = 3, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`gravity_radius`	Hop radius r: a nonnegative number (default 3, the value of Ma et al. 2016), `"auto"`, or `NULL` or `Inf` for the whole component.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and the masses are always k-shell indices. The radius applies around each neighbor j, so a contribution can come from $r + 1$ hops away from i, including paths back to i. Radius 0 and isolated nodes give 0. The `"auto"` radius is half the mean finite positive hop distance, rounded to the nearest integer with a minimum of 1. This rule is a package choice. A negative radius raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Ma, L. L., Ma, C., Zhang, H. F., & Wang, B. H. (2016). Identifying influential spreaders in complex networks based on gravity formula. Physica A, 451, 205-212. doi:10.1016/j.physa.2015.12.162.

See Also

`centrality_gravity`, `centrality_extended_coreness`, `centrality`.

Examples

```
centrality_extended_gravity(regulation_net)
```

`centrality_extended_local_bridging`
Extended Local Bridging Centrality

Description

Extended local bridging centrality (Macker 2016) multiplies a node's betweenness in its two-hop ego network by its bridging coefficient, the reciprocal of its degree divided by the sum of the reciprocal degrees of its neighbors. The two-hop ego network contains every node within two hops and every edge among them, so its shortest paths can have up to four edges.

Usage

```
centrality_extended_local_bridging(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized` (divide by the maximum, default `FALSE`).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Degrees are taken from the whole network. Ego betweenness counts unordered pairs, excludes endpoints and is not normalized. Isolated nodes, leaves and every node of a complete graph score zero. The weighted model of Macker (2016), with link quality and path costs, is not implemented.

Value

A named numeric vector with one score per node, in input node order.

References

Macker, J. P. (2016). An improved local bridging centrality model for distributed network analytics. MILCOM, pp. 600-605. doi:10.1109/MILCOM.2016.7795393.

See Also

`centrality_localized_bridging`, `centrality_bridging`, `centrality`.

Examples

```
centrality_extended_local_bridging(regulation_net)
```

`centrality_extended_mixed_gravity` *Extended Mixed Gravitational Centrality*

Description

Extended mixed gravitational centrality (Wang, Li and Xia 2018), also called IGC+, sums the mixed gravitational scores of the immediate neighbors of a node:

$$EMGC_i = \sum_{j \in N(i)} MGC_j.$$

Usage

```
centrality_extended_mixed_gravity(x, gravity_radius = 3, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`gravity_radius`	Hop-distance cutoff of the inner scores. Default 3. `NULL` or `Inf` includes every reachable node.
`...`	Further arguments to `centrality`, such as `normalized` (divide by the maximum, default `FALSE`).

Details

Each inner score MGC_j uses the radius around j, so a contribution can come from up to `gravity_radius` + 1 hops from the focal node. The implementation follows the reproduction in Li and Huang (2022), equation 8. The input handling and radius options of `centrality_mixed_gravity` apply, so direction, weights, loops and parallel edges are ignored. Isolated nodes score zero.

Value

A named numeric vector with one score per node, in input node order.

References

Wang, J., Li, C. and Xia, C. (2018). Improved centrality indicators to characterize the nodal spreading capability in complex networks. Applied Mathematics and Computation, 334, 388-400. doi:10.1016/j.amc.2018.04.028.

Li, Z. and Huang, X. (2022). Identifying influential spreaders by gravity model considering multi-characteristics of nodes. Scientific Reports, 12, 9879. doi:10.1038/s41598022140053.

See Also

`centrality_mixed_gravity`, `centrality_gravity`, `centrality`.

Examples

```
centrality_extended_mixed_gravity(regulation_net)
```

`centrality_flow_betweenness` *Flow Betweenness*

Description

Flow betweenness (Freeman, Borgatti and White 1991) sums, over pairs of other nodes s and t, the flow that passes through the node when a maximum flow is sent from s to t with the edge weights as capacities.

Usage

```
centrality_flow_betweenness(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which is `TRUE` for tna input), `alpha` (inversion exponent, default 1) and `normalized` (default `FALSE`).

Details

The measure requires the igraph package and raises an error of class `cograph_needs_igraph` without it. The flow through a node is read from the maximum flow that `igraph::max_flow()` returns. On a directed network the sum runs over ordered pairs along the edge direction, and on an undirected network over unordered pairs. `weighted = FALSE` gives every edge capacity one, and `invert_weights = TRUE` uses $1/w^{\alpha}$ as the capacity.

Value

A named numeric vector with one score per node, in input node order.

References

Freeman, L. C., Borgatti, S. P., & White, D. R. (1991). Centrality in valued graphs: A measure of betweenness based on network flow. Social Networks, 13(2), 141-154. doi:10.1016/0378-8733(91)90017N.

See Also

`centrality_betweenness`, `centrality_current_flow_betweenness`, `centrality`.

Examples

```
centrality_flow_betweenness(regulation_net)
```

`centrality_gateway` *Gateway Coefficient*

Description

The gateway coefficient (Ruiz Vargas and Wahl 2014) refines the participation coefficient by weighting the links of node i into each module s by how much of the connection between the two modules they carry and by the degree of the neighbors they reach:

$$G_i = 1 - \frac{1}{k_i^2} \sum_s k_{is}^2 \, g_{is}^2,$$

where k_{is} is the number of links of i into module s and g_{is} lies between 0 and 1.

Usage

```
centrality_gateway(x, membership = NULL, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`membership`	Module labels, one per node: integer codes, character labels or a factor.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `directed` and `normalized`.

Details

Edge weights are ignored, and the score lies between 0 and 1. On a directed network `mode` chooses the ties: `"out"` uses outgoing links, `"in"` incoming links and `"all"` (default) both, with a reciprocated tie counted twice. The degree k_i, the module links k_{is} and the neighbors whose degrees enter g_{is} all use the same ties. On an undirected network the three modes agree and equal `brainGraph::gateway_coeff(centr = "degree")`. `membership` can hold integer, character or factor labels. Without `membership` the function returns `NA` with a warning of classes `cograph_bad_membership` and `cograph_undefined_measure`, and a `membership` of the wrong length raises a `cograph_bad_membership` error. With a single module every node scores 0, and so does a node without links in the chosen mode.

Value

A named numeric vector with one score per node, in input node order.

References

Ruiz Vargas, E., & Wahl, L. M. (2014). The gateway coefficient: A novel metric for identifying critical connections in modular networks. The European Physical Journal B, 87(7), 161. doi:10.1140/epjb/e2014408007.

See Also

centrality_participation, centrality_within_module_z, centrality.

Examples

```
centrality_gateway(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_generalized_closeness

Generalized Closeness

Description

Generalized closeness, as in the tidygraph package, sums a decay factor α raised to the distance from the node to every node:

$$GC(v) = \sum_{w} \alpha^{d(v,w)}.$$

The sum includes the node itself, which adds one to every score, and unreachable nodes contribute 0.

Usage

```
centrality_generalized_closeness(x, mode = "all", decay_parameter = 0.5, ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
decay_parameter	Decay factor α of the formula. Default 0.5.
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which is TRUE for tna input), alpha (inversion exponent, default 1) and cutoff (largest path length considered, default -1 for no limit).

Details

Edge weights are read as path lengths. invert_weights = TRUE uses $1/w^{\alpha}$ as the length, with the inversion exponent alpha of centrality, and weighted = FALSE counts hops. mode sets the direction of the paths. The values equal those of centrality_decay with the same decay_parameter, which must lie strictly between 0 and 1; other values raise a cograph_bad_parameter error.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_decay, centrality_harmonic, centrality.

Examples

```
centrality_generalized_closeness(regulation_net)
```

---

centrality_gilschmidt *Gil-Schmidt Power Index*

Description

The Gil-Schmidt power index sums the reciprocal hop distances from a node to the nodes it reaches and divides by $n - 1$:

$$GS(v) = \frac{1}{n-1} \sum_{w \neq v} \frac{1}{d(v, w)}.$$

Unreachable nodes contribute 0, so the score lies between 0 and 1, and a node adjacent to every other node scores 1.

Usage

```
centrality_gilschmidt(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Distances are hop counts, so edge weights are ignored and invert_weights has no effect. mode sets the direction of the paths. With mode = "out" the values match sna::gilschmidt() with its default settings.

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_harmonic, centrality_closeness, centrality.

Examples

```
centrality_gilschmidt(regulation_net)
```

`centrality_global_structure` *Global Structure Model Centrality*

Description

The global structure model (GSM; Ullah et al. 2021) multiplies a coreness factor of the node by the coreness of all other nodes, each discounted by its hop distance:

$$GSM(i) = \exp\left(\frac{k_s(i)}{N}\right) \sum_{j \neq i} \frac{k_s(j)}{d_{ij}}.$$

Here k_s is the core number and N the number of nodes, isolated nodes included.

Usage

```
centrality_global_structure(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and `mode` has no effect. Only reachable nodes contribute to the sum, which extends the measure to disconnected networks. Other components still affect the scores through N. An isolated node scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Ullah, A., Wang, B., Sheng, J., Long, J., Khan, N., & Sun, Z. (2021). Identification of nodes influence based on global structure model in complex networks. Scientific Reports, 11, 6173. doi:10.1038/s4159802184684x.

See Also

`centrality_hybrid_global_structure`, `centrality_improved_global_structure`, `centrality`.

Examples

```
centrality_global_structure(regulation_net)
```

`centrality_graph_regularization` *Graph Regularization Centrality*

Description

Graph regularization centrality (Dal Col and Petronetto 2023) is the reciprocal of the diagonal of the inverse regularized Laplacian, where L is the weighted Laplacian of the undirected network and γ is `grc_gamma`:

$$GRC_i = \frac{1}{[(I + \gamma L)^{-1}]_{ii}}.$$

A larger score means that Laplacian smoothing retains less of a unit impulse placed at the node.

Usage

```
centrality_graph_regularization(x, grc_gamma = 1, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`grc_gamma`	Regularization strength, a finite nonnegative number. Default 1.
`...`	Further arguments to `centrality`. The measure uses `weighted` (use edge weights, default `TRUE`).

Details

The measure is computed on the undirected network with loops removed. For a directed network the weights of opposite arcs are added, and with `weighted = FALSE` the simple undirected skeleton is used. Edge weights must be finite and nonnegative, and a zero weight is an absent edge. At `grc_gamma = 0` every score is one, an isolated node always scores one, and within a component of n nodes the scores lie between one and n. A negative or nonfinite `grc_gamma`, or a weight range beyond double precision, raises an error. The author software (Dal Col 2023) approximates the same filter with ten Chebyshev terms, so its values can differ from the exact inverse computed here.

Value

A named numeric vector with one score per node, in input node order.

References

Dal Col, A., & Petronetto, F. (2023). Graph regularization centrality. Physica A, 628, 129188. doi:10.1016/j.physa.2023.129188.

Dal Col, A. (2023). GRC. Mendeley Data, version 1. doi:10.17632/ns63f5dj86.1.

See Also

`centrality_laplacian`, `centrality_information`, `centrality`.

Examples

```
centrality_graph_regularization(regulation_net)
```

`centrality_gravity` *Gravity Centrality*

Description

Gravity centrality (Ma et al. 2016) treats node masses as attracting each other with a force that falls with the squared hop distance, summed over the nodes within a radius r:

$$G(i) = \sum_{j:\, 0<d_{ij}\le r} \frac{m_i m_j}{d_{ij}^2}.$$

The default uses the k-shell index as mass and $r = 3$ (Ma et al. 2016). Degree mass without truncation is the gravity model of Li et al. (2019, eq. 1), and degree mass with `gravity_radius = "auto"` is their local gravity model (eq. 2).

Usage

```
centrality_gravity(
  x,
  mode = "all",
  gravity_mass = "kshell",
  gravity_radius = 3,
  ...
)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`gravity_mass`	Node mass: `"kshell"` (default), `"degree"` or `"legacy"`.
`gravity_radius`	Largest hop distance included: a number (default 3), `"auto"`, or `NULL` for the whole network.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Distances are hop counts, so edge weights are ignored. `mode` sets the direction of both the distances and the degree or k-shell masses, and `mode = "all"` treats edges as undirected. The `"auto"` radius is half the mean finite positive distance, rounded to the nearest integer with a minimum of 1 (Li et al. 2019, eq. 5). A radius below 1 gives a score of 0 for every node. `gravity_mass = "legacy"` with `gravity_radius = NULL` computes $\sum_j k_j s_j / d_{ij}^2$, with k_j the degree, s_j the k-shell index and no mass on the focal node. This form differs from the formula of Li et al. (2019).

Value

A named numeric vector with one score per node, in input node order.

References

Ma, L.-L., Ma, C., Zhang, H.-F., & Wang, B.-H. (2016). Identifying influential spreaders in complex networks based on gravity formula. Physica A, 451, 205-212.

Li, Z., Ren, T., Ma, X., Liu, S., Zhang, Y., & Zhou, T. (2019). Identifying influential spreaders by gravity model. Scientific Reports, 9, 8387.

See Also

`centrality_extended_gravity`, `centrality_coreness`, `centrality`.

Examples

```
centrality_gravity(regulation_net)
```

---

centrality_harary *Harary Centrality*

Description

Harary centrality sums the inverse squared shortest-path distances from a node to the other nodes:

$$H(i) = \sum_{j \neq i} \frac{1}{d_{ij}^2},$$

with $1/\infty = 0$, so an unreachable node contributes nothing and the score is defined on disconnected networks.

Usage

```
centrality_harary(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which inverts for tna input only), `alpha` (inversion exponent, default 1) and `cutoff` (largest distance counted, default -1 for no limit).

Details

Edge weights are read as distances. On `regulation_net`, whose weights lie below one, the scores are therefore large. `weighted = FALSE` uses hop counts, and `invert_weights = TRUE` converts a weight w to the distance $1/w^{\alpha}$. `mode = "all"` treats edges as undirected, `"out"` uses distances from the node and `"in"` distances to it. The Harary index of Plavsic et al. (1993) sums the inverse distances $1/d_{ij}$; its node-level form is `centrality_harmonic`.

Value

A named numeric vector with one score per node, in input node order.

References

Plavsic, D., Nikolic, S., Trinajstic, N., & Mihalic, Z. (1993). On the Harary index for the characterization of chemical graphs. Journal of Mathematical Chemistry, 12(1), 235-250. doi:10.1007/BF01164638.

See Also

`centrality_harmonic`, `centrality_closeness`, `centrality`.

Examples

```
centrality_harary(regulation_net)
```

`centrality_harmonic` *Harmonic Centrality*

Description

Harmonic centrality (Marchiori and Latora 2000) sums the inverse shortest-path distances from a node to the other nodes:

$$H(i) = \sum_{j \neq i} \frac{1}{d_{ij}},$$

with $1/\infty = 0$, so the score is defined on disconnected networks (Boldi and Vigna 2014). `centrality_inharmonic()` and `centrality_outharmonic()` are the `mode = "in"` and `mode = "out"` forms.

Usage

```
centrality_harmonic(x, mode = "all", ...)

centrality_inharmonic(x, ...)

centrality_outharmonic(x, ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses invert_weights (default NULL, which inverts for tna input only), alpha (inversion exponent, default 1), cutoff (largest distance counted, default -1 for no limit) and normalized (default FALSE).

Details

Edge weights are read as distances, and invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. weighted = FALSE uses hop counts. mode = "all" treats edges as undirected, "out" uses distances from the node and "in" distances to it. The scores equal igraph::harmonic_centrality() on the weighted graph. normalized = TRUE divides the scores by $n-1$, as igraph::harmonic_centrality(normalized = TRUE) does.

Value

A named numeric vector with one score per node, in input node order.

References

Marchiori, M., & Latora, V. (2000). Harmony in the small-world. Physica A, 285(3-4), 539-546. doi:10.1016/S03784371(00)003113.

Boldi, P., & Vigna, S. (2014). Axioms for centrality. Internet Mathematics, 10(3-4), 222-262. doi:10.1080/15427951.2013.865686.

See Also

centrality_closeness, centrality_reaching_local, centrality.

Examples

```
centrality_harmonic(regulation_net)
```

---

centrality_hcc *Hybrid Characteristic Centrality*

Description

Hybrid characteristic centrality (Liu and Zheng 2023) adds the extended degree $k^{ex}(u) = \delta k(u) + (1-\delta)\sum_{v\in\phi(u)} k(v)$ to the E-shell position index $pos(u)$, each divided by its maximum. The E-shell decomposition removes, round by round, the remaining nodes of minimum extended degree, recomputed on the residual graph, and $pos(u)$ is the round in which u leaves.

$$HCC(u) = \frac{k^{ex}(u)}{k^{ex}_{max}} + \frac{pos(u)}{pos_{max}}$$

Usage

```
centrality_hcc(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses hcc_delta (weight δ of the node's own degree in the extended degree, default 0.5).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Scores lie in $[0, 2]$. Equation (4) uses the extended degrees of the original graph, as the source's worked example requires. The normalizers are global, so adding a disconnected component can change every score. An isolate leaves in the first round, and every node of an edgeless graph scores one. A hcc_delta outside $[0, 1]$ raises a cograph_bad_parameter error. Step 3 of the source's E-shell procedure prints $\arg\max$ where its text and Table 2 require $\arg\min$, and the minimum is implemented. The Centrality Zoo describes the E-shell decomposition as a k-shell variant, which gives different positions on the source's Figure 1.

Value

A named numeric vector with one score per node, in input node order.

References

Liu, J. and Zheng, J. (2023). Identifying important nodes in complex networks based on extended degree and E-shell hierarchy decomposition. Scientific Reports, 13, 3197. doi:10.1038/s41598023-303085.

See Also

centrality_ehcc, centrality_dkgm, centrality.

Examples

```
centrality_hcc(regulation_net)
```

---

centrality_heatmap *Heatmap, Flow Coefficient, Local Entropy, Weighted h-index and Redundancy*

Description

Five local measures. Heatmap centrality (Duron 2020) is the farness of a node minus the mean farness of its neighbors, $C(v) = f(v) - \frac{1}{k_v}\sum_{u \in N(v)} f(u)$, with f the sum of hop distances to the reachable nodes. The flow coefficient (Honey et al. 2007) is the fraction of ordered pairs of distinct neighbors joined by a two-step path through the node and by no direct link, as in the Brain Connectivity Toolbox. Local entropy (Nie et al. 2016) is $-\sum_{j \in N(i)} k_j \ln k_j$. The weighted h-index (Gao et al. 2019) is the h-index of the multiset in which each neighbor j contributes the value $k_i k_j$ repeated k_j times. Redundancy (Burt 1992; Borgatti 1997) is the mean degree of the neighbors of a node within its ego network.

Usage

```
centrality_heatmap(x, mode = "all", ...)

centrality_flow_coefficient(x, ...)

centrality_local_entropy(x, mode = "all", ...)

centrality_weighted_h_index(x, mode = "all", ...)

centrality_redundancy(x, ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Edge weights are ignored by all five. Heatmap, local entropy and the weighted h-index follow mode. Redundancy ignores direction, and the flow coefficient uses the direction of the links. On an undirected network the flow coefficient of a node with at least two neighbors equals one minus its clustering coefficient, and redundancy equals degree minus effective size. Lower values mark more central nodes for heatmap and local entropy. Isolated nodes return NaN for heatmap and 0 for local entropy, and nodes with fewer than two neighbors score 0 on the flow coefficient. The formula for local entropy follows the Centrality Zoo and the survey of Omar and Plapper (2021), which agree.

Value

A named numeric vector with one score per node, in input node order. The weighted h-index is an integer vector.

References

Duron, C. (2020). Heatmap centrality: A new measure to identify super- spreader nodes in scale-free networks. PLOS ONE, 15(7), e0235690.

Honey, C. J., Kotter, R., Breakspear, M., & Sporns, O. (2007). Network structure of cerebral cortex shapes functional connectivity on multiple time scales. PNAS, 104(24), 10240-10245.

Nie, T., Guo, Z., Zhao, K., & Lu, Z.-M. (2016). Using mapping entropy to identify node centrality in complex networks. Physica A, 453, 290-297.

Gao, L., Yu, S., Li, M., Shen, Z., & Gao, Z. (2019). Weighted h-index for identifying influential spreaders. Symmetry, 11(10), 1263.

Borgatti, S. P. (1997). Structural holes: Unpacking Burt's redundancy measures. Connections, 20(1), 35-38.

See Also

centrality_effective_size, centrality_transitivity, centrality.

Examples

```
centrality_heatmap(regulation_net)
centrality_flow_coefficient(regulation_net)
centrality_local_entropy(regulation_net)
centrality_weighted_h_index(regulation_net)
centrality_redundancy(regulation_net)
```

---

centrality_hubbell *Hubbell Centrality*

Description

Hubbell (1965) centrality solves an input-output system in which the score of a node is one plus the attenuated scores of the nodes it sends ties to:

$$c = (I - wW)^{-1}\mathbf{1},$$

where W is the weighted adjacency matrix and w is `hubbell_weight`.

Usage

```
centrality_hubbell(x, hubbell_weight = 0.5, ...)
```

Arguments

`x`	Network input accepted by centrality.
`hubbell_weight`	Attenuation factor w, a positive number (default 0.5).
`...`	Further arguments to centrality.

Details

The system is solvable when the spectral radius of wW is below one. Otherwise every score is `NA` with a `cograph_undefined_measure` warning. A `hubbell_weight` of zero or below raises a `cograph_bad_parameter` error. `weighted = FALSE` gives every edge weight one. The rows of W are outgoing ties, so on a directed network the score sums attenuated walks that leave the node. `centiserve::hubbell()` with `weights = NULL` sets every weight to 1, so it reproduces these values only when the weights are passed explicitly.

Value

A named numeric vector with one score per node, in input node order.

References

Hubbell, C. H. (1965). An input-output approach to clique identification. *Sociometry*, 28(4), 377-399.

See Also

`centrality_katz`, `centrality_power`, `centrality`.

Examples

```
centrality_hubbell(regulation_net)
```

`centrality_hybrid_global_structure`
Hybrid Global Structure Model Centrality

Description

The hybrid global structure model (H-GSM; Mukhtar et al. 2023) combines a self-influence $s_i = \exp(k_s(i)\, k_i / N)$ with a distance exponent a computed from the mean self-influence:

$$HGSM(i) = s_i \sum_{j \neq i} \frac{s_j}{d_{ij}^{a}}, \qquad a = \left\lceil \log_2 \frac{1}{N} \sum_{l} s_l \right\rceil .$$

Here $k_s(i)$ is the core number, k_i the degree and N the number of nodes, isolated nodes included.

Usage

```
centrality_hybrid_global_structure(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and `mode` has no effect. Only reachable nodes contribute to the sum. Other components affect the scores through N and the mean self-influence, to which each isolated node adds 1. An isolated node scores 0. Raw scores beyond double precision raise an error. With `normalized = TRUE` the scores are computed on the log scale and remain available in that case.

Value

A named numeric vector with one score per node, in input node order.

References

Mukhtar, M. F., et al. (2023). Integrating local and global information to identify influential nodes in complex networks. Scientific Reports, 13, 11411. doi:10.1038/s41598023375707.

See Also

`centrality_global_structure`, `centrality_improved_global_structure`, `centrality`.

Examples

```
centrality_hybrid_global_structure(regulation_net)
```

`centrality_iec` *Immediate Effects Centrality*

Description

Immediate effects centrality (Friedkin 1991) is the reciprocal of the mean length of the influence sequences that end at a node. The influence matrix W is the adjacency matrix with a unit diagonal, divided by its row sums. With c its stationary vector, $Z = (I - W + \mathbf{1}c')^{-1}$ and the mean first passage times $M = (I - Z + EZ_{dg})\,\mathrm{diag}(1/c)$:

$$IEC_j = \frac{n-1}{\sum_{i \neq j} m_{ij}}$$

Usage

```
centrality_iec(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Direction is used, and weights, loops and parallel edges are ignored, so the unit diagonal is calibrated against unit edges. `mode` has no effect. The measure needs an irreducible chain, that is a connected undirected or a strongly connected directed network. Otherwise, and on a single node, every score is `NA` with a `cograph_undefined_measure` warning. The measure differs from `centrality_markov`, which omits the unit diagonal and divides by n, and the two can rank nodes differently. The measure is costly and is computed only when requested by name or through `include`.

Value

A named numeric vector with one score per node, in input node order, or `NA` at every node when the influence chain is reducible or the network has one node.

References

Friedkin, N. E. (1991). Theoretical foundations for centrality measures. American Journal of Sociology, 96(6), 1478-1504. doi:10.1086/229694.

See Also

`centrality_markov`, `centrality_random_walk`, `centrality`.

Examples

```
centrality_iec(regulation_net)
```

`centrality_iira` *Improved Iterative Resource Allocation*

Description

Improved iterative resource allocation (Zhong, Liu and Shang 2015) is `centrality_ira` with the receiver's share scaled by its spreading capacity $1-(1-\beta)^{k_i}$, where k_i is the degree and β the spreading rate. The recursion $I(t+1) = A\,I(t)$ runs a fixed number of steps from $I(0) = (1,\ldots,1)$.

$$a_{ij} = \left[1-(1-\beta)^{k_i}\right] \frac{\theta_i}{\sum_{u\in\Gamma(j)}\theta_u}$$

Usage

```
centrality_iira(
  x,
  ira_mass = "coreness",
  iira_beta = 0.2,
  iira_steps = 50,
  ...
)
```

Arguments

x	Network input accepted by centrality.
ira_mass	Node centrality θ: "coreness" (default) or "degree".
iira_beta	Spreading rate β, a number in $(0, 1]$ (default 0.2).
iira_steps	Number of iterations, a nonnegative whole number (default 50).
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Every column of A sums to less than one, so the scores decay geometrically and only their order carries meaning. normalized = TRUE divides them by their maximum. Each connected component decays at its own rate, so raw scores are comparable only within a component, and a large iira_steps underflows to zero. An isolate scores zero, and iira_steps = 0 returns a vector of ones. The Centrality Zoo (section 2.185) states a different matrix, which is stochastic in neither direction and does not reproduce the source's worked example. The implementation follows the source.

Value

A named numeric vector with one score per node, in input node order.

References

Zhong, L.-F., Liu, J.-G. and Shang, M.-S. (2015). Iterative resource allocation based on propagation feature of node for identifying the influential nodes. Physics Letters A, 379(38), 2272-2276. doi:10.1016/j.physleta.2015.05.021.

See Also

centrality_ira, centrality.

Examples

```
centrality_iira(regulation_net, normalized = TRUE)
```

---

centrality_improved_closeness

Improved Closeness Centrality

Description

Improved closeness (Luan et al. 2021) divides each hop distance by a power of the number of shortest paths, so that a partner reached along many shortest paths counts as closer:

$$ICC(i) = \frac{n-1}{\sum_{j \neq i} d_{ij} / \sigma_{ij}^{\alpha}}.$$

Here d_{ij} is the hop distance and σ_{ij} the number of shortest paths between i and j.

Usage

```
centrality_improved_closeness(x, icc_alpha = 0.2, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`icc_alpha`	Exponent α of the number of shortest paths, between 0 and 1. Default 0.2, one of the values studied by Luan et al. (2021).
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and `mode` has no effect. With $\alpha = 0$ the score on a connected network is the normalized closeness, and on a tree it does not depend on α. Scores can exceed 1. On a disconnected network every node scores 0, because each node has an unreachable partner at infinite distance. An isolated node also scores 0. A value of `icc_alpha` outside 0 to 1 raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Luan, Y., Bao, Z., & Zhang, H. (2021). Identifying Influential Spreaders in Complex Networks by Considering the Impact of the Number of Shortest Paths. Journal of Systems Science and Complexity, 34, 2168-2181. doi:10.1007/s11424021011117.

See Also

`centrality_closeness`, `centrality_harmonic`, `centrality`.

Examples

```
centrality_improved_closeness(regulation_net)
```

`centrality_improved_global_structure` *Improved Global Structure Model Centrality*

Description

The improved global structure model (IGSM; Zhu and Wang 2022) replaces the core numbers of `centrality_global_structure` with degrees and raises each distance to an exponent set by the mean degree $\bar{k}$:

$$IGSM(i) = \exp\left(\frac{k_i}{N}\right) \sum_{j \neq i} \frac{k_j}{d_{ij}^{a}}, \qquad a = \lceil \log_2 \bar{k} \rceil .$$

The formula follows equation 5 of Mukhtar et al. (2023).

Usage

```
centrality_improved_global_structure(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored, and `mode` has no effect. Only reachable nodes contribute to the sum, and N and the mean degree include every node of the network. When the mean degree is at most 1 the exponent is zero or negative, and with a negative exponent distant nodes contribute more than near ones. An isolated node scores 0, and so does every node of a network without edges.

Value

A named numeric vector with one score per node, in input node order.

References

Zhu, J.-C., & Wang, L.-W. (2022). An extended improved global structure model for influential node identification in complex networks. Chinese Physics B, 31, 068904. doi:10.1088/16741056/ac380d.

Mukhtar, M. F., et al. (2023). Integrating local and global information to identify influential nodes in complex networks. Scientific Reports, 13, 11411. doi:10.1038/s41598023375707.

See Also

`centrality_global_structure`, `centrality_hybrid_global_structure`, `centrality`.

Examples

```
centrality_improved_global_structure(regulation_net)
```

`centrality_information` *Information Centrality*

Description

Information centrality (Stephenson and Zelen 1989) measures the information carried by all paths between a node and the others, each path weighted by its length. With $C = B^{-1}$, where B has diagonal $1 + s_i$ (s_i the strength) and off-diagonal entries $1 - w_{ij}$,

$$I_i = \frac{1}{C_{ii} + (T - 2R_i)/n},$$

where T is the trace of C and R_i the sum of row i.

Usage

```
centrality_information(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`).

Details

The network is symmetrized with $(w_{ij} + w_{ji})/2$, so direction is ignored. Edge weights enter as tie strengths, and `weighted = FALSE` uses the binary matrix. Isolated nodes score 0 and are left out of n. When the other nodes do not form one connected component, or B is singular, every score is `NA` with a `cograph_undefined_measure` warning. On unweighted undirected networks the values equal `sna::infocent()`.

Value

A named numeric vector with one score per node, in input node order.

References

Stephenson, K., & Zelen, M. (1989). Rethinking centrality: Methods and examples. *Social Networks*, 11(1), 1-37.

See Also

`centrality_current_flow_closeness`, `centrality_closeness`, `centrality`.

Examples

```
centrality_information(regulation_net)
```

`centrality_integration` *Integration Centrality*

Description

Integration centrality (Valente and Foreman 1998) scores each distance against the diameter D, the largest finite hop distance, and sums:

$$I(i) = \sum_j \left(1 - \frac{d_{ij} - 1}{D}\right),$$

where an unreachable node contributes 0.

Usage

```
centrality_integration(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`.

Details

Distances are hop counts, so edge weights are ignored. The sum includes the node itself, which contributes $1+1/D$, and is not divided by $n-1$; the values equal `tidygraph::centrality_integration()`. `mode = "all"` treats edges as undirected, `"out"` uses distances from the node and `"in"` distances to it. On a network without edges every node scores n.

Value

A named numeric vector with one score per node, in input node order.

References

Valente, T. W., & Foreman, R. K. (1998). Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. Social Networks, 20(1), 89-105. doi:10.1016/S03788733(97)000075.

See Also

`centrality_radiality`, `centrality_closeness`, `centrality`.

Examples

```
centrality_integration(regulation_net)
```

`centrality_ira` *Iterative Resource Allocation*

Description

Iterative resource allocation (Ren et al. 2014) starts every node with one unit of resource and repeatedly passes it to the neighbors in proportion to the receiver's centrality θ. The steady state ranks the spreaders.

$$I(t+1) = A\, I(t), \qquad a_{ij} = \frac{\theta_i^{\alpha}}{\sum_{u \in \Gamma(j)} \theta_u^{\alpha}}, \qquad I(0) = (1, \ldots, 1)$$

Usage

```
centrality_ira(
  x,
  ira_mass = "coreness",
  ira_alpha = 1,
  ira_tol = 1e-06,
  ira_max_iter = 1000,
  ...
)
```

Arguments

`x`	Network input accepted by `centrality`.
`ira_mass`	Node centrality θ: `"coreness"` (default) or `"degree"`.
`ira_alpha`	Exponent α on the mass, a finite number (default 1).
`ira_tol`	Positive stopping tolerance on the largest absolute change between iterates (default `1e-6`).
`ira_max_iter`	Maximum number of iterations, a whole number of at least one (default 1000).
`...`	Further arguments to `centrality`, such as `normalized` (divide by the maximum, default `FALSE`).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. The total resource is conserved, so on a graph without isolates the scores sum to the number of nodes. An isolate receives nothing and scores zero. The iteration stops when the largest absolute change falls below `ira_tol`. On a bipartite component whose two vertex classes differ in size, such as a star, the iteration has a period-two cycle. It then reaches `ira_max_iter`, raises a `cograph_no_converge` warning and returns the last iterate. The Centrality Zoo states the measure as the left eigenvector of the transposed matrix, which agrees up to scale where the limit exists. The implementation follows the source's iteration.

Value

A named numeric vector with one score per node, in input node order.

References

Ren, Z.-M., Zeng, A., Chen, D.-B., Liao, H. and Liu, J.-G. (2014). Iterative resource allocation for ranking spreaders in complex networks. EPL (Europhysics Letters), 106(4), 48005. doi:10.1209/02955075/106/48005.

See Also

centrality_iira, centrality.

Examples

```
centrality_ira(regulation_net)
```

---

centrality_katz *Katz Centrality*

Description

Katz (1953) status sums the walks of every length that end at a node, each step attenuated by α:

$$c = (I - \alpha A^T)^{-1}\mathbf{1},$$

where A is the weighted adjacency matrix and α is `katz_alpha`.

Usage

```
centrality_katz(x, katz_alpha = 0.1, ...)
```

Arguments

`x`	Network input accepted by centrality.
`katz_alpha`	Attenuation factor α (default 0.1).
`...`	Further arguments to centrality.

Details

The series converges for $\alpha < 1/\rho(A)$, where $\rho(A)$ is the spectral radius. A divergent series is detected from scores below one and raises a `cograph_katz_diverged` warning that names the bound; the returned values are then not Katz scores. `weighted = FALSE` gives every edge weight one. On a directed network the score counts walks that arrive at the node. A network of one node without a self-loop scores 1. The values equal `igraph::alpha_centrality(exo = 1)` with the same `alpha`.

Value

A named numeric vector with one score per node, in input node order.

References

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39-43.

See Also

centrality_alpha, centrality_eigenvector, centrality_hubbell, centrality.

Examples

```
centrality_katz(regulation_net)
```

---

centrality_ked *KED Centrality*

Description

The KED method (Chen et al. 2014) combines a node's degree k_i with the number and the diversity of the local paths that leave it. The local path number $K_i = \sum_{j \in N(i)} k_j$ is the neighbors' degree sum, and the path diversity H_i is the entropy of the shares $p_j = k_j / K_i$ divided by $\log k_i$.

$$KED_i = k_i \, (1 + H_i) \, \exp(K_i / N), \qquad H_i = \frac{-\sum_{j \in N(i)} p_j \log p_j}{\log k_i}$$

Usage

```
centrality_ked(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. The source's directed variant is not implemented. H_i lies in $[0, 1]$, and it is set to zero for a node with one neighbor, where it is $0/0$. Isolates score zero. N is the number of nodes in the whole network, so adding a disconnected component changes every score and can change the ranking. The source's stated range $1 \le \exp(K_i/N) \le e$ holds only when $K_i \le N$, and an overflow of the exponential raises an error. The Centrality Zoo (section 2.215) adds exponents, drops the $1+$ term and divides by the largest K_l. That form does not reproduce the scores printed in the source.

Value

A named numeric vector with one score per node, in input node order.

References

Chen, D.-B., Xiao, R., Zeng, A. and Zhang, Y.-C. (2014). Path diversity improves the identification of influential spreaders. Europhysics Letters, 104(6), 68006. doi:10.1209/02955075/104/68006.

See Also

centrality_lnc, centrality_entropy, centrality.

Examples

```
centrality_ked(regulation_net)
```

---

centrality_kreach *Geodesic K-Reach Centrality*

Description

Geodesic k-reach centrality (Borgatti and Everett 2006) counts the nodes that lie within shortest-path distance k of a node.

Usage

```
centrality_kreach(x, mode = "all", k = 3, ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
k	Largest distance counted, a positive number (default 3).
...	Further arguments to centrality. The measure uses invert_weights (default NULL, which inverts for tna input only) and alpha (inversion exponent, default 1).

Details

Edge weights are read as distances, so on a weighted network k is compared with the summed weights along a path; on regulation_net, whose weights lie below one, every node reaches all others within $k = 1$. invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. weighted = FALSE uses hop counts. mode = "all" treats edges as undirected, "out" counts nodes reached from the node and "in" nodes that reach it. A k of zero or below raises a cograph_bad_parameter error.

Value

A named integer vector with one count per node, in input node order.

References

Borgatti, S. P., & Everett, M. G. (2006). A graph-theoretic perspective on centrality. Social Networks, 28(4), 466-484. doi:10.1016/j.socnet.2005.11.005.

See Also

centrality_geodesic_kpath, centrality_harmonic, centrality.

Examples

```
centrality_kreach(regulation_net, k = 0.2)
```

---

centrality_lac *Local Average Connectivity*

Description

Local average connectivity (Li et al. 2011) is the mean degree of the neighbors of a node within the subgraph C_v induced by those neighbors:

$$LAC(v) = \frac{1}{k_v} \sum_{u \in N(v)} k_u^{C_v},$$

where $k_u^{C_v}$ is the degree of u inside C_v. High values mark nodes whose neighbors are tied to each other.

Usage

```
centrality_lac(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality.

Details

Edge weights are ignored. mode selects the neighbors and the degree counted inside C_v. With mode = "all" on a directed network in-ties and out-ties are both counted, so a reciprocated tie counts twice. An isolated node scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Li, M., Wang, J., Chen, X., Wang, H., & Pan, Y. (2011). A local average connectivity-based method for identifying essential proteins from the network level. *Computational Biology and Chemistry*, 35(3), 143-150.

See Also

centrality_dmnc, centrality_mnc, centrality.

Examples

```
centrality_lac(regulation_net)
```

---

centrality_laplacian *Laplacian Centrality*

Description

Laplacian centrality (Qi et al. 2012) is the drop in the Laplacian energy of a network when a node is removed. Without edge weights the drop is

$$L(v) = k_v^2 + k_v + 2 \sum_{u \in N(v)} k_u.$$

Usage

```
centrality_laplacian(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses normalized (default FALSE).

Details

Edge weights are ignored, so the score is the unweighted case of the weighted measure of Qi et al. (2012). On a directed network k is the total degree, in plus out, and the neighbor sum runs over out-neighbors. A self-loop adds 2 to the degree. On undirected networks the values equal centiserve::laplacian(). normalized = TRUE divides the scores by their maximum.

Value

A named numeric vector with one score per node, in input node order.

References

Qi, X., Fuller, E., Wu, Q., Wu, Y., & Zhang, C.-Q. (2012). Laplacian centrality: A new centrality measure for weighted networks. Information Sciences, 194, 240-253. doi:10.1016/j.ins.2011.12.027.

See Also

centrality_degree, centrality_semilocal, centrality.

Examples

```
centrality_laplacian(regulation_net)
```

---

centrality_leaderrank *LeaderRank Centrality*

Description

LeaderRank (Lu et al. 2011) adds a ground node joined in both directions to every node and runs a random walk without damping on the extended network. The final score of the ground node is shared equally among the other nodes.

Usage

```
centrality_leaderrank(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

The measure needs a directed network. On undirected input every score is NA with a cograph_undefined_measure warning. Edge weights are ignored; centrality_weighted_leaderrank uses them. The walk starts with one unit at every node and none at the ground node, so the scores sum to n. The values equal centiserve::leaderrank().

Value

A named numeric vector with one score per node, in input node order.

References

Lu, L., Zhang, Y.-C., Yeung, C. H., & Zhou, T. (2011). Leaders in social networks, the Delicious case. PLoS ONE, 6(6), e21202. doi:10.1371/journal.pone.0021202.

See Also

centrality_weighted_leaderrank, centrality_pagerank, centrality.

Examples

```
centrality_leaderrank(regulation_net)
```

`centrality_length_scaled_betweenness`
Length-Scaled, Delta and Ego Betweenness, and Delta Closeness

---

Description

Length-scaled betweenness (Borgatti and Everett 2006; Brandes 2008) weights each pair s, t in the betweenness sum by $1/d(s,t)$. Delta betweenness (Agneessens et al. 2017) uses the pair weight $(h(s,t)-1)^{-\delta}$, where $h(s,t)$ is the number of edges on a shortest path, so $\delta = 0$ gives ordinary betweenness. Ego betweenness (Everett and Borgatti 2005) is the betweenness of a node inside its own ego network. Delta closeness (Agneessens et al. 2017, eq. 2) is

$$C_\delta(i) = \frac{1}{n-1} \sum_{j \neq i} d_{ij}^{-\delta}.$$

Usage

```
centrality_length_scaled_betweenness(x, ...)

centrality_delta_betweenness(x, betweenness_delta = 1, ...)

centrality_ego_betweenness(x, ...)

centrality_delta_closeness(x, mode = "all", closeness_delta = 1, ...)
```

Arguments

`x`	Network input accepted by centrality.
`...`	Further arguments to centrality. The weighted measures use `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which inverts for tna input only) and `alpha` (inversion exponent, default 1). Delta closeness also uses `cutoff` (default -1, no limit).
`betweenness_delta`	Decay exponent δ of delta betweenness. Default 1.
`mode`	Direction for delta closeness: `"all"` (default), `"out"` or `"in"`.
`closeness_delta`	Distance exponent δ of delta closeness. Default 1.

Details

On a directed network the three betweenness measures count directed paths. Length-scaled and delta betweenness and delta closeness read edge weights as distances, and `invert_weights = TRUE` converts weights to distances $1/w^{\alpha}$. In delta betweenness the weighted distances decide which paths are shortest, and the pair weight $(h-1)^{-\delta}$ uses the number of edges h on a shortest path (the fewest when several tie), so $h-1$ is the number of intermediaries. On a binary network $h = d(s,t)$. A pair joined by a one-edge shortest path contributes nothing. `weighted = FALSE` uses hop counts. Ego betweenness ignores weights, and a node with fewer than two neighbors scores 0. Delta closeness follows `mode` and `cutoff`. On hop distances $\delta = 1$ gives harmonic closeness divided by $n-1$ and $\delta = 0$ gives the share of nodes reached. Bounded-distance betweenness, listed in the Centrality Zoo as k-betweenness, is `centrality_betweenness(x, cutoff = k)`.

Value

A named numeric vector with one score per node, in input node order.

References

Borgatti, S. P., & Everett, M. G. (2006). A graph-theoretic perspective on centrality. Social Networks, 28(4), 466-484. doi:10.1016/j.socnet.2005.11.005.

Agneessens, F., Borgatti, S. P., & Everett, M. G. (2017). Geodesic based centrality: Unifying the local and the global. Social Networks, 49, 12-26.

Brandes, U. (2008). On variants of shortest-path betweenness centrality and their generic computation. Social Networks, 30(2), 136-145.

Everett, M., & Borgatti, S. P. (2005). Ego network betweenness. Social Networks, 27(1), 31-38.

See Also

`centrality_betweenness`, `centrality_harmonic`, `centrality`.

Examples

```
centrality_length_scaled_betweenness(regulation_net)
centrality_delta_betweenness(regulation_net, weighted = FALSE)
centrality_ego_betweenness(regulation_net)
centrality_delta_closeness(regulation_net, closeness_delta = 2)
```

---

centrality_leverage *Leverage Centrality*

Description

Leverage centrality (Joyce et al. 2010) compares the degree of a node with the degrees of its neighbors:

$$l_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} \frac{k_i - k_j}{k_i + k_j}.$$

Positive values mark nodes with more ties than their typical neighbor.

Usage

```
centrality_leverage(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality.

Details

Edge weights are ignored. mode selects both the degree and the neighbor set, and with mode = "all" on a directed network the degree is in plus out. The score lies between -1 and 1. An isolated node is NaN. On undirected networks the values equal centiserve::leverage().

Value

A named numeric vector with one score per node, in input node order.

References

Joyce, K. E., Laurienti, P. J., Burdette, J. H., & Hayasaka, S. (2010). A new measure of centrality for brain networks. PLoS ONE, 5(8), e12200. doi:10.1371/journal.pone.0012200.

See Also

centrality_degree, centrality_neighborhood_connectivity, centrality.

Examples

```
centrality_leverage(regulation_net)
```

---

centrality_lhc *Lhc Index*

Description

The Lhc index (Wang et al. 2021) sums over a node's neighbors w an influence $C(w)$, the degree of every node u within distance d of w, inflated by its triangle share and divided by the squared distance. The triangle share is $TP(u) = NTS(u)/TNTS$, where $NTS(u)$ counts the triangles on u and $TNTS = \sum_u NTS(u)$.

$$Lhc(v) = \sum_{w \in N(v)} C(w), \qquad C(w) = \sum_{1 \le d(u,w) \le d} \frac{k_u(1 + TP(u))}{d(u,w)^2}$$

Usage

```
centrality_lhc(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses lhc_radius (the distance range d, default 2).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. On a triangle-free graph TP is set to zero. $TNTS$ is a global sum, so adding a component that carries a triangle changes every score. Isolates score zero. A lhc_radius that is not a whole number of at least one raises a cograph_bad_parameter error. The Centrality Zoo (section 2.221) names the denominator the number of triangles, which is one third of $TNTS$. The implementation follows the source.

Value

A named numeric vector with one score per node, in input node order.

References

Wang, X., Yang, Q., Liu, M. and Ma, X. (2021). Comprehensive influence of topological location and neighbor information on identifying influential nodes in complex networks. PLoS ONE, 16(5), e0251208. doi:10.1371/journal.pone.0251208.

See Also

centrality_hcc, centrality_ked, centrality.

Examples

```
centrality_lhc(regulation_net)
```

---

centrality_lin *Lin Centrality*

Description

Lin centrality divides the squared number of nodes a node reaches by the sum of its distances to them:

$$L(i) = \frac{r_i^2}{\sum_{j \in R_i} d_{ij}},$$

where R_i is the set of r_i nodes reachable from i. The score is defined on disconnected networks.

Usage

```
centrality_lin(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only), alpha (inversion exponent, default 1) and cutoff (largest distance counted, default -1 for no limit).

Details

Edge weights are read as distances. weighted = FALSE uses hop counts, and invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. mode = "all" treats edges as undirected, "out" uses distances from the node and "in" distances to it. A node that reaches no other node scores 0, and a single-node network gives NA. On undirected networks the values equal centiserve::lincent().

Value

A named numeric vector with one score per node, in input node order.

See Also

centrality_closeness, centrality_harmonic, centrality.

Examples

```
centrality_lin(regulation_net)
```

---

centrality_linerank *LineRank Centrality*

Description

LineRank (Kang et al. 2011) computes PageRank on the line graph, whose nodes are the edges of the network, and gives each node the sum of the stationary probabilities of its incident edges. In a directed network edge e leads to edge f when the target of e is the source of f. In an undirected network two edges are adjacent when they share an endpoint (Kosa et al. 2015).

Usage

```
centrality_linerank(
  x,
  damping = 0.85,
  linerank_aggregation = "probability",
  ...
)
```

Arguments

x	Network input accepted by centrality.
damping	Probability of continuing the walk on the line graph, in $[0, 1)$. Default 0.85.
linerank_aggregation	"probability" (default) sums the stationary edge probabilities. "weight" multiplies them by the edge weights first.
...	Further arguments to centrality. The measure uses weighted (use edge weights, default TRUE) and normalized (divide by the maximum, default FALSE).

Details

The walk on the line graph moves to an adjacent edge with probability proportional to that edge's weight and jumps to a uniformly chosen edge with probability 1 - damping. An edge without successors also jumps uniformly. With linerank_aggregation = "probability" the scores sum to two on a network with edges. With "weight" each stationary probability is multiplied by its edge weight before aggregation, following the weighted incidence matrix of Algorithm 2 in Kang et al. (2011). A loop contributes twice to its node, and an isolated node scores zero. Edge weights must be finite and nonnegative, and weighted = FALSE gives every edge weight one. A damping outside $[0, 1)$ raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Kang, U., Papadimitriou, S., Sun, J., & Tong, H. (2011). Centralities in Large Networks: Algorithms and Observations. Proceedings of the 2011 SIAM International Conference on Data Mining, 119-130. doi:10.1137/1.9781611972818.11.

Kosa, B., Balassi, M., Englert, P., & Kiss, A. (2015). Betweenness versus Linerank. Computer Science and Information Systems, 12(1), 33-48. doi:10.2298/CSIS141101092K.

See Also

centrality_pagerank, centrality_betweenness, centrality.

Examples

```
centrality_linerank(regulation_net)
```

`centrality_lnc` *Local Neighbor Contribution Centrality*

Description

Local neighbor contribution (Dai et al. 2019) multiplies a node's own contribution $d_i(1-1/d_i)^{d_i-1}$ by its neighbor contribution, the squared degree times the neighbors' degree sum divided by $n-1$.

$$LNC_i = d_i^3 \left(1 - \frac{1}{d_i}\right)^{d_i-1} \frac{\sum_{j \in N(i)} d_j}{n-1}$$

Usage

```
centrality_lnc(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored, and it takes no parameters. The count n is the number of nodes in the whole network, so adding a disconnected component rescales every score and leaves the ranking unchanged. Isolates and the node of a single-node graph score zero. The factorization above is the one that reproduces the source's printed intermediates and Table 1. The Centrality Zoo (section 2.238) replaces the degree by the two-hop neighborhood size and does not reproduce the source's values.

Value

A named numeric vector with one score per node, in input node order.

References

Dai, J., Wang, B., Sheng, J., Sun, Z., Khawaja, F. R., Ullah, A., Dejene, D. A. and Duan, G. (2019). Identifying influential nodes in complex networks based on local neighbor contribution. IEEE Access, 7, 131719-131731. doi:10.1109/ACCESS.2019.2939804.

See Also

`centrality_semilocal`, `centrality_ked`, `centrality`.

Examples

```
centrality_lnc(regulation_net)
```

centrality_load *Load Centrality*

Description

Load centrality (Goh et al. 2001) sends one unit of load from every node to every other node along shortest paths. At each branching the load is split equally among the shortest-path predecessors, and the score of a node is the total load that passes through it.

Usage

```
centrality_load(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only) and alpha (inversion exponent, default 1).

Details

Edge weights are read as distances. weighted = FALSE uses hop counts, and invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. Paths follow edge direction on a directed network. The values equal sna::loadcent(), which credits the endpoints of each path as well as the intermediate nodes, so the scores are larger than betweenness.

Value

A named numeric vector with one score per node, in input node order.

References

Goh, K.-I., Kahng, B., & Kim, D. (2001). Universal behavior of load distribution in scale-free networks. Physical Review Letters, 87(27), 278701. doi:10.1103/PhysRevLett.87.278701.

See Also

centrality_betweenness, centrality_stress, centrality.

Examples

```
centrality_load(regulation_net)
```

`centrality_lobby` *Lobby Index*

Description

The lobby index (Korn et al. 2009) is the h-index of the degrees in the closed neighborhood of a node. It is the largest k such that the node and its neighbors include at least k nodes of degree k or more.

Usage

```
centrality_lobby(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`.

Details

Edge weights are ignored. `mode` selects both the degree and the neighbors, and with `mode = "all"` on a directed network the degree is in plus out. An isolated node scores 0. On undirected networks the values equal `centiserve::lobby()`.

Value

A named integer vector with one index per node, in input node order.

References

Korn, A., Schubert, A., & Telcs, A. (2009). Lobby index in networks. Physica A, 388(11), 2221-2226. doi:10.1016/j.physa.2009.02.013.

See Also

`centrality_degree`, `centrality_coreness`, `centrality`.

Examples

```
centrality_lobby(regulation_net)
```

`centrality_local_bridging` *Local Bridging Centrality*

Description

Local bridging centrality multiplies the bridging coefficient of a node by its inverse degree:

$$LB(v) = \frac{1}{k_v} \cdot \frac{1/k_v}{\sum_{u \in N(v)} 1/k_u}.$$

A node of low degree whose neighbors have high degree scores high.

Usage

```
centrality_local_bridging(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`.

Details

Edge weights are ignored. On a directed network k is the total degree, in plus out, and a reciprocated tie counts twice. An isolated node scores 0. The local bridging centrality of Nanda and Kotz, the product of ego betweenness and the bridging coefficient, is `centrality_localized_bridging`.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_bridging`, `centrality_localized_bridging`, `centrality`.

Examples

```
centrality_local_bridging(regulation_net)
```

`centrality_local_dimension` *Local Dimension*

Description

The local dimension (Silva and Costa 2013; Pu et al. 2014) is the growth exponent of the ball around a node. With $B_i(r)$ the number of nodes within r hops of i, the node itself included, it is the least-squares slope of $\ln B_i(r)$ on $\ln r$ over $r = 1, \ldots, d_{\max}(i)$:

$$D_i = \frac{d \ln B_i(r)}{d \ln r}.$$

Usage

```
centrality_local_dimension(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Distances are hop counts, so edge weights are ignored. On a directed network `mode` sets the direction of the paths. A node that reaches most of the network in a few hops has a small exponent, so lower values mark more influential nodes. A node with a single radius returns the discretized derivative $r\, n_i(r)/B_i(r)$ at $r = 1$, where $n_i(r)$ counts the nodes at distance exactly r. A node that reaches no other node returns `NaN`.

Value

A named numeric vector with one score per node, in input node order.

References

Silva, F. N., & Costa, L. da F. (2013). Local dimension of complex networks. arXiv:1209.2476.

Pu, J., Chen, X., Wei, D., Liu, Q., & Deng, Y. (2014). Identifying influential nodes based on local dimension. EPL, 107(1), 10010.

Wen, T., & Jiang, W. (2019). Identifying influential nodes based on fuzzy local dimension in complex networks. Chaos, Solitons & Fractals, 119, 332-342.

See Also

`centrality_local_information_dimension`, `centrality_local_dimension_fixed`, `centrality`.

Examples

```
centrality_local_dimension(regulation_net)
```

`centrality_local_dimension_fixed`
Fixed-Radius, Fuzzy and Volume Local Dimensions

Description

Three members of the local-dimension family, with the center node counted in its own ball. The fixed-radius local dimension (Silva and Costa 2013) is $D_i(r) = r\, n_i(r)/B_i(r)$ at one radius r, where $n_i(r)$ counts the nodes at distance r and $B_i(r)$ those within it. The fuzzy local dimension (Wen and Jiang 2019) is the slope of $\log N_i(r)$ on $\log r$ for the fuzzy ball

$$N_i(r) = \frac{\sum_{d_{ij} \le r} e^{-d_{ij}^2/r^2}}{|\{j : d_{ij} \le r\}|}.$$

The local volume dimension (Li and Deng 2021) is the slope of $\ln V_i(l)$ on $\ln l$ for the volume $V_i(l) = \sum_{d_{ij} \le l} k_j$.

Usage

```
centrality_local_dimension_fixed(x, mode = "all", ld_radius = 2, ...)

centrality_fuzzy_local_dimension(x, mode = "all", ...)

centrality_local_volume_dimension(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`ld_radius`	Radius r for `local_dimension_fixed`, in hops (default 2).
`...`	Further arguments to `centrality`, such as `normalized`.

Details

Distances are hop counts, so edge weights are ignored. On a directed network `mode` sets the direction of the paths. The fixed-radius form is a structural descriptor, and nodes whose eccentricity is below the radius score 0. Larger fuzzy dimensions and smaller volume dimensions mark more influential nodes. The two regression forms return `NaN` for a node with fewer than two radii. The volume dimension follows a later preprint of the authors and the Centrality Zoo entry. A `ld_radius` that is not a single number of at least 1 raises an error of class `cograph_bad_parameter`.

Value

A named numeric vector with one score per node, in input node order.

References

Silva, F. N., & Costa, L. da F. (2013). Local dimension of complex networks. arXiv:1209.2476.

Wen, T., & Jiang, W. (2019). Identifying influential nodes based on fuzzy local dimension in complex networks. Chaos, Solitons & Fractals, 119, 332-342.

Li, H., & Deng, Y. (2021). Local volume dimension: A novel approach for important nodes identification in complex networks. International Journal of Modern Physics B, 35(5), 2150069.

See Also

centrality_local_dimension, centrality_local_information_dimension, centrality.

Examples

```
centrality_local_dimension_fixed(regulation_net)
centrality_fuzzy_local_dimension(regulation_net)
centrality_local_volume_dimension(regulation_net)
```

---

centrality_local_efficiency

Local Efficiency, s-Core, Fragmentation, k-Path and EPC

Description

Local efficiency (Latora and Marchiori 2001) is the mean of $1/d_{jl}$ over ordered pairs of a node's neighbors, with distances measured inside the subgraph induced on those neighbors. The s-core index (Eidsaa and Almaas 2013) is the largest strength threshold s whose s-core contains the node. Fragmentation (Borgatti 2006) is the distance-weighted fragmentation of the network after the node is deleted,

$$F_{-v} = 1 - \frac{\sum_{i \neq j} 1/d_{ij}}{(n-1)(n-2)}.$$

The k-path count (Sade 1989) is the number of simple paths of length at most `kpath_len` that pass through or end at the node. The edge percolated component (EPC; Lin et al. 2008) is the mean size of the node's component, as a share of all nodes, when each edge is kept with probability `1 - epc_threshold`.

Usage

```
centrality_local_efficiency(x, mode = "all", ...)

centrality_s_core(x, ...)

centrality_fragmentation(x, mode = "all", ...)

centrality_kpath(x, mode = "all", kpath_len = 3, ...)

centrality_epc(x, epc_threshold = 0.5, epc_runs = 1000, epc_seed = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. Local efficiency and fragmentation use weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only) and alpha (inversion exponent, default 1). The s-core index uses weighted.
kpath_len	Maximum path length for the k-path count. Default 3. Length 1 gives the degree in the undirected skeleton.
epc_threshold	Probability that an edge is removed in one realization. Default 0.5.
epc_runs	Number of percolation realizations. Default 1000.
epc_seed	Random seed. The default NULL uses the caller's random stream, so the estimate varies between calls. A seed gives a reproducible value and leaves the caller's stream unchanged.

Details

Local efficiency, fragmentation and the k-path count follow mode, and mode = "all" treats edges as undirected. Local efficiency and fragmentation read edge weights as distances, so with weights below one local efficiency exceeds one and fragmentation is negative; invert_weights = TRUE converts weights to distances $1/w^{\alpha}$. On unweighted input both lie between 0 and 1, and a node with fewer than two neighbors has local efficiency 0. Fragmentation is NaN on a network with fewer than three nodes. The s-core index symmetrizes the network by the stronger of the two directions and equals the k-core number when all weights are one. The k-path count and EPC ignore weights, and EPC uses the undirected skeleton. EPC is a Monte Carlo estimate, and cytoHubba and centiserve::epc() report the same quantity multiplied by the number of runs.

Value

A named numeric vector with one score per node, in input node order.

References

Latora, V., & Marchiori, M. (2001). Efficient behavior of small-world networks. Physical Review Letters, 87(19), 198701.

Eidsaa, M., & Almaas, E. (2013). s-core network decomposition: A generalization of k-core analysis to weighted networks. Physical Review E, 88(6), 062819. doi:10.1103/PhysRevE.88.062819.

Borgatti, S. P. (2006). Identifying sets of key players in a social network. Computational and Mathematical Organization Theory, 12(1), 21-34.

Sade, D. S. (1989). Sociometrics of Macaca mulatta III: n-path centrality in grooming networks. Social Networks, 11(3), 273-292.

Lin, C.-Y., Chin, C.-H., Wu, H.-H., Chen, S.-H., Ho, C.-W., & Ko, M.-T. (2008). Hubba: hub objects analyzer, a framework of interactome hubs identification for network biology. Nucleic Acids Research, 36, W438-W443. doi:10.1093/nar/gkn257.

See Also

centrality_coreness, centrality_geodesic_kpath, network_local_efficiency, centrality.

Examples

```
centrality_local_efficiency(regulation_net)
centrality_s_core(regulation_net)
centrality_fragmentation(regulation_net, weighted = FALSE)
centrality_kpath(regulation_net, kpath_len = 2)
centrality_epc(regulation_net, epc_runs = 100, epc_seed = 1)
```

---

centrality_local_information_dimension

Local Information Dimensionality

Description

Local information dimensionality (Wen and Deng 2020) weights the local dimension by information. With $p_i(l) = B_i(l)/N$ the share of the network within l hops of i, the node included, and box information $I_i(l) = -p_i(l) \ln p_i(l)$, the measure is minus the least-squares slope of $I_i(l)$ on $\ln l$ for $l = 1, \ldots, \lceil d_{\max}(i)/2 \rceil$:

$$D_i^I = -\frac{dI_i(l)}{d \ln l}.$$

Usage

```
centrality_local_information_dimension(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	For directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality, such as normalized.

Details

Distances are hop counts, so edge weights are ignored. On a directed network mode sets the direction of the paths. Higher values mark more influential nodes. A node with a single box size returns the discretized derivative of the source, $l(1 + \ln p_i(l))\, n_i(l)/N$. A node that reaches no other node returns NaN.

Value

A named numeric vector with one score per node, in input node order.

References

Wen, T., & Deng, Y. (2020). Identification of influencers in complex networks by local information dimensionality. Information Sciences, 512, 549-562.

See Also

centrality_local_dimension, centrality_local_dimension_fixed, centrality.

Examples

```
centrality_local_information_dimension(regulation_net)
```

---

centrality_localized_bridging

Localized Bridging Centrality

Description

Localized bridging centrality (Nanda and Kotz 2012) is the product of a node's betweenness B_i^{ego} in its one-hop ego network and its bridging coefficient, the reciprocal of its degree divided by the sum of the reciprocal degrees of its neighbors:

$$LBC_i = B_i^{ego} \frac{1/d_i}{\sum_{j \in N(i)} 1/d_j}.$$

Usage

```
centrality_localized_bridging(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Degrees are taken from the whole network. Ego betweenness counts unordered pairs of the other ego-network nodes, excludes endpoints and is not normalized. Isolated nodes, leaves and every node of a complete graph score zero.

Value

A named numeric vector with one score per node, in input node order.

References

Nanda, S. and Kotz, D. (2012). Localized Bridging Centrality. In Handbook of Optimization in Complex Networks, pp. 197-224. doi:10.1007/9781461408574_7.

See Also

centrality_extended_local_bridging, centrality_local_bridging, centrality_ego_betweenness, centrality.

Examples

```
centrality_localized_bridging(regulation_net)
```

---

centrality_malatya *Malatya Centrality*

Description

The Malatya centrality of a node (Karci et al. 2022) is the sum of its degree divided by the degree of each neighbor, $M(i) = \sum_{j \in N(i)} d_i/d_j$. High scores mark nodes with many neighbors of low degree.

Usage

```
centrality_malatya(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Isolated nodes score 0. On a regular network the score equals the degree. On every node with at least one neighbor it is the reciprocal of centrality_bridging_coefficient.

Value

A named numeric vector with one score per node, in input node order.

References

Karci, A., Yakut, S., & Oztemiz, F. (2022). A New Approach Based on Centrality Value in Solving the Minimum Vertex Cover Problem: Malatya Centrality Algorithm. Journal of Computer Science, 7(2), 81-88. doi:10.53070/bbd.1195501.

See Also

centrality_bridging_coefficient, centrality_degree, centrality.

Examples

```
centrality_malatya(regulation_net)
```

`centrality_map_equation` *Map Equation Centrality*

Description

Map equation centrality (Blocker et al. 2022) is the reduction in codelength, in bits, when a node's codeword is removed from its module codebook and the codebook is redesigned. With p the visit rate of the node and s the use rate of its module codebook,

$$MEC_i = -(s - p) \log_2 \frac{s - p}{s}.$$

Usage

```
centrality_map_equation(
  x,
  membership = NULL,
  map_flow = "unrecorded",
  map_convention = "paper",
  ...
)
```

Arguments

`x`	Network input accepted by centrality.
`membership`	One module label per node. `NULL` (default) gives one module. A named vector is matched to the node names.
`map_flow`	`"unrecorded"` (default) for unrecorded link teleportation or `"recorded"` for recorded uniform teleportation.
`map_convention`	`"paper"` (default) includes module exits in the codebook rate. `"infomap"` uses node visits only.
`...`	Further arguments to centrality. The measure uses `weighted` (use edge weights, default `TRUE`) and `damping` (probability of following a link, default 0.85).

Details

The partition is held fixed and is given by `membership`, and `NULL` places every node in one module. With `map_convention = "paper"` the rate s includes module exits (equations 2 and 9-11), and with `"infomap"` it includes node visits only, which reproduces Infomap 2.15.1 and Table 1 of the paper. With `map_flow = "unrecorded"` (Lambiotte and Rosvall 2012) the walk teleports in proportion to out-strength and only link steps are recorded, so on an undirected network the visit rates are proportional to strength for every `damping`. With `"recorded"` the walk teleports uniformly and teleportation steps are recorded. Direction is kept, loops are removed and edge weights must be finite and nonnegative. Under unrecorded flow an isolated node scores zero. An invalid `membership`, `map_flow` or `map_convention`, or a `damping` outside $[0, 1)$, raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Blocker, C., Nieves, J. C. and Rosvall, M. (2022). Map equation centrality: community-aware centrality based on the map equation. Applied Network Science, 7, 56. doi:10.1007/s41109022-004779.

Lambiotte, R. and Rosvall, M. (2012). Ranking and clustering of nodes in networks with smart teleportation. Physical Review E, 85, 056107. doi:10.1103/PhysRevE.85.056107.

See Also

`centrality_community_based`, `centrality_participation`, `centrality`.

Examples

```
centrality_map_equation(regulation_net, membership = rep(1:2, each = 5))
```

`centrality_markov` *Markov Centrality*

Description

Markov centrality (White and Smyth 2003) is the inverse of the mean first passage time of a random walk into a node:

$$M(j) = \left(\frac{1}{n} \sum_i m_{ij} \right)^{-1},$$

where m_{ij} is the expected number of steps from i to the first visit of j and $m_{jj} = 0$.

Usage

```
centrality_markov(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

The walk moves from a node to each of its out-neighbors with equal probability, so edge weights are ignored. A disconnected network, or a directed network with a node that has no outgoing edge, gives NA for every node with a cograph_undefined_measure warning. On a directed network that is not strongly connected, a node that some other node cannot reach has an infinite mean passage time, and its score is NA with the same warning. On connected undirected and strongly connected directed networks the values equal centiserve::markovcent().

Value

A named numeric vector with one score per node, in input node order.

References

White, S., & Smyth, P. (2003). Algorithms for estimating relative importance in networks. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 266-275). doi:10.1145/956750.956782.

See Also

centrality_random_walk, centrality_pagerank, centrality.

Examples

```
centrality_markov(regulation_net)
```

---

centrality_mcc *Maximal Clique Centrality*

Description

Maximal clique centrality (MCC; Chin et al. 2014) sums $(|C| - 1)!$ over the maximal cliques C that contain the node. A clique contained in a larger clique does not count.

Usage

```
centrality_mcc(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Single-node cliques are excluded, so an isolated node scores 0. A node whose neighbors share no edge scores its degree. Maximal clique enumeration has exponential worst-case cost. A score beyond double precision, which includes any clique with more than 171 nodes, raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Chin, C. H., et al. (2014). cytoHubba: identifying hub objects and sub-networks from complex interactome. BMC Systems Biology, 8(Suppl 4), S11. doi:10.1186/175205098S4S11.

See Also

centrality_cross_clique, centrality_epc, centrality.

Examples

```
centrality_mcc(regulation_net)
```

---

centrality_mcgm *Multi-Characteristics Gravity Model Centrality*

Description

MCGM (Li and Huang 2022) is a gravity model whose node mass combines degree K, core number S and eigenvector centrality X, each divided by its maximum over the network:

$$MCGM_i = \sum_{j:0<d(i,j)\leq R} \frac{m_i m_j}{d(i,j)^2}, \qquad m_i = K_i + \alpha S_i + X_i.$$

By default α is the larger of the medians of K and X divided by the median of S (equation 17).

Usage

```
centrality_mcgm(x, mcgm_radius = 2, mcgm_alpha = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
mcgm_radius	Hop-distance cutoff R. Default 2, the setting recommended in the paper. NULL or Inf includes every reachable node.
mcgm_alpha	NULL (default) uses the median-based α. A finite nonnegative number replaces it.
...	Further arguments to centrality. The measure uses normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored, and `gravity_mass` and `gravity_radius` have no effect. On a disconnected network the eigenvector feature is the projection of the all-ones vector onto the dominant eigenspace, so a component with a smaller spectral radius has $X = 0$. When the median core number is zero the automatic α is undefined, and an error asks for `mcgm_alpha`. `mcgm_alpha = 1` gives equation 16 of the paper. Isolated nodes score zero, and an edgeless network or a radius below one gives zero scores.

Value

A named numeric vector with one score per node, in input node order.

References

Li, Z. and Huang, X. (2022). Identifying influential spreaders by gravity model considering multi-characteristics of nodes. Scientific Reports, 12, 9879. doi:10.1038/s41598022140053.

See Also

`centrality_gravity`, `centrality_mixed_gravity`, `centrality`.

Examples

```
centrality_mcgm(regulation_net)
```

`centrality_mixed_gravity`

Mixed Gravitational Centrality

Description

Mixed gravitational centrality (Wang, Li and Xia 2018), also called improved gravitational centrality, uses the core number $k_s(i)$ of the focal node and the degree $k(j)$ of each partner node as masses:

$$MGC_i = k_s(i) \sum_{j:0<d(i,j)\le r} \frac{k(j)}{d(i,j)^2}.$$

Usage

```
centrality_mixed_gravity(x, gravity_radius = 3, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`gravity_radius`	Hop-distance cutoff r. Default 3. `NULL` or `Inf` includes every reachable node, and a value below one gives zero scores.
`...`	Further arguments to `centrality`, such as `normalized` (divide by the maximum, default `FALSE`).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored, and `gravity_mass` has no effect. The implementation follows the reproduction of the method in Li and Huang (2022), equations 5-8. The Centrality Zoo writes the inner sum over immediate neighbors, which corresponds to `gravity_radius = 1`. Isolated nodes score zero, and unreachable nodes contribute nothing. `gravity_radius = "auto"` is a cograph heuristic that rounds half the mean finite distance to the nearest integer, with a minimum of one. A negative radius raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Wang, J., Li, C. and Xia, C. (2018). Improved centrality indicators to characterize the nodal spreading capability in complex networks. Applied Mathematics and Computation, 334, 388-400. doi:10.1016/j.amc.2018.04.028.

Li, Z. and Huang, X. (2022). Identifying influential spreaders by gravity model considering multi-characteristics of nodes. Scientific Reports, 12, 9879. doi:10.1038/s41598022140053.

See Also

`centrality_extended_mixed_gravity`, `centrality_gravity`, `centrality`.

Examples

```
centrality_mixed_gravity(regulation_net)
```

---

centrality_mnc *Maximum Neighborhood Component*

Description

The maximum neighborhood component (Lin et al. 2008) is the number of nodes in the largest connected component of the subgraph induced by the neighbors of a node.

Usage

```
centrality_mnc(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`.

Details

Edge weights are ignored and the neighbor subgraph is read as undirected. `mode` selects the neighbors. An isolated node scores 0. On undirected networks the values equal `centiserve::mnc()`.

Value

A named integer vector with one size per node, in input node order.

References

Lin, C.-Y., Chin, C.-H., Wu, H.-H., Chen, S.-H., Ho, C.-W., & Ko, M.-T. (2008). Hubba: hub objects analyzer, a framework of interactome hubs identification for network biology. Nucleic Acids Research, 36, W438-W443. doi:10.1093/nar/gkn257.

See Also

`centrality_dmnc`, `centrality_lac`, `centrality`.

Examples

```
centrality_mnc(regulation_net)
```

`centrality_modified_expected_force` *Modified Expected Force Centrality*

Description

The modified Expected Force (Lawyer 2015, equation 2) multiplies the Expected Force of a node by the logarithm of its scaled degree αd_i:

$$ExF_i^{\alpha} = \log(\alpha d_i)\, ExF_i.$$

Usage

```
centrality_modified_expected_force(x, exf_alpha = 2, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`exf_alpha`	Degree scaling factor α, a finite number greater than one. Default 2, as in the paper.
`...`	Further arguments to `centrality`, such as `normalized` (divide by the maximum, default `FALSE`).

Details

On a directed network d_i is the out-degree, in line with the outgoing transmission process. An isolated node scores zero. The input handling and zero conventions of `centrality_expected_force` apply. An `exf_alpha` that is not a finite number greater than one raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Lawyer, G. (2015). Understanding the influence of all nodes in a network. Scientific Reports, 5, 8665. doi:10.1038/srep08665.

See Also

`centrality_expected_force`, `centrality`.

Examples

```
centrality_modified_expected_force(regulation_net)
```

`centrality_modularity_vitality` *Modularity Vitality*

Description

Modularity vitality (Magelinski, Bartulovic and Carley 2021) is the drop in Newman modularity of a fixed partition C when a node is deleted and the remaining nodes keep their communities:

$$V_Q(i) = Q(G, C) - Q(G - i, C \setminus \{i\}).$$

Positive values mark community hubs and negative values mark bridges between communities.

Usage

```
centrality_modularity_vitality(x, membership = NULL, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`membership`	Community labels, one per node (integer, factor or character), for example from `detect_communities`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (use edge weights, default `TRUE`) and `loops` (keep self-loops, default `TRUE`).

Details

Edge weights are used. A directed network uses the Leicht-Newman directed modularity, and the values equal those obtained by deleting each node and recomputing `igraph::modularity()`. Self-loops enter the modularity, and `loops = FALSE` drops them. A node whose deletion leaves a graph with no edges returns `NaN`. Without `membership` the function raises a warning of classes `cograph_bad_membership` and `cograph_undefined_measure` and returns `NA` for every node. A `membership` that is not one non-missing label per node raises an error of class `cograph_bad_membership`.

Value

A named numeric vector with one score per node, in input node order.

References

Magelinski, T., Bartulovic, M., & Carley, K. M. (2021). Measuring node contribution to community structure with modularity vitality. IEEE Transactions on Network Science and Engineering, 8(1), 707-723.

See Also

`centrality_participation`, `centrality_within_module_z`, `centrality`.

Examples

```
centrality_modularity_vitality(regulation_net,
                               membership = rep(1:2, each = 5))
```

`centrality_ncvoterank` *NCVoteRank*

Description

NCVoteRank (Kumar and Panda 2020) is VoteRank (Zhang et al. 2016) with the voting ability of each voter weighted by its neighborhood coreness. A node collects the score

$$s_u = \sum_{v \in N(u)} va_v \left[\theta + (1-\theta)\, nc_v\right], \qquad nc_v = \frac{\sum_{w \in N(v)} ks(w)}{\max_j \sum_{w \in N(j)} ks(w)},$$

with ks the k-shell index. The top scorer is elected, its ability drops to 0, its neighbors lose $1/\langle k \rangle$ and the nodes two steps away lose $1/(2\langle k \rangle)$.

Usage

```
centrality_ncvoterank(x, ncvote_theta = 0.5, ...)
```

Arguments

x	Network input accepted by centrality.
ncvote_theta	Weight θ of the plain vote (default 0.5).
...	Further arguments to centrality, such as normalized.

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights and loops are ignored. Elections continue until every node is placed. The first elected scores 1 and the last $1/n$, so scores lie in $(0, 1]$. The definition follows the Centrality Zoo and three independent restatements of the article, and the scaling of the coreness by its maximum follows Yu et al. (2020). With $\theta = 1$ and no two-step weakening the procedure is VoteRank.

Value

A named numeric vector with one score per node, in input node order.

References

Kumar, S., & Panda, B. S. (2020). Identifying influential nodes in social networks: Neighborhood coreness based voting approach. Physica A, 553, 124215.

Zhang, J.-X., Chen, D.-B., Dong, Q., & Zhao, Z.-D. (2016). Identifying a set of influential spreaders in complex networks. Scientific Reports, 6, 27823.

See Also

centrality_voterank, centrality_wvoterank, centrality.

Examples

```
centrality_ncvoterank(regulation_net)
```

---

centrality_neighbor_distance

Neighborhood Centrality

Description

Neighborhood centrality (Liu et al. 2016) adds to a node's benchmark centrality θ the benchmark centrality of the endpoints of its non-backtracking walks of length 1 to n, discounted by a^k at step k. With the defaults (degree benchmark, two steps, $a = 0.2$) it is the neighbor distance centrality.

$$C_i = \theta_i + a \sum_{j \in \Gamma_i} \theta_j + a^2 \sum_{j \in \Gamma_i} \sum_{l \in \Gamma_j \setminus i} \theta_l + \dots$$

Usage

```
centrality_neighbor_distance(
  x,
  nd_order = 2,
  nd_decay = 0.2,
  nd_mass = "degree",
  ...
)
```

Arguments

x	Network input accepted by centrality.
nd_order	Number of steps n, a nonnegative whole number (default 2).
nd_decay	Per-step decay a, a finite number (default 0.2).
nd_mass	Benchmark centrality θ: "degree" (default) or "coreness".
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Each level excludes only the node the walk came from, so a walk may revisit a node. Isolates score θ_i, which is zero for both benchmarks, and nd_order = 0 returns the benchmark itself. The source takes a in $[0, 1]$, and the function accepts any finite value. The Centrality Zoo paraphrases the measure with sums over distance shells, which agree with the source's walk sums on trees and differ on graphs with short cycles. The implementation follows the source.

Value

A named numeric vector with one score per node, in input node order.

References

Liu, Y., Tang, M., Zhou, T. and Do, Y. (2016). Identify influential spreaders in complex networks, the role of neighborhood. Physica A: Statistical Mechanics and its Applications, 452, 289-298. doi:10.1016/j.physa.2016.02.028.

See Also

centrality_semilocal, centrality_extended_coreness, centrality.

Examples

```
centrality_neighbor_distance(regulation_net)
```

`centrality_neighborhood_connectivity` *Neighborhood Connectivity*

Description

Neighborhood connectivity (Maslov and Sneppen 2002) is the mean degree of the neighbors of a node, the average neighbor degree reported by Cytoscape:

$$C_{NC}(i) = \frac{1}{k_i} \sum_{j \in N(i)} k_j.$$

Usage

```
centrality_neighborhood_connectivity(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The measure uses `loops` (keep self-loops, default `TRUE`).

Details

Edge weights are ignored. Under `mode = "out"` the out-degrees of the out-neighbors are averaged, and under `mode = "in"` the in-degrees of the in-neighbors. Self-loops change the degrees, and `loops = FALSE` drops them. Isolated nodes score 0. High values mark nodes attached to hubs.

Value

A named numeric vector with one score per node, in input node order.

References

Maslov, S., & Sneppen, K. (2002). Specificity and stability in topology of protein networks. Science, 296(5569), 910-913.

See Also

`centrality_degree`, `centrality`.

Examples

```
centrality_neighborhood_connectivity(regulation_net)
```

`centrality_ninl` *Node and Neighbor Layer Information Centrality*

Description

NINL (Zhu and Wang 2021) starts each node with the sum of the degrees k_j in its closed neighborhood of radius r and then, for p iterations, replaces every score by the sum of the previous scores of its neighbors:

$$NINL^{(p)} = A^p \, NINL^{(0)}, \qquad NINL_i^{(0)} = \sum_{j:d(i,j)\le r} k_j.$$

Usage

```
centrality_ninl(x, ninl_order = 3, ninl_radius = NULL, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`ninl_order`	Number of iterations p, a nonnegative integer. Default 3, as in the paper.
`ninl_radius`	Radius r. `NULL` (default) uses the automatic radius of the paper, a nonnegative integer fixes the hop radius, and `Inf` includes every reachable node.
`...`	Further arguments to `centrality`. The measure uses `normalized` (divide by the maximum, default `FALSE`).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. The default radius is the ceiling of the average shortest-path length, as in the paper. On a disconnected network that average is infinite, so the default radius covers the whole component of each node. Isolated nodes score zero, and `ninl_order = 0` returns the initial degree sums. With `normalized = TRUE` the scores are rescaled at every iteration, and on a bipartite network they can alternate between iterations. An invalid `ninl_order` or `ninl_radius`, and raw scores that overflow, raise an error.

Value

A named numeric vector with one score per node, in input node order.

References

Zhu, J. and Wang, L. (2021). Identifying Influential Nodes in Complex Networks Based on Node Itself and Neighbor Layer Information. Symmetry, 13, 1570. doi:10.3390/sym13091570.

See Also

`centrality_semilocal`, `centrality_eigenvector`, `centrality`.

Examples

```
centrality_ninl(regulation_net)
```

`centrality_node_contraction` *Node Contraction Centrality*

Description

Node contraction importance (Tan, Wu and Deng 2006, as restated by Wang et al. 2011) compares the agglomeration $\partial(G) = 1/(N\bar{L})$ of a network, with $\bar{L}$ the mean shortest-path length, before and after the node is merged with all its neighbors into one node:

$$IMC(v) = 1 - \frac{\partial(G)}{\partial(G_v)}.$$

The improved form adds the same score of the edges of the node computed on the line graph, $IIMC(v) = \alpha\, IMC(v) + \beta \sum_{e \ni v} IMC_{L(G)}(e)$, with $\alpha + \beta = 1$.

Usage

```
centrality_node_contraction(x, ...)

centrality_node_contraction_improved(x, contraction_rho = 5, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.
`contraction_rho`	Ratio α/β for the improved form (default 5).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights and loops are ignored. Higher values mark more important nodes. The sources assume a connected network. On a disconnected network the mean path length is taken over the mutually reachable ordered pairs. An isolated node scores 0, and a node whose contraction leaves no pair of mutually reachable nodes returns `NaN`. The Centrality Zoo describes the contracted graph as the graph with the node removed. The sources define it by contraction, which is implemented here.

Value

A named numeric vector with one score per node, in input node order.

References

Tan, Y.-J., Wu, J., & Deng, H.-Z. (2006). Evaluation method for node importance based on node contraction in complex networks. Systems Engineering: Theory & Practice, 26(11), 79-83.

See Also

centrality_closeness_vitality, centrality.

Examples

```
centrality_node_contraction(regulation_net)
centrality_node_contraction_improved(regulation_net)
```

`centrality_pagerank` *PageRank Centrality*

Description

PageRank (Brin and Page 1998) is the stationary distribution of a random walk that follows an out-edge with probability d, choosing edges in proportion to their weights, and otherwise jumps to a node drawn from the reset distribution p:

$$PR = (1 - d)\, p + d\, P^T PR,$$

where P is the row-normalized weight matrix and d is `damping`.

Usage

```
centrality_pagerank(x, damping = 0.85, personalized = NULL, ...)
```

Arguments

`x`	Network input accepted by centrality.
`damping`	Probability d of following an edge (default 0.85).
`personalized`	Reset distribution p, a non-negative numeric vector with one entry per node, named by node or in input node order. The default `NULL` gives the uniform distribution.
`...`	Further arguments to centrality.

Details

`weighted = FALSE` gives every edge weight one. A node without out-edges passes its score to the reset distribution. The scores sum to one and equal `igraph::page_rank()`. The vector `personalized` is rescaled to sum to one. A named vector is matched to the node names, and its names must be the node names, each used once; an unnamed vector is matched by position. A negative weight raises a `cograph_negative_weights` error, an invalid `personalized` a `cograph_bad_input` error, and a `damping` outside $[0, 1]$ a `cograph_bad_parameter` error.

Value

A named numeric vector with one score per node, in input node order.

References

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1-7), 107-117. doi:10.1016/S01697552(98)00110X.

See Also

centrality_eigenvector, centrality_leaderrank, centrality.

Examples

```
centrality_pagerank(regulation_net)
```

---

centrality_pairwisedis

Pairwise Disconnectivity

Description

Pairwise disconnectivity (Potapov et al. 2008) is the share of ordered reachable pairs that become unreachable when a node is removed:

$$PD(v) = \frac{|P(G)| - |P(G - v)|}{|P(G)|},$$

where $|P(G)|$ is the number of ordered pairs (s, t), $s \neq t$, with a directed path from s to t.

Usage

```
centrality_pairwisedis(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

The measure needs a directed network. On undirected input every score is NA with a cograph_undefined_measure warning. Reachability uses hop counts, so edge weights are ignored. The score lies between 0 and 1, and a network without reachable pairs scores 0 everywhere. The values equal centiserve::pairwisedis().

Value

A named numeric vector with one score per node, in input node order.

References

Potapov, A. P., Goemann, B., & Wingender, E. (2008). The pairwise disconnectivity index as a new metric for the topological analysis of regulatory networks. *BMC Bioinformatics*, 9, 227. doi:10.1186/147121059227.

See Also

centrality_prestige_domain, robustness, centrality.

Examples

```
centrality_pairwisedis(regulation_net)
```

---

centrality_participation

Participation Coefficient

Description

The participation coefficient (Guimera and Nunes Amaral 2005) measures how evenly the ties of a node spread over communities:

$$P_i = 1 - \sum_s \left(\frac{k_{is}}{k_i} \right)^2,$$

where k_{is} counts the ties of node i to community s and k_i is its degree.

Usage

```
centrality_participation(x, membership = NULL, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
membership	Community of each node, a vector with one entry per node in input node order (default NULL).
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality.

Details

Edge weights are ignored. mode selects the ties counted, and with mode = "all" on a directed network a reciprocated tie counts twice. A node whose ties all stay in one community scores 0, as does an isolated node, and the score is below 1. Without membership every score is NA with a warning of classes cograph_bad_membership and cograph_undefined_measure, and a membership of the wrong length raises a cograph_bad_membership error. On undirected networks the values equal brainGraph::part_coeff().

Value

A named numeric vector with one score per node, in input node order.

References

Guimera, R., & Nunes Amaral, L. A. (2005). Functional cartography of complex metabolic networks. Nature, 433(7028), 895-900. doi:10.1038/nature03288.

See Also

centrality_within_module_z, centrality_gateway, centrality.

Examples

```
centrality_participation(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_percolation

Percolation Centrality

Description

Percolation centrality (Piraveenan et al. 2013) weights each shortest path through a node by the percolation state x_s of its source:

$$PC(v) = \frac{1}{n-2} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \frac{x_s}{\sum_i x_i - x_v},$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ the number through v. With equal states the score is the normalized betweenness.

Usage

```
centrality_percolation(x, states = NULL, ...)
```

Arguments

x	Network input accepted by centrality.
states	Percolation state of each node, between 0 and 1. The default NULL gives every node state 1.
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only) and alpha (inversion exponent, default 1).

Details

Edge weights are read as distances, `weighted = FALSE` uses hop counts, and `invert_weights = TRUE` uses the distance $1/w^{\alpha}$, so tna input is inverted by default. Paths follow edge direction on a directed network. `states` is matched to nodes by name when it has names and by position otherwise. Its values are clipped to $[0, 1]$ and missing values are set to 1, and a vector of the wrong length raises an error. A network with fewer than three nodes scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Piraveenan, M., Prokopenko, M., & Hossain, L. (2013). Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks. PLoS ONE, 8(1), e53095. doi:10.1371/journal.pone.0053095.

See Also

`centrality_betweenness`, `centrality_load`, `centrality`.

Examples

```
centrality_percolation(regulation_net)
```

---

centrality_power *Bonacich Power Centrality*

Description

Bonacich (1987) power centrality with exponent $\beta = 1$ sums the walks leaving a node, each further step weighted by β:

$$c = \gamma \, (I - A)^{-1} A \mathbf{1},$$

where A is the binary adjacency matrix and γ scales the scores so that their squares sum to n.

Usage

```
centrality_power(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`.

Details

The exponent is fixed at 1. Edge weights and self-loops are ignored. On a directed network `mode = "out"` sums over out-ties and equals `igraph::power_centrality(exponent = 1)`, `mode = "in"` sums over in-ties, and `mode = "all"` (default) uses the undirected skeleton, the binary matrix with a tie wherever either direction has one. On an undirected network the three modes agree. Where $I - A$ is singular, as on a network that contains an isolated edge, a `cograph_singular_system` error is raised, and a network without edges gives `NaN`. The scores can be negative, as on `regulation_net`, and `normalized = TRUE` leaves scores that are all negative unchanged.

Value

A named numeric vector with one score per node, in input node order.

References

Bonacich, P. (1987). Power and centrality: A family of measures. American Journal of Sociology, 92(5), 1170-1182. doi:10.1086/228631.

See Also

`centrality_eigenvector`, `centrality_katz`, `centrality`.

Examples

```
centrality_power(regulation_net)
```

`centrality_prestige_domain` *Domain Prestige*

Description

Domain prestige (Wasserman and Faust 1994) counts the other nodes that reach a node through a directed path:

$$D(v) = |\{u \neq v : u \to^* v\}|.$$

Usage

```
centrality_prestige_domain(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`.

Details

The measure needs a directed network. On undirected input every score is `NA` with a `cograph_undefined_measure` warning. Edge weights are ignored. The score is a whole number between 0 and $n - 1$. The values equal `sna::prestige(cmode = "domain")`.

Value

A named numeric vector with one score per node, in input node order.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

`centrality_prestige_domain_proximity`, `centrality_reaching_local`, `centrality`.

Examples

```
centrality_prestige_domain(regulation_net)
```

`centrality_prestige_domain_proximity` *Domain Proximity Prestige*

Description

Domain proximity prestige (Wasserman and Faust 1994) combines the number of nodes that reach a node with their distance to it:

$$PD(v) = \frac{R_v^2}{(n-1)\, D_v},$$

where R_v is the number of other nodes with a directed path to v and D_v the sum of their hop distances to v.

Usage

```
centrality_prestige_domain_proximity(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`.

Details

The measure needs a directed network. On undirected input every score is NA with a cograph_undefined_measure warning. Edge weights are ignored. A node that no other node reaches scores 0, and the score lies between 0 and 1. On strongly connected networks the values equal sna::prestige(cmode = "domain.proximity"). On other networks sna sets some scores to 0, because its sum multiplies an infinite distance by zero; cograph sums the finite distances only.

Value

A named numeric vector with one score per node, in input node order.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

centrality_prestige_domain, centrality_reaching_local, centrality.

Examples

```
centrality_prestige_domain_proximity(regulation_net)
```

---

centrality_proximal_betweenness
Proximal Betweenness Centrality

Description

Proximal betweenness (Brandes 2008, section 3.2) is the fraction of shortest paths on which a node is the last intermediate node before the destination (proximal source) or the first intermediate node after the source (proximal target). Each reachable ordered pair of nodes contributes one unit, divided equally among its shortest paths.

Usage

```
centrality_proximal_betweenness(x, proximal_variant = "source", ...)
```

Arguments

x	Network input accepted by centrality.
proximal_variant	"source" (default), "target", "sum" or "union".
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

Shortest paths are counted with unit edge lengths on the simple network, so direction is kept and weights, loops and parallel edges are ignored. Ordered pairs are counted on undirected networks as well, so the scores are not halved. Endpoints are excluded, and paths with fewer than two edges contribute nothing. On an undirected network the source and target variants agree, and `"sum"` is twice either of them. The `"union"` variant counts a node once when a two-edge path makes it both proximal source and proximal target. Isolated nodes and every node of a complete graph score zero.

Value

A named numeric vector with one score per node, in input node order.

References

Brandes, U. (2008). On variants of shortest-path betweenness centrality and their generic computation. Social Networks, 30, 136-145. doi:10.1016/j.socnet.2007.11.001.

See Also

`centrality_betweenness`, `centrality_stress`, `centrality`.

Examples

```
centrality_proximal_betweenness(regulation_net)
```

`centrality_radiality` *Radiality Centrality*

Description

Radiality (Valente and Foreman 1998) reverses each distance against the diameter D and averages over the network:

$$R(i) = \frac{1}{n-1} \sum_{j:\, d_{ij} < \infty} (D + 1 - d_{ij}).$$

Nodes close to the others score high.

Usage

```
centrality_radiality(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which inverts for tna input only), `alpha` (inversion exponent, default 1) and `cutoff` (largest distance counted, default -1 for no limit).

Details

The sum includes the node itself, which contributes $D+1$, and the values equal `centiserve::radiality()` on undirected networks. Unreachable nodes contribute 0. Edge weights are read as distances; `weighted = FALSE` uses hop counts, and `invert_weights = TRUE` converts a weight w to the distance $1/w^{\alpha}$. `mode = "all"` treats edges as undirected, `"out"` uses distances from the node and `"in"` distances to it. The diameter is taken on the stored edge weights in the direction of the network, whatever `mode` and `invert_weights` are. A single-node network gives `NA`.

Value

A named numeric vector with one score per node, in input node order.

References

Valente, T. W., & Foreman, R. K. (1998). Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. Social Networks, 20(1), 89-105. doi:10.1016/S03788733(97)000075.

See Also

`centrality_integration`, `centrality_closeness`, `centrality`.

Examples

```
centrality_radiality(regulation_net)
```

`centrality_random_walk` *Random Walk Centrality*

Description

Random walk centrality is the inverse of the summed random-walk distances from a node to the others:

$$RW(i) = \left(\sum_{j} \frac{m_{ij} + m_{ji}}{2} \right)^{-1},$$

where m_{ij} is the mean first passage time from i to j of a walk that moves to each out-neighbor with equal probability.

Usage

```
centrality_random_walk(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`.

Details

Edge weights are ignored. The measure is defined for connected undirected and strongly connected directed networks. On any other network every score is `NA`, with a `cograph_undefined_measure` warning, because some passage times are infinite. The passage times are symmetrized before the sum, so the values differ from `tidygraph::centrality_random_walk()`, which sums them unsymmetrized.

Value

A named numeric vector with one score per node, in input node order.

See Also

`centrality_markov`, `centrality_current_flow_closeness`, `centrality`.

Examples

```
centrality_random_walk(regulation_net)
```

---

centrality_random_walk_decay

Random Walk Decay Centrality

Description

Random walk decay centrality (Was, Rahwan and Skibski 2019) sums, over all starting nodes u, the starting weight b_u times the expected discounted first arrival of a random walk from u at node v. With T_v the first arrival time and a the decay factor `rwd_decay`,

$$RWD_v = \sum_u b_u \, E_u \left[a^{T_v}; T_v < \infty \right].$$

Usage

```
centrality_random_walk_decay(x, rwd_decay = 0.5, rwd_node_weights = NULL, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`rwd_decay`	Decay factor a, a finite number in $[0, 1)$. Default 0.5.
`rwd_node_weights`	Nonnegative starting weights b, one per node. `NULL` (default) gives every node weight one. A named vector is matched to the node names.
`...`	Further arguments to `centrality`. The measure uses `weighted` (use edge weights, default `TRUE`) and `normalized` (divide by the maximum, default `FALSE`).

Details

The walk follows outgoing edges with probability proportional to their weights, and an undirected edge is traversed in both directions. A walk that reaches a node without outgoing edges stops. Loops are kept as transitions that stay at the node unless `loops = FALSE`. The start counts as an arrival at time zero, so an isolated node scores its own starting weight and `rwd_decay = 0` returns the starting weights. Edge weights must be finite and nonnegative, and `weighted = FALSE` ignores edge weights while keeping `rwd_node_weights`. Invalid `rwd_decay` or `rwd_node_weights`, and raw scores that overflow, raise an error. Example 3 of the paper contains inconsistent numerical values, and the implementation follows Definition 1, equation 6.

Value

A named numeric vector with one score per node, in input node order.

References

Was, T., Rahwan, T., & Skibski, O. (2019). Random Walk Decay Centrality. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 2197-2204. doi:10.1609/aaai.v33i01.33012197.

See Also

`centrality_pagerank`, `centrality_random_walk`, `centrality`.

Examples

```
centrality_random_walk_decay(regulation_net)
```

---

centrality_reaching_local

Local Reaching Centrality

Description

Local reaching centrality (Mones et al. 2012) measures how much of the network a node reaches. On an unweighted directed network it is the share of the other nodes reachable from the node. On an unweighted undirected network it is the mean inverse distance to the other nodes, harmonic centrality divided by $n - 1$.

Usage

```
centrality_reaching_local(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`.

Details

A network counts as weighted unless every weight is 1, and `weighted = FALSE` gives the unweighted form. In the weighted form a shortest path uses the edge lengths W/w_e, with W the total edge weight, and each reached node contributes the mean edge weight along its path; the sum is divided by $n-1$. With `mode = "out"` the weighted values equal `networkx.local_reaching_centrality()` with `normalized = False`. `mode = "all"` treats edges as undirected, and `"in"` counts the nodes that reach the node. A negative weight raises an error. `reaching_global` is the network-level hierarchy measure built from these scores.

Value

A named numeric vector with one score per node, in input node order.

References

Mones, E., Vicsek, L., & Vicsek, T. (2012). Hierarchy measure for complex networks. *PLoS ONE*, 7(3), e33799.

See Also

`reaching_global`, `centrality_harmonic`, `centrality`.

Examples

```
centrality_reaching_local(regulation_net)
```

`centrality_relative_entropy`

Relative-Entropy Integrated Evaluation

Description

The relative-entropy integrated evaluation (Chen, Wang and Luo 2016) combines several centrality indexes into one score. Each index is turned into a distribution over the nodes, and the score is the distribution with the smallest total relative entropy to all of them, which is their normalized geometric mean:

$$w_i = \frac{\prod_{j=1}^{m} u_{ji}^{1/m}}{\sum_l \prod_{j=1}^{m} u_{jl}^{1/m}}$$

A positive index (larger is more important) maps to $u_i = C_i / \sum_j C_j$, and a negative index (smaller is more important) maps to the normalized complement $1 - C_i / \sum_j C_j$.

Usage

```
centrality_relative_entropy(
  x,
  re_indexes = c("degree", "closeness", "betweenness", "constraint"),
  re_negative = NULL,
  ...
)
```

Arguments

x	Network input accepted by centrality.
re_indexes	Character vector of constituent indexes, without repeats. The default c("degree", "closeness", "betweenness", "constraint") is the source's four-index set.
re_negative	Character vector naming the members of re_indexes treated as negative. The default NULL uses the source's declarations, which make constraint and largest_component negative. character(0) treats every index as positive.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights and loops are ignored. The available indexes are "degree", "closeness" (the reciprocal of the distance sum over reachable partners), "betweenness" (summed over ordered pairs), "constraint" (the source's equation 6, whose outer sum runs over every other node), "n_components" and "largest_component" (the components left after deleting the node). The scores sum to one, and a node that scores zero on any index scores zero overall. An index that is zero at every node, such as betweenness on a complete graph, raises a cograph_undefined_index error when the measure is requested by name. Inside type = "all" the same case gives a cograph_undefined_measure warning and an NA column. The source's equation 6 differs from Burt's constraint as computed by igraph::constraint().

Value

A named numeric vector with one score per node, in input node order, summing to one. With normalized = TRUE the scores are divided by their maximum and no longer sum to one.

References

Chen, B., Wang, Z. and Luo, C. (2016). Integrated evaluation approach for node importance of complex networks based on relative entropy. Journal of Systems Engineering and Electronics, 27(6), 1219-1226. doi:10.21629/JSEE.2016.06.10.

See Also

centrality_bridging, list_centralities, centrality.

Examples

```
centrality_relative_entropy(regulation_net)
```

`centrality_residual_closeness` *Residual Closeness Centrality*

Description

Residual closeness (Dangalchev 2006) sums distances that decay by half with every step:

$$C(i) = \sum_j 2^{-d_{ij}},$$

with $2^{-\infty} = 0$, so the score is defined on disconnected networks.

Usage

```
centrality_residual_closeness(x, mode = "all", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`mode`	Direction for directed networks: `"all"` (default), `"out"` or `"in"`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (default `TRUE`), `invert_weights` (default `NULL`, which inverts for tna input only), `alpha` (inversion exponent, default 1) and `cutoff` (largest distance counted, default -1 for no limit).

Details

The sum includes the node itself, which adds 1, and the values equal `centiserve::closeness.residual()` on undirected networks. Edge weights are read as distances; `weighted = FALSE` uses hop counts, and `invert_weights = TRUE` converts a weight w to the distance $1/w^{\alpha}$. `mode = "all"` treats edges as undirected, `"out"` uses distances from the node and `"in"` distances to it. `centrality_dangalchev` returns the same values.

Value

A named numeric vector with one score per node, in input node order.

References

Dangalchev, C. (2006). Residual closeness in networks. Physica A, 365(2), 556-564. doi:10.1016/j.physa.2005.12.020.

See Also

centrality_dangalchev, centrality_decay, centrality.

Examples

```
centrality_residual_closeness(regulation_net)
```

`centrality_resistance_curvature` *Resistance Curvature*

Description

The node resistance curvature of Devriendt and Lambiotte (2022) is

$$p_i = 1 - \frac{1}{2} \sum_{j \sim i} w_{ij} R_{ij},$$

where the weights w_{ij} are conductances and R_{ij} is the effective resistance. It equals one minus half the expected degree of the node in a random spanning tree of its component.

Usage

```
centrality_resistance_curvature(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses weighted (default TRUE) and normalized (default FALSE), which divides by the largest score and leaves negative scores negative.

Details

Edge weights are conductances, and weighted = FALSE uses the simple undirected skeleton. On a weighted directed network the two arcs between a pair are added. Self-loops are removed, mode has no effect, and negative or non-finite weights raise an error. Scores can be negative at tree-like junctions. On a tree the score is one minus half the degree, on an unweighted cycle or complete graph of n nodes it is $1/n$, and an isolated node scores 1. The raw scores sum to the number of components.

Value

A named numeric vector with one score per node, in input node order.

References

Devriendt, K., & Lambiotte, R. (2022). Discrete curvature on graphs from the effective resistance. Journal of Physics: Complexity, 3, 025008. doi:10.1088/2632072X/ac730d.

See Also

centrality_current_flow_closeness, centrality_dynamical_importance, centrality.

Examples

```
centrality_resistance_curvature(regulation_net)
```

`centrality_rsp_betweenness`

Randomized Shortest Paths Betweenness Centrality

Description

Randomized shortest paths betweenness (Kivimaki et al. 2016) scores a node by the expected number of visits it receives from absorbing walks between every ordered source-target pair, under a Boltzmann distribution that favors low-cost walks with inverse temperature β. Large β approaches shortest-path betweenness, and $\beta \to 0$ approaches a random-walk quantity that is proportional to degree on an undirected graph. With $Z = (I - W)^{-1}$ and $W = (D^{-1}A) \circ \exp(-\beta C)$:

$$bet_i = \sum_{s,t} \left(\frac{z_{si}}{z_{st}} - \frac{z_{ti}}{z_{tt}} \right) z_{it}$$

Usage

```
centrality_rsp_betweenness(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses rsp_beta (inverse temperature, default 0.01), rsp_cost ("inverse" (default) or "weight") and weighted (use edge weights, default TRUE).

Details

Direction and edge weights are used and loops are dropped. mode, cutoff and invert_weights have no effect. The cost C is $1/w$ with rsp_cost = "inverse" and w with "weight", and with weighted = FALSE every arc costs one. Following the source, a pair with no connecting path contributes zero, so scores are component-local, and nodes with no out-edges score zero. A rsp_beta at or below zero raises a cograph_bad_parameter error, and negative or non-finite weights raise a cograph_bad_input error. The measure needs one dense matrix inverse, so it is costly and is computed only when requested by name or through include.

Value

A named numeric vector with one score per node, in input node order.

References

Kivimaki, I., Lebichot, B., Saramaki, J. and Saerens, M. (2016). Two betweenness centrality measures based on Randomized Shortest Paths. Scientific Reports, 6, 19668. doi:10.1038/srep19668.

See Also

centrality_betweenness, centrality_current_flow_betweenness, centrality.

Examples

```
centrality_rsp_betweenness(regulation_net)
```

---

centrality_rumor *Rumor Centrality*

Description

Rumor centrality (Shah and Zaman 2010, 2011) is the maximum-likelihood score for the source of a rumor that has spread to every node under the susceptible-infected model. On a tree rooted at v it counts the spreading orders that start at v,

$$R(v) = \frac{N!}{\prod_u T_u^v},$$

where T_u^v is the size of the subtree rooted at u. On a general graph R is evaluated on the breadth-first tree rooted at each node.

Usage

```
centrality_rumor(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized.

Details

The value is returned as $\log R(v)$ with the natural logarithm, because $N!$ overflows beyond 170 nodes. N is the size of the component of the node, so a disconnected network is scored component by component and an isolated node scores 0. The breadth-first tree attaches each node to the earliest discovered node of the previous layer, scanning neighbors in node order. Direction, edge weights and self-loops are ignored. Higher values mark more plausible origins.

Value

A named numeric vector with one score per node, in input node order, holding $\log R$.

References

Shah, D., & Zaman, T. (2010). Detecting sources of computer viruses in networks: theory and experiment. ACM SIGMETRICS, 203-214.

Shah, D., & Zaman, T. (2011). Rumors in a network: Who's the culprit? IEEE Transactions on Information Theory, 57(8), 5163-5181.

See Also

centrality_closeness, centrality.

Examples

```
centrality_rumor(regulation_net)
```

---

centrality_s_shell *s-shell Index*

Description

The s-shell index (Liu, Tang, Do and Hui 2017) peels the network like the k-shell decomposition but by a strength computed from the topology. Each link gets the asymmetric weight

$$w_{ij} = 1 + (k_i \, k_j^{out})^a,$$

where k_j^{out} counts the neighbors of j outside the closed neighborhood of i, and the strength of i is $s_i = \sum_{j \in N(i)} w_{ij}$. At each step the nodes at the minimum remaining strength are removed, the removals cascade, and the removed nodes receive the next shell index.

Usage

```
centrality_s_shell(x, s_shell_a = 0.5, ...)
```

Arguments

x	Network input accepted by centrality.
s_shell_a	Exponent a of the link weights (default 0.5, the value the source recommends).
...	Further arguments to centrality.

Details

Direction, edge weights and self-loops are ignored. The index is an ordinal counter starting at 1 for the outermost shell, so values are comparable within a network only. Higher indices mark more central nodes. Isolated nodes form shell 1 on their own and shift every other shell up by one. With $a = 0$ the shells are the dense ranks of the k-core numbers. A s_shell_a that is not a single non-negative number raises an error of class cograph_bad_parameter.

Value

A named integer vector with one shell index per node, in input node order.

References

Liu, Y., Tang, M., Do, Y., & Hui, P. M. (2017). Accurate ranking of influential spreaders in networks based on dynamically asymmetric link weights. Physical Review E, 96(2), 022323.

See Also

centrality_coreness, centrality_weighted_kshell, centrality.

Examples

```
centrality_s_shell(regulation_net)
```

---

centrality_salsa *SALSA Authority Centrality*

Description

SALSA (Lempel and Moran 2000) ranks authorities by a random walk that alternates between following an edge backward and forward. The authority score of a node is its entry in the stationary distribution of the chain

$$\tilde{A} = W_c^T W_r,$$

where W_r and W_c are the row-normalized and column-normalized adjacency matrices.

Usage

```
centrality_salsa(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

The measure needs a directed network. On undirected input every score is NA with a cograph_undefined_measure warning. Edge weights are ignored. The scores are scaled so that the largest is 1, and a node without incoming edges scores 0.

Value

A named numeric vector with one score per node, in input node order.

References

Lempel, R., & Moran, S. (2000). The stochastic approach for link-structure analysis (SALSA) and the TKC effect. Computer Networks, 33(1-6), 387-401. doi:10.1016/S13891286(00)000347.

See Also

centrality_authority, centrality_pagerank, centrality.

Examples

```
centrality_salsa(regulation_net)
```

---

centrality_semilocal *Semi-Local Centrality*

Description

Semi-local centrality (Chen et al. 2012) sums, over the neighbors u of a node and the neighbors w of each u, the number $N(w)$ of nodes within two steps of w:

$$C_L(v) = \sum_{u \in \Gamma(v)} \sum_{w \in \Gamma(u)} N(w).$$

Usage

```
centrality_semilocal(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality.

Details

Edge weights are ignored. mode selects the neighbors, and with mode = "all" on a directed network a reciprocated tie counts twice. An isolated node scores 0. On undirected networks the values equal centiserve::semilocal().

Value

A named numeric vector with one score per node, in input node order.

References

Chen, D., Lu, L., Shang, M.-S., Zhang, Y.-C., & Zhou, T. (2012). Identifying influential nodes in complex networks. Physica A, 391(4), 1777-1787. doi:10.1016/j.physa.2011.09.017.

See Also

centrality_laplacian, centrality_degree, centrality.

Examples

```
centrality_semilocal(regulation_net)
```

`centrality_shapley_game1`
Shapley Value Centrality

Description

Shapley value centrality (Michalak et al. 2013) is the Shapley value of each node in a coalition game whose worth $v(C)$ counts the nodes a coalition C covers. Each game has an exact closed form. In game 1 a coalition covers its members and their neighbors, and

$$SV(v) = \sum_{u \in \{v\} \cup N(v)} \frac{1}{1 + k_u}.$$

In game 2 a coalition covers its members and the nodes with at least k neighbors in it. In game 3 it covers the nodes within `shapley_cutoff` hops of it, and $N(v)$ is replaced by the set of nodes within that distance.

Usage

```
centrality_shapley_game1(x, ...)

centrality_shapley_game2(x, shapley_k = 2, ...)

centrality_shapley_game3(x, shapley_cutoff = 2, ...)
```

Arguments

`x`	Network input accepted by centrality.
`...`	Further arguments to centrality, such as `normalized`.
`shapley_k`	Neighbor threshold k for game 2 (default 2).
`shapley_cutoff`	Hop cutoff for game 3 (default 2).

Details

The values of every game sum to the number of nodes. Game 2 with $k = 1$ and game 3 with cutoff 1 equal game 1. Degrees exclude self-loops, and edge weights are ignored. On a directed network coverage runs along out-edges and the denominators use in-degrees, the extension the source states. Distances in game 3 are hop counts.

Value

A named numeric vector with one Shapley value per node, in input node order.

References

Michalak, T. P., Aadithya, K. V., Szczepanski, P. L., Ravindran, B., & Jennings, N. R. (2013). Efficient computation of the Shapley value for game-theoretic network centrality. Journal of Artificial Intelligence Research, 46, 607-650.

See Also

centrality_degree, centrality.

Examples

```
centrality_shapley_game1(regulation_net)
centrality_shapley_game2(regulation_net)
centrality_shapley_game3(regulation_net)
```

---

centrality_spectralrank

SpectralRank Centrality

Description

SpectralRank (Xu et al. 2019) is the positive eigenvector of the largest eigenvalue of the adjacency matrix augmented by a ground node, which is linked in both directions to every node with unit weight:

$$B = \begin{pmatrix} A+P & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix}.$$

The diagonal prior P is zero for ordinary SpectralRank, and a nonnegative prior gives the weighted SpectralRank family of the paper.

Usage

```
centrality_spectralrank(x, sr_prior = 0, ...)
```

Arguments

x	Network input accepted by centrality.
sr_prior	Diagonal prior P, a nonnegative scalar or one value per node. A named vector is matched to the node names. Default 0, which gives ordinary SpectralRank.
...	Further arguments to centrality. The measure uses weighted (use edge weights, default TRUE).

Details

The eigenvector is divided by its maximum over all nodes including the ground node, which is then dropped, so the largest returned score can be below one. `normalized = TRUE` further divides by the maximum over the original nodes. An arc from i to j contributes the score of j to i. Edge weights must be finite and nonnegative, `weighted = FALSE` gives every edge weight one, and loops are removed. Every node, isolated nodes included, scores positive, and without edges or priors each of n nodes scores $1/\sqrt{n}$. The update line of Algorithm 1 in the paper omits P, and the implementation follows section III-A2 with $B = \tilde{A} + P$. An invalid `sr_prior` or an unresolved eigenvector raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Xu, S., Wang, P., Zhang, C.-X. and Lu, J. (2019). Spectral Learning Algorithm Reveals Propagation Capability of Complex Networks. IEEE Transactions on Cybernetics, 49(12), 4253-4261. doi:10.1109/TCYB.2018.2861568.

See Also

`centrality_eigenvector`, `centrality_leaderrank`, `centrality`.

Examples

```
centrality_spectralrank(regulation_net)
```

`centrality_strength` *Strength Centrality*

Description

Strength (Barrat et al. 2004) is the sum of the weights of the edges incident to a node, the weighted counterpart of degree. With `mode = "in"` it sums incoming weights and with `mode = "out"` outgoing weights. `centrality_instrength()` and `centrality_outstrength()` are these two forms.

Usage

```
centrality_strength(x, mode = "all", ...)

centrality_instrength(x, ...)

centrality_outstrength(x, ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "in" or "out".
...	Further arguments to centrality. The measure uses loops (keep self-loops, default TRUE) and normalized (default FALSE).

Details

The stored weights are summed, and weighted = FALSE gives every edge weight one, so the scores equal centrality_degree. A self-loop counts twice on an undirected network and under mode = "all", and once under "in" or "out"; loops = FALSE drops it. Negative weights are summed with their sign. normalized = TRUE divides the scores by their maximum.

Value

A named numeric vector with one score per node, in input node order.

References

Barrat, A., Barthelemy, M., Pastor-Satorras, R., & Vespignani, A. (2004). The architecture of complex weighted networks. Proceedings of the National Academy of Sciences, 101(11), 3747-3752. doi:10.1073/pnas.0400087101.

See Also

centrality_degree, centrality.

Examples

```
centrality_strength(regulation_net)
```

---

centrality_stress *Stress Centrality*

Description

Stress centrality (Shimbel 1953) counts the shortest paths between other pairs of nodes that pass through a node:

$$S(v) = \sum_{s \neq v \neq t} \sigma_{st}(v),$$

where $\sigma_{st}(v)$ is the number of shortest paths from s to t through v. Betweenness divides each count by the number of shortest paths between the pair; stress keeps the counts.

Usage

```
centrality_stress(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only) and alpha (inversion exponent, default 1).

Details

Edge weights are read as distances. weighted = FALSE uses hop counts, and invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. On a directed network the paths follow edge direction, and on an undirected network each pair is counted once. The values equal sna::stresscent().

Value

A named numeric vector with one score per node, in input node order.

References

Shimbel, A. (1953). Structural parameters of communication networks. The Bulletin of Mathematical Biophysics, 15(4), 501-507. doi:10.1007/BF02476438.

See Also

centrality_betweenness, centrality_load, centrality.

Examples

```
centrality_stress(regulation_net)
```

---

centrality_subgraph *Subgraph Centrality*

Description

Subgraph centrality (Estrada and Rodriguez-Velazquez 2005) counts the closed walks that start and end at a node, weighting a walk of length k by $1/k!$:

$$SC(i) = \sum_{k=0}^{\infty} \frac{(A^k)_{ii}}{k!} = \left(e^A\right)_{ii}.$$

Usage

```
centrality_subgraph(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

A is the binary adjacency matrix without self-loops, so edge weights are ignored. On a directed network A is replaced by $A + A^T$, so a reciprocated pair has entry 2; the values then equal `igraph::subgraph_centrality()`. An empty network raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Estrada, E., & Rodriguez-Velazquez, J. A. (2005). Subgraph centrality in complex networks. Physical Review E, 71(5), 056103. doi:10.1103/PhysRevE.71.056103.

See Also

`centrality_communicability`, `centrality_eigenvector`, `centrality`.

Examples

```
centrality_subgraph(regulation_net)
```

`centrality_topological_coefficient` *Topological Coefficient*

Description

The topological coefficient (Stelzl et al. 2005) measures how far a node shares neighbors with the nodes it is linked to through a common neighbor:

$$T(v) = \frac{\sum_{u \in U_v} J(v,u)}{|U_v|\, k_v},$$

where U_v is the set of nodes that share at least one neighbor with v, $J(v,u)$ is the number of shared neighbors plus one when u and v are linked, and k_v is the degree.

Usage

```
centrality_topological_coefficient(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`.

Details

Edge weights and direction are ignored. A node that shares no neighbor with any other node scores 0, as does an isolated node. On undirected networks the values equal `centiserve::topocoefficient()`.

Value

A named numeric vector with one score per node, in input node order.

References

Stelzl, U., Worm, U., Lalowski, M., Haenig, C., Brembeck, F. H., Goehler, H., et al. (2005). A human protein-protein interaction network: A resource for annotating the proteome. Cell, 122(6), 957-968. doi:10.1016/j.cell.2005.08.029.

See Also

`centrality_transitivity`, `centrality_lac`, `centrality`.

Examples

```
centrality_topological_coefficient(regulation_net)
```

---

centrality_transitivity

Local Transitivity

Description

Local transitivity, the clustering coefficient of Watts and Strogatz (1998), is the share of pairs of a node's neighbors that are themselves linked:

$$C_i = \frac{2T_i}{k_i(k_i - 1)},$$

where T_i is the number of triangles through node i and k_i its degree.

Usage

```
centrality_transitivity(x, transitivity_type = "local", isolates = "nan", ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`transitivity_type`	One of `"local"` (default), `"global"`, `"undirected"`, `"localundirected"`, `"barrat"`, `"weighted"` or `"onnela"`.
`isolates`	Value for nodes with fewer than two ties: `"nan"` (default) or `"zero"`.
`...`	Further arguments to `centrality`. The measure uses `tna_network` (default `NULL`, which detects tna input).

Details

Triangles and degrees are counted on the undirected skeleton, so a reciprocated tie counts once, and edge weights are ignored. The values equal `igraph::transitivity(type = "local")` on directed and undirected networks. `"localundirected"` gives the same values as `"local"`, as in igraph. `"global"` and `"undirected"` return the network-level ratio of closed to connected triples for every node. `"barrat"` and `"weighted"` compute the weighted coefficient of Barrat et al. (2004) and raise a `cograph_directed_unsupported` error on directed input. `"onnela"` computes the weighted clustering coefficient of Zhang and Horvath (2005), $(M^3)_{ii}/(s_i^2 - \sum_j M_{ij}^2)$ on $M = W + W^T$ with strengths s_i. This is the value `tna::centralities()` reports as Clustering, and it is the default for tna input. The option keeps the name `"onnela"` used by tna, but the formula multiplies the raw weights of a triangle and differs from the geometric-mean form of Onnela et al. (2005). Under the local and Barrat types a node with fewer than two neighbors is `NaN`, or 0 with `isolates = "zero"`.

Value

A named numeric vector with one score per node, in input node order.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. Nature, 393(6684), 440-442. doi:10.1038/30918.

Barrat, A., Barthelemy, M., Pastor-Satorras, R., & Vespignani, A. (2004). The architecture of complex weighted networks. Proceedings of the National Academy of Sciences, 101(11), 3747-3752. doi:10.1073/pnas.0400087101.

Zhang, B., & Horvath, S. (2005). A general framework for weighted gene co-expression network analysis. Statistical Applications in Genetics and Molecular Biology, 4(1), Article 17. doi:10.2202/15446115.1128.

Onnela, J.-P., Saramaki, J., Kertesz, J., & Kaski, K. (2005). Intensity and coherence of motifs in weighted complex networks. Physical Review E, 71(6), 065103. doi:10.1103/PhysRevE.71.065103.

See Also

`centrality_clusterrank`, `centrality_topological_coefficient`, `centrality`.

Examples

```
centrality_transitivity(regulation_net)
```

`centrality_truss` *Truss, Mixed-Degree Decomposition, Bridging Coefficient, Godfather and Support*

Description

The truss number of a node (Malliaros et al. 2016) is the largest truss number of an incident edge, where a k-truss requires every edge to lie in at least $k - 2$ triangles of the subgraph. Mixed-degree decomposition (Zeng and Zhang 2013) peels nodes by residual degree plus `mdd_lambda` times exhausted degree. The bridging coefficient (Hwang et al. 2008) is $(1/d_i)/\sum_{j \in N(i)} 1/d_j$. The Godfather index (Jackson 2020) counts the unordered pairs of neighbors with no edge between them, and the support (Jackson 2020) counts the neighbors that share at least one common neighbor with the node.

Usage

```
centrality_truss(x, ...)

centrality_mdd(x, mdd_lambda = 0.7, ...)

centrality_bridging_coefficient(x, ...)

centrality_godfather(x, ...)

centrality_support(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`, such as `normalized`.
`mdd_lambda`	Weight of the exhausted degree, between 0 and 1. Default 0.7, the value of the worked example of Zeng and Zhang (2013). A value outside that range raises an error.

Details

All five measures use the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Isolated nodes score 0. An edge outside every triangle has truss number 2, and a node of a complete graph on k nodes has truss number k, as in NetworkX. Sources that label trusses by the triangle threshold report values two smaller. `mdd_lambda = 0` gives the k-core number and `mdd_lambda = 1` gives the degree. The bridging coefficient is the factor that bridging centrality multiplies with betweenness. LocalRank (Chen et al. 2012), listed in the Centrality Zoo, is `centrality_semilocal`.

Value

A named numeric vector with one score per node, in input node order.

References

Malliaros, F. D., Rossi, M. E. G., & Vazirgiannis, M. (2016). Locating influential nodes in complex networks. Scientific Reports, 6, 19307. doi:10.1038/srep19307.

Zeng, A., & Zhang, C. J. (2013). Ranking spreaders by decomposing complex networks. Physics Letters A, 377, 1031-1035. doi:10.1016/j.physleta.2013.02.039.

Hwang, W., Kim, T., Ramanathan, M., & Zhang, A. (2008). Bridging centrality: graph mining from element level to group level. KDD '08, 336-344. doi:10.1145/1401890.1401934.

Jackson, M. O. (2020). A typology of social capital and associated network measures. Social Choice and Welfare, 54, 311-336. doi:10.1007/s00355019011893.

Chen, D., Lu, L., Shang, M. S., Zhang, Y. C., & Zhou, T. (2012). Identifying influential nodes in complex networks. Physica A, 391, 1777-1787. doi:10.1016/j.physa.2011.09.017.

See Also

centrality_coreness, centrality_bridging, centrality.

Examples

```
centrality_truss(regulation_net)
centrality_mdd(regulation_net)
centrality_bridging_coefficient(regulation_net)
centrality_godfather(regulation_net)
centrality_support(regulation_net)
```

---

centrality_trust_pagerank

Trust-PageRank

Description

Trust-PageRank (Sheng et al. 2020) is a damped PageRank in which a node passes its score to a neighbor in proportion to a trust value. The trust value mixes the SimRank similarity s of the two nodes with a degree ratio:

$$T(i,j) = (1-k)\frac{s(i,j)}{\sum_{l \in N_j} s(j,l)} + k\,\frac{d_i}{\sum_{l \in N_j} d_l}, \qquad TPR_i = \frac{1-\alpha}{n} + \alpha \sum_{j \in N_i} T(i,j)\,TPR_j.$$

Usage

```
centrality_trust_pagerank(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality. The measure uses tpr_alpha (damping factor, default 0.85), tpr_k (weight of the degree ratio, default 0.85), tpr_decay (SimRank decay constant, default 1), tpr_tol (relative tolerance, default 1e-14) and tpr_max_iter (default 1000).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. The similarity recursion runs over adjacent pairs and is anchored by triangles. Every node of a component that has edges but no triangle is returned as `NA` with a `cograph_undefined_measure` warning, and an isolated node scores $(1-\alpha)/n$. Both the similarity and the score iterations stop on a relative tolerance, and an iteration that reaches `tpr_max_iter` raises `cograph_no_converge`. The Centrality Zoo attributes the measure to Sheng et al., Physica A 541:123262, which defines a different index. The measure is defined in the Algorithms paper cited below.

Value

A named numeric vector with one score per node, in input node order. The scores sum to one on a network with no isolated node and no undefined component.

References

Sheng, J., Zhu, J., Wang, Y., Wang, B. and Hou, Z. (2020). Identifying Influential Nodes of Complex Networks Based on Trust-Value. Algorithms, 13(11), 280. doi:10.3390/a13110280.

See Also

`centrality_pagerank`, `centrality`.

Examples

```
centrality_trust_pagerank(regulation_net)
```

`centrality_two_way_rw` *Two-Way Random Walk Betweenness*

Description

Two-way random walk betweenness (Curado et al. 2022) counts how often a node lies on the strongest two-way route between a pair. For every unordered pair (i, j) the two-step transfer $P_{itj} = w_{it}w_{tj}/(d_i d_j)$ is combined into $T_{ij}[t, k] = P_{itj}P_{jki}$, the diagonal is dropped, and the largest entry credits one count to t and one to k. The score of a node is its total count over all pairs.

Usage

```
centrality_two_way_rw(x, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. The measure uses `weighted` (use edge weights, default `TRUE`).

Details

Edge weights are used, and direction and self-loops are ignored. Ties in the maximum go to the first entry in row-major order. Nodes that never lie on a winning two-way route score 0. The source divides the transfer by $d_i d_j$, where a random-walk probability would divide by $d_i d_t$. The formula is implemented as printed.

Value

A named numeric vector with one count per node, in input node order.

References

Curado, M., Rodriguez, R., Tortosa, L., & Vicent, J. F. (2022). A new centrality measure in dense networks based on two-way random walk betweenness. Applied Mathematics and Computation, 412, 126560.

See Also

`centrality_current_flow_betweenness`, `centrality_betweenness`, `centrality`.

Examples

```
centrality_two_way_rw(regulation_net)
```

---

centrality_volume *Volume Centrality*

Description

Volume centrality (Wehmuth and Ziviani 2011, 2013) is the sum of the degrees of all nodes within `volume_radius` hops of a node, the node itself included. Degrees are those of the whole network, so edges that leave the neighborhood also count.

Usage

```
centrality_volume(x, volume_radius = 2, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`volume_radius`	Hop radius: a nonnegative integer or `Inf`. Default 2.
`...`	Further arguments to `centrality`, such as `normalized`.

Details

The measure uses the simple undirected skeleton, so direction, weights, loops and parallel edges are ignored. Isolated nodes score 0. Radius 0 gives the degree, and an infinite radius gives twice the number of edges in the node's component. A radius that is negative or not a whole number raises an error.

Value

A named numeric vector with one score per node, in input node order.

References

Wehmuth, K., & Ziviani, A. (2011). Distributed Assessment of Network Centrality. arXiv:1108.1067.

Wehmuth, K., & Ziviani, A. (2013). DACCER: Distributed Assessment of the Closeness CEntrality Ranking in complex networks. Computer Networks, 57, 2536-2548. doi:10.1016/j.comnet.2013.05.001.

See Also

centrality_kreach, centrality_degree, centrality.

Examples

```
centrality_volume(regulation_net, volume_radius = 1)
```

---

centrality_voterank *VoteRank Centrality*

Description

VoteRank (Zhang et al. 2016) elects spreaders one at a time. Every node votes for its neighbors with a voting ability that starts at 1, the node with the most votes is elected, and each neighbor of the elected node loses $1/\langle k \rangle$ of its ability, where $\langle k \rangle$ is the mean degree. A node elected in round r of m rounds scores $(m + 1 - r)/m$, so the first node elected scores 1.

Usage

```
centrality_voterank(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality.

Details

Edge weights are ignored. On a directed network a node receives the votes of its in-neighbors, the election reduces the ability of the out-neighbors of the elected node, and the mean degree counts in-ties and out-ties. Every node is elected in turn, so on n nodes the scores are $1/n, 2/n, \ldots, 1$. A tie in votes goes to the node listed first.

Value

A named numeric vector with one score per node, in input node order.

References

Zhang, J.-X., Chen, D.-B., Dong, Q., & Zhao, Z.-D. (2016). Identifying a set of influential spreaders in complex networks. Scientific Reports, 6, 27823. doi:10.1038/srep27823.

See Also

centrality_ncvoterank, centrality_wvoterank, centrality.

Examples

```
centrality_voterank(regulation_net)
```

`centrality_weighted_kshell`

Weighted k-shell, Renewed Coreness and Geodesic k-path

Description

Three shell and path counts. The weighted k-shell (Garas, Schweitzer and Havlin 2012) runs the k-shell decomposition on the generalized degree

$$k'_i = \left(k_i^{\alpha} s_i^{\beta}\right)^{1/(\alpha+\beta)},$$

with strength s_i after the weight normalization of the source. Renewed coreness (Liu, Tang, Zhou and Do 2015) gives each link the diffusion importance $D_{ij} = (|N(j) \setminus N[i]| + |N(i) \setminus N[j]|)/2$, removes the links below `renewed_threshold`, and returns the k-core number of the residual graph. Geodesic k-path centrality (Borgatti and Everett 2006) counts the shortest paths of length at most `kpath_k` that start at the node, with multiplicity.

Usage

```
centrality_weighted_kshell(x, wks_alpha = 1, wks_beta = 1, ...)

centrality_renewed_coreness(x, renewed_threshold = 2, ...)

centrality_geodesic_kpath(x, mode = "all", kpath_k = 3, ...)
```

Arguments

`x`	Network input accepted by centrality.
`wks_alpha, wks_beta`	Exponents of degree and strength in the weighted k-shell (default 1 and 1).
`...`	Further arguments to centrality. The weighted k-shell uses `weighted` (use edge weights, default `TRUE`).
`renewed_threshold`	Diffusion-importance threshold for renewed coreness (default 2).
`mode`	For directed networks: `"all"` (default), `"out"` or `"in"`.
`kpath_k`	Maximum path length for geodesic k-path (default 3).

Details

The weighted k-shell uses edge weights, and with unit weights it equals the k-core number. Isolated nodes score 0. Renewed coreness ignores weights, and a clique with no outside links scores 0. Both ignore direction. Geodesic k-path ignores weights and follows `mode`. The Centrality Zoo transcribes the diffusion importance with open neighborhoods, which is off by one, and the closed neighborhoods of the source are used here. `centiserve::geokpath` counts the nodes within k hops, which is the vertex-disjoint variant of Borgatti and Everett.

Value

A named numeric vector with one score per node, in input node order. The weighted k-shell and renewed coreness are integer vectors.

References

Garas, A., Schweitzer, F., & Havlin, S. (2012). A k-shell decomposition method for weighted networks. New Journal of Physics, 14, 083030.

Liu, Y., Tang, M., Zhou, T., & Do, Y. (2015). Improving the accuracy of the k-shell method by removing redundant links: From a perspective of spreading dynamics. Scientific Reports, 5, 13172. doi:10.1038/srep13172.

Borgatti, S. P., & Everett, M. G. (2006). A graph-theoretic perspective on centrality. Social Networks, 28(4), 466-484.

See Also

`centrality_coreness`, `centrality_s_shell`, `centrality`.

Examples

```
centrality_weighted_kshell(regulation_net)
centrality_renewed_coreness(regulation_net)
centrality_geodesic_kpath(regulation_net)
```

---

centrality_weighted_leaderrank

Weighted LeaderRank

Description

Weighted LeaderRank (Li et al. 2014) adds a ground node g that is linked in both directions to every node. Each original arc and each arc into the ground has weight 1, and the arc from the ground to node i has weight $(k_i^{in})^{\alpha}$, where k_i^{in} is the in-degree before the ground is added. The score is the stationary resource of the random walk on the row-normalized weights, started with one unit on every node and the ground included.

Usage

```
centrality_weighted_leaderrank(x, wlr_alpha = 1, ...)
```

Arguments

x	Network input accepted by centrality.
wlr_alpha	Exponent α of the in-degree, a finite number. Default 1, one of the values studied by Li et al. (2014).
...	Further arguments to centrality, such as normalized.

Details

Arcs keep their direction, and an undirected edge counts as two opposite arcs. Edge weights, loops and parallel arcs are ignored, and mode has no effect. The returned scores omit the ground, so they sum to less than $N + 1$. With $\alpha = 0$ every ground arc has weight 1. With a positive α a node with in-degree 0 scores 0, and when every in-degree is 0 all scores are NaN with a cograph_undefined_measure warning. A negative α requires a positive in-degree at every node and raises a cograph_bad_parameter error otherwise, as does a wlr_alpha that is not a finite number.

Value

A named numeric vector with one score per node, in input node order.

References

Li, Q., Zhou, T., Lu, L., & Chen, D. (2014). Identifying influential spreaders by weighted LeaderRank. Physica A, 404, 47-55. doi:10.1016/j.physa.2014.02.041.

See Also

centrality_leaderrank, centrality_adaptive_leaderrank, centrality.

Examples

```
centrality_weighted_leaderrank(regulation_net)
```

---

centrality_wiener *Wiener Index Centrality*

Description

The Wiener centrality of a node is the sum of its shortest-path distances to the nodes it reaches:

$$W(i) = \sum_{j \neq i,\, d_{ij} < \infty} d_{ij}.$$

High values mark peripheral nodes. On a connected undirected network half the sum of the scores is the Wiener index of the network (Wiener 1947).

Usage

```
centrality_wiener(x, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality. The measure uses weighted (default TRUE), invert_weights (default NULL, which inverts for tna input only), alpha (inversion exponent, default 1) and cutoff (largest distance counted, default -1 for no limit).

Details

Edge weights are read as distances. weighted = FALSE uses hop counts, and invert_weights = TRUE converts a weight w to the distance $1/w^{\alpha}$. mode = "all" treats edges as undirected, "out" uses distances from the node and "in" distances to it. Unreachable nodes add nothing, so on a disconnected network a node in a small component also scores low.

Value

A named numeric vector with one score per node, in input node order.

References

Wiener, H. (1947). Structural determination of paraffin boiling points. Journal of the American Chemical Society, 69(1), 17-20. doi:10.1021/ja01193a005.

See Also

centrality_closeness, centrality_average_distance, centrality.

Examples

```
centrality_wiener(regulation_net)
```

---

centrality_within_module_z

Within-Module Degree Z-Score

Description

The within-module degree z-score (Guimera and Nunes Amaral 2005) standardizes the number of ties a node has inside its own community against the other members of that community:

$$z_i = \frac{\kappa_i - \bar{\kappa}_{s_i}}{\sigma_{\kappa_{s_i}}},$$

where κ_i counts the ties of i to its community s_i. High values mark hubs within their community.

Usage

```
centrality_within_module_z(x, membership = NULL, mode = "all", ...)
```

Arguments

x	Network input accepted by centrality.
membership	Community of each node, a vector with one entry per node in input node order (default NULL).
mode	Direction for directed networks: "all" (default), "out" or "in".
...	Further arguments to centrality.

Details

Edge weights are ignored. mode selects the ties counted, and with mode = "all" on a directed network a reciprocated tie counts twice. The standard deviation is the sample value. A community with one member, or whose members all have the same within-community degree, gives NaN. Without membership every score is NA with a warning of classes cograph_bad_membership and cograph_undefined_measure, and a membership of the wrong length raises a cograph_bad_membership error. On undirected networks the values equal brainGraph::within_module_deg_z_score().

Value

A named numeric vector with one score per node, in input node order.

References

Guimera, R., & Nunes Amaral, L. A. (2005). Functional cartography of complex metabolic networks. Nature, 433(7028), 895-900. doi:10.1038/nature03288.

See Also

centrality_participation, centrality_gateway, centrality.

Examples

```
centrality_within_module_z(regulation_net, membership = rep(1:2, each = 5))
```

---

centrality_wvoterank *WVoteRank, EnRenew and VoteRank++*

Description

Three spreader-selection procedures in the VoteRank family that elect one node per round. WVoteRank (Sun et al. 2019) scores a node by $s_v = \sqrt{k_v \sum_{u \in N(v)} v a_u w_{vu}}$ and lowers the ability of the neighbors of a winner by $1/\langle w \rangle$, the inverse of the average strength. EnRenew (Guo et al. 2020) elects the largest neighbor entropy $E_v = -\sum_{u \in N(v)} p_{uv} \ln p_{uv}$, $p_{uv} = k_u / \sum_{l \in N(v)} k_l$, and scales the entropy terms within `enrenew_depth` steps by $1 - 1/(2^{d-1} \ln \langle k \rangle)$. VoteRank++ (Liu et al. 2021) starts from ability $\ln(1 + k_i/k_{\max})$, splits votes in proportion to degree, and multiplies abilities by λ one step from a winner and by $\sqrt{\lambda}$ two steps away.

Usage

```
centrality_wvoterank(x, ...)

centrality_enrenew(x, enrenew_depth = 2, ...)

centrality_voterank_plus(x, voterank_lambda = 0.1, ...)
```

Arguments

`x`	Network input accepted by `centrality`.
`...`	Further arguments to `centrality`. WVoteRank uses `weighted` (use edge weights, default `TRUE`).
`enrenew_depth`	Renewal radius l for EnRenew (default 2).
`voterank_lambda`	Suppression factor λ for VoteRank++ (default 0.1).

Details

Elections continue until every node is placed. The first elected scores 1 and the last $1/n$, so scores lie in $(0, 1]$, and ties go to the lowest node index. Direction and self-loops are ignored. WVoteRank uses edge weights, and with unit weights it is VoteRank with a square-root score. The other two ignore weights. The renewal factor of EnRenew is negative when $\langle k \rangle < e$. EnRenew follows the article where the released code of the authors differs from it. VoteRank++ follows the released code of the authors, including the exclusion of elected nodes from the vote-share denominator.

Value

A named numeric vector with one score per node, in input node order.

References

Sun, H.-L., Chen, D.-B., He, J.-L., & Ch'ng, E. (2019). A voting approach to uncover multiple influential spreaders on weighted networks. Physica A, 519, 303-312.

Guo, C., Yang, L., Guo, X., Pan, J., & Chen, X. (2020). Influential nodes identification in complex networks via information entropy. Entropy, 22(2), 242.

Liu, P., Li, L., Fang, S., & Yao, Y. (2021). Identifying influential nodes in social networks: A voting approach. Chaos, Solitons & Fractals, 152, 111309.

See Also

centrality_voterank, centrality_ncvoterank, centrality.

Examples

```
centrality_wvoterank(regulation_net)
centrality_enrenew(regulation_net)
centrality_voterank_plus(regulation_net)
```

---

centrality_x_degree *X-Degree Centrality*

Description

X-degree (Torres et al. 2021, equation 3.15) counts the oriented nonbacktracking walks of four edges whose middle node is i. It depends only on the degrees d_j of the neighbors of i:

$$Xdeg(i) = \Big(\sum_{j \in N(i)} (d_j - 1) \Big)^2 - \sum_{j \in N(i)} (d_j - 1)^2.$$

Usage

```
centrality_x_degree(x, ...)
```

Arguments

x	Network input accepted by centrality.
...	Further arguments to centrality, such as normalized (divide by the maximum, default FALSE).

Details

The measure is computed on the simple undirected skeleton of the network, so direction, weights, loops and parallel edges are ignored. Isolated nodes and leaves score zero, and every node of a star scores zero. Components are scored independently. The function computes the score on the supplied network. The iterative node-removal immunization procedure of the paper is a separate algorithm.

Value

A named numeric vector with one score per node, in input node order.

References

Torres, L., Chan, K. S., Tong, H., & Eliassi-Rad, T. (2021). Nonbacktracking Eigenvalues under Node Removal: X-Centrality and Targeted Immunization. SIAM Journal on Mathematics of Data Science, 3(2), 656-675. doi:10.1137/20M1352132.

See Also

centrality_degree, centrality_dynamical_importance, centrality.

Examples

```
centrality_x_degree(regulation_net)
```

`centralization` *Centralization index*

Description

Computes Freeman's centralization for degree, betweenness, closeness, or eigenvector centrality.

Usage

```
centralization(
  x,
  measure = c("degree", "betweenness", "closeness", "eigenvector"),
  directed = NULL,
  mode = "all",
  ...
)
```

Arguments

`x`	Network input (matrix, edge-list data frame, igraph, network, cograph_network, tna object).
`measure`	One of `"degree"` (default), `"betweenness"`, `"closeness"` or `"eigenvector"`.
`directed`	Logical or `NULL`. `NULL` (default) auto-detects from matrix symmetry; `TRUE`/`FALSE` forces it.
`mode`	For directed networks: `"all"` (default), `"in"` or `"out"`. Used by `"degree"` and `"closeness"` only.
`...`	Ignored; accepted for call compatibility with the other centrality verbs.

Details

A weighted input carries its weights into betweenness, closeness and eigenvector centrality; degree centralization ignores them.

Value

A single number. It is the summed gap between the most central node and every other node, divided by a theoretical maximum. For degree, betweenness and closeness the maximum is the value of an unweighted star, so unweighted input gives 0 for a perfectly even network and 1 for an undirected star. For eigenvector centrality, whose scores are scaled to a maximum of 1, the divisor is $n - 1$, and a star gives a value below 1. Weighted closeness scales with the inverse of the edge weights, so its centralization can exceed 1. Nodes whose score is `NA` or `NaN` are dropped from the sum. The value is 0 when the network has two or fewer nodes.

Examples

```
cograph::centralization(regulation_net, measure = "degree")
```

`cluster_quality` *Cluster Quality Metrics*

Description

Computes per-cluster and global quality metrics for network partitioning. Supports both binary and weighted networks.

Usage

```
cluster_quality(x, clusters, weighted = TRUE, directed = TRUE)

cqual(x, clusters, weighted = TRUE, directed = TRUE)
```

Arguments

`x`	Adjacency matrix (numeric)
`clusters`	Cluster specification (named list, data frame, or membership vector; see `csum`)
`weighted`	Logical; if TRUE (default), use edge weights; if FALSE, binarize the matrix first
`directed`	Logical; if TRUE (default), treat as directed network

Value

A `cluster_quality` object (a list) with:

`per_cluster`	Data frame, one row per cluster, with columns `cluster` (index), `cluster_name`, `n_nodes`, `internal_edges` (within-cluster weight), `cut_edges` (boundary-crossing weight), `internal_density`, `avg_internal_degree`, `expansion`, `cut_ratio` and `conductance`.
`global`	List with `modularity` (Newman-Girvan, computed on the weighted or binarized matrix), `coverage` (share of total weight that is internal to some cluster) and `n_clusters`.

See `cluster_quality`.

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
cluster_quality(regulation_net, clusters = clusters)
```

---

cluster_significance *Test Significance of Community Structure*

Description

Compares observed modularity against a null model distribution to assess whether the detected community structure is statistically significant.

Usage

```
cluster_significance(
  x,
  communities,
  n_random = 100,
  method = c("configuration", "gnm"),
  null = c("detect", "fixed"),
  seed = NULL
)

csig(
  x,
  communities,
  n_random = 100,
  method = c("configuration", "gnm"),
  null = c("detect", "fixed"),
  seed = NULL
)
```

Arguments

x	Network input: adjacency matrix, igraph object, or cograph_network.
communities	A communities object (from communities or igraph) or a membership vector (integer vector where communities[i] is the community of node i).
n_random	Number of random networks to generate for the null distribution. Default 100.
method	Null model type:

"configuration" (default) Undirected configuration model that preserves the total degree of each node.

"gnm" Erdos-Renyi G(n, m) model with the same number of nodes and edges and the same directedness.

`null` Which null question to answer. Default `"detect"`:

"detect" The null value is the modularity of the partition found by community detection on each null graph.

"fixed" The null value is the modularity of the supplied `communities` membership evaluated on each null graph.

`seed` Random seed for reproducibility. Default NULL.

Details

The function generates `n_random` random networks from the null model. With `null = "detect"`, community detection (Louvain, or fast greedy when Louvain fails) is run on each null network and its modularity is recorded. A low p-value then indicates that the observed partition is stronger than the partitions detection recovers on random networks. With `null = "fixed"`, the supplied membership is evaluated on each null network. A low p-value then indicates that the partition explains more structure in the observed network than in random networks, independently of any detection algorithm.

The observed modularity is computed with the edge weights of `x`. When `x` is weighted, each null network receives the observed edge weights, randomly reassigned to its edges, so the observed and null modularity are on the same scale.

Value

A `cograph_cluster_significance` object with:

observed_modularity Modularity of the input communities

null_mean Mean modularity of random networks

null_sd Standard deviation of null modularity

z_score Standardized score (observed - null_mean) / null_sd, or `NA` when `null_sd` is zero

p_value One-sided upper-tail p-value of `z_score` under the standard normal distribution; `NA` when `null_sd` is zero

null_values Vector of modularity values from null distribution

method Null model method used

null Which null question was asked ("detect" or "fixed")

n_random Number of random networks generated

Printing and plotting

Printing the result shows the null model, the observed and null modularity, the z-score and the p-value. `plot()` on the result plots a histogram of the null modularity values with the observed value marked.

References

Reichardt, J., & Bornholdt, S. (2006). Statistical mechanics of community detection. *Physical Review E*, 74, 016110.

See Also

communities, cluster_quality

Examples

```
comm <- communities(regulation_net, method = "walktrap")
cluster_significance(regulation_net, comm, n_random = 20, seed = 1)
```

---

cograph *Create a Network Visualization*

Description

The main entry point for cograph. Accepts adjacency matrices, edge lists, igraph, statnet network, qgraph, or tna objects and creates a visualization-ready network object.

Usage

```
cograph(
  input,
  layout = NULL,
  directed = NULL,
  nodes = NULL,
  seed = 42,
  simplify = FALSE,
  ...
)
```

Arguments

input	Network input. Can be: • A square numeric matrix (adjacency/weight matrix) • A data frame with edge list (from, to, optional weight columns) • An igraph object • A statnet network object • A qgraph object • A tna object
layout	Layout algorithm name such as "circle", "oval", "spring", "groups", "grid", "random", "star", "bipartite" or "gephi" (see list_layouts); a coordinate matrix or data frame; or an igraph layout function, name or two-letter code. Default NULL computes no layout. A layout can also be set later with sn_layout.
directed	Logical. Force directed interpretation. NULL for auto-detect.
nodes	Node metadata. Can be NULL or a data frame with node attributes. If data frame has a label or labels column, those are used for display.

seed	Random seed for deterministic layouts. Default 42. Set NULL for random.
simplify	Logical or character. Used for tna input only. If FALSE (default), every transition from tna sequence data is a separate edge. If TRUE or a string ("sum", "mean", "max", "min"), duplicate edges are aggregated, and TRUE uses "sum".
...	Additional arguments passed to the layout function.

Value

A cograph_network object. It is a list with the elements nodes, edges, directed, weights, data, meta and node_groups.

See Also

splot for base R graphics rendering, soplot for grid graphics rendering, sn_nodes for node customization, sn_edges for edge customization, sn_layout for changing layouts, sn_theme for visual themes, sn_palette for color palettes, from_qgraph and from_tna for converting external objects

Examples

```
cograph(regulation_net) |> splot(layout = "circle")
```

CographLayout *CographLayout R6 Class*

Description

Class for managing layout algorithms and computing node positions.

Value

A CographLayout R6 object.

Methods

Public methods:

- CographLayout$new()
- CographLayout$compute()
- CographLayout$normalize_coords()
- CographLayout$get_type()
- CographLayout$get_params()
- CographLayout$print()
- CographLayout$clone()

Method new(): Create a new CographLayout object.

Usage:

```
CographLayout$new(type = "circle", ...)
```

Arguments:

`type` Layout name. One of the names returned by `list_layouts()`, or `"custom"` together with a `coords` argument.

`...` Additional parameters stored and passed to the layout function.

Returns: A new CographLayout object.

Method `compute()`**:** Compute layout coordinates for a network.

Usage:

```
CographLayout$compute(network, ...)
```

Arguments:

`network` A CographNetwork or cograph_network object.

`...` Additional parameters passed to the layout function. They override parameters given to `$new()`.

Returns: A data frame with columns `x` and `y`, one row per node, rescaled by `$normalize_coords()`.

Method `normalize_coords()`**:** Rescale coordinates into the unit square. Both axes are scaled by the same factor, so the larger spread spans `[padding, 1 - padding]` and the layout is centered at 0.5.

Usage:

```
CographLayout$normalize_coords(coords, padding = 0.1)
```

Arguments:

`coords` Matrix or data frame. Columns `x` and `y` are used, or the first two columns when these names are absent.

`padding` Numeric. Margin left on each side of the larger spread.

Returns: A data frame with rescaled `x` and `y` columns.

Method `get_type()`**:** Get layout type.

Usage:

```
CographLayout$get_type()
```

Returns: A character string.

Method `get_params()`**:** Get layout parameters.

Usage:

```
CographLayout$get_params()
```

Returns: A list of the parameters given to `$new()`.

Method `print()`**:** Print layout summary.

Usage:

```
CographLayout$print()
```

Returns: The object itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CographLayout$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
layout <- CographLayout$new("circle")
layout$compute(CographNetwork$new(regulation_net))
```

`CographNetwork` *CographNetwork R6 Class*

Description

Core class representing a network for visualization. Stores nodes, edges, layout coordinates, and aesthetic mappings.

Value

A `CographNetwork` R6 object.

Active bindings

`n_nodes` Number of nodes in the network.

`n_edges` Number of edges in the network.

`is_directed` Whether the network is directed.

`has_weights` `TRUE` when any edge weight differs from 1.

`node_labels` Vector of node labels, taken from the `labels` column of the node table when present and from `label` otherwise.

Methods

Public methods:

- `CographNetwork$new()`
- `CographNetwork$clone_network()`
- `CographNetwork$set_nodes()`
- `CographNetwork$set_edges()`
- `CographNetwork$set_directed()`
- `CographNetwork$set_weights()`
- `CographNetwork$set_layout_coords()`
- `CographNetwork$set_node_aes()`
- `CographNetwork$set_edge_aes()`
- `CographNetwork$set_theme()`
- `CographNetwork$get_nodes()`
- `CographNetwork$get_edges()`
- `CographNetwork$get_layout()`
- `CographNetwork$get_node_aes()`
- `CographNetwork$get_edge_aes()`

- CographNetwork$get_theme()
- CographNetwork$set_layout_info()
- CographNetwork$get_layout_info()
- CographNetwork$set_plot_params()
- CographNetwork$get_plot_params()
- CographNetwork$print()
- CographNetwork$clone()

Method `new()`**:** Create a new CographNetwork object.

Usage:

```
CographNetwork$new(
  input = NULL,
  directed = NULL,
  nodes = NULL,
  simplify = FALSE
)
```

Arguments:

`input` Network input, such as a matrix, edge list, igraph, statnet network, qgraph or tna object. `NULL` creates an empty object.

`directed` Logical. Forces a directed or undirected interpretation. `NULL` detects it from the input.

`nodes` `NULL` or a data frame of node attributes. Its rows are matched to the node labels by a `name`, `label` or `id` column, tried in that order, and the remaining columns are added to the node table. When no column matches and the row count equals the number of nodes, the columns are added in row order. A `labels` column supplies the display labels returned by `$node_labels`.

`simplify` Logical or character. If FALSE (default), every transition from tna sequence data is a separate edge. If TRUE or a string ("sum", "mean", "max", "min"), duplicate transitions are aggregated. Other inputs are not affected.

Returns: A new CographNetwork object.

Method `clone_network()`**:** Create a copy with the same nodes, edges, weights, layout, aesthetics, theme, layout information and plot parameters.

Usage:

```
CographNetwork$clone_network()
```

Returns: A new CographNetwork object.

Method `set_nodes()`**:** Set nodes data frame.

Usage:

```
CographNetwork$set_nodes(nodes)
```

Arguments:

`nodes` Data frame with node information.

Returns: The object itself, invisibly.

Method `set_edges()`**:** Set edges data frame.

Usage:

```
CographNetwork$set_edges(edges)
```

Arguments:

`edges` Data frame with edge information.

Returns: The object itself, invisibly.

Method `set_directed()`**:** Set directed flag.

Usage:

```
CographNetwork$set_directed(directed)
```

Arguments:

`directed` Logical.

Returns: The object itself, invisibly.

Method `set_weights()`**:** Set edge weights.

Usage:

```
CographNetwork$set_weights(weights)
```

Arguments:

`weights` Numeric vector of edge weights, one per edge.

Returns: The object itself, invisibly.

Method `set_layout_coords()`**:** Set layout coordinates.

Usage:

```
CographNetwork$set_layout_coords(coords)
```

Arguments:

`coords` Matrix or data frame with at least two columns and one row per node. The first two columns are renamed `x` and `y` and are also written to the node table. `NULL` leaves the layout unchanged.

Returns: The object itself, invisibly.

Method `set_node_aes()`**:** Set node aesthetics. The list is merged into the current node aesthetics.

Usage:

```
CographNetwork$set_node_aes(aes)
```

Arguments:

`aes` Named list of aesthetic parameters.

Returns: The object itself, invisibly.

Method `set_edge_aes()`**:** Set edge aesthetics. The list is merged into the current edge aesthetics.

Usage:

```
CographNetwork$set_edge_aes(aes)
```

Arguments:

`aes` Named list of aesthetic parameters.

Returns: The object itself, invisibly.

Method `set_theme()`**:** Set theme.

Usage:

```
CographNetwork$set_theme(theme)
```

Arguments:

`theme` CographTheme object or theme name.

Returns: The object itself, invisibly.

Method `get_nodes()`**:** Get nodes data frame.

Usage:

```
CographNetwork$get_nodes()
```

Returns: Data frame with node information.

Method `get_edges()`**:** Get edges data frame.

Usage:

```
CographNetwork$get_edges()
```

Returns: Data frame with edge information.

Method `get_layout()`**:** Get layout coordinates.

Usage:

```
CographNetwork$get_layout()
```

Returns: A data frame with `x` and `y` columns, or `NULL` when no layout is set.

Method `get_node_aes()`**:** Get node aesthetics.

Usage:

```
CographNetwork$get_node_aes()
```

Returns: List of node aesthetic parameters.

Method `get_edge_aes()`**:** Get edge aesthetics.

Usage:

```
CographNetwork$get_edge_aes()
```

Returns: List of edge aesthetic parameters.

Method `get_theme()`**:** Get theme.

Usage:

```
CographNetwork$get_theme()
```

Returns: The stored theme (a CographTheme object or theme name), or `NULL`.

Method `set_layout_info()`**:** Set layout info.

Usage:

```
CographNetwork$set_layout_info(info)
```

Arguments:

`info` List with layout information (name, seed, etc.).

Returns: The object itself, invisibly.

Method `get_layout_info()`**:** Get layout info.

Usage:

```
CographNetwork$get_layout_info()
```

Returns: List with layout information.

Method `set_plot_params()`**:** Set plot parameters.

Usage:

```
CographNetwork$set_plot_params(params)
```

Arguments:

`params` List of all plot parameters used.

Returns: The object itself, invisibly.

Method `get_plot_params()`**:** Get plot parameters.

Usage:

```
CographNetwork$get_plot_params()
```

Returns: List of plot parameters.

Method `print()`**:** Print network summary.

Usage:

```
CographNetwork$print()
```

Returns: The object itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CographNetwork$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
CographNetwork$new(regulation_net)
```

CographTheme *CographTheme R6 Class*

Description

Class for managing visual themes for network plots.

Value

A CographTheme R6 object.

Active bindings

`name` Theme name.

Methods

Public methods:

- CographTheme$new()
- CographTheme$get()
- CographTheme$set()
- CographTheme$get_all()
- CographTheme$merge()
- CographTheme$clone_theme()
- CographTheme$print()
- CographTheme$clone()

Method `new()`**:** Create a new CographTheme object.

Usage:

```
CographTheme$new(
  name = "custom",
  background = "white",
  node_fill = "#4A90D9",
  node_border = "#2C5AA0",
  node_border_width = 1,
  edge_color = "gray50",
  edge_positive_color = "#2E7D32",
  edge_negative_color = "#C62828",
  edge_width = 1,
  label_color = "black",
  label_size = 10,
  title_color = "black",
  title_size = 14,
  legend_background = "white"
)
```

Arguments:

`name` Theme name (optional).

`background` Background color.

`node_fill` Default node fill color.

`node_border` Default node border color.

`node_border_width` Default node border width.

`edge_color` Default edge color.

`edge_positive_color` Color for positive edge weights.

`edge_negative_color` Color for negative edge weights.

`edge_width` Default edge width.

`label_color` Default label color.

`label_size` Default label size.

`title_color` Title color.

`title_size` Title size.

`legend_background` Legend background color.

Returns: A new CographTheme object.

Method `get()`**:** Get a theme parameter.

Usage:

```
CographTheme$get(name)
```

Arguments:

`name` Parameter name.

Returns: Parameter value, or `NULL` if the parameter is not set.

Method `set()`**:** Set a theme parameter.

Usage:

```
CographTheme$set(name, value)
```

Arguments:

`name` Parameter name.

`value` Parameter value.

Returns: The object itself, invisibly.

Method `get_all()`**:** Get all theme parameters.

Usage:

```
CographTheme$get_all()
```

Returns: List of parameters.

Method `merge()`**:** Merge with another theme. Parameters in `other` replace the matching parameters of this theme; parameters present in only one of the two, including ones added with `$set()`, are kept.

Usage:

```
CographTheme$merge(other)
```

Arguments:

`other` Another CographTheme or list of parameters.

Returns: A new CographTheme named `"merged"`.

Method `clone_theme()`**:** Clone the theme. Parameters added with `$set()` are copied as well.

Usage:

```
CographTheme$clone_theme()
```

Returns: A new CographTheme.

Method `print()`**:** Print theme summary.

Usage:

```
CographTheme$print()
```

Returns: The object itself, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CographTheme$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
CographTheme$new(background = "white", node_fill = "steelblue")
```

`color_communities` *Color Nodes by Community*

Description

Generate colors for nodes based on community membership. Designed for direct use with `splot()` `node_fill` parameter.

Usage

```
color_communities(x, method = "louvain", palette = NULL, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`method`	Community detection algorithm. See `detect_communities` for available methods. Default `"louvain"`.
`palette`	Color palette to use. Can be: • `NULL` (default): Uses a colorblind-friendly palette • A character vector of colors

- A function that takes n and returns n colors
- A palette name: "rainbow", "colorblind", "pastel", "viridis"

Any other single string is used as one color for every community.

`...` Additional arguments passed to `detect_communities`.

Value

A character vector of colors with one element per node, named by node, for use as the `node_fill` argument of `splot()`.

See Also

`detect_communities`, `splot`

Examples

```
color_communities(regulation_net, method = "walktrap")
```

`communities` *Community Detection*

Description

Detects communities in a network with one of the community detection algorithms of igraph. Each method calls the matching `community_*()` function.

Usage

```
communities(
  x,
  method = c("louvain", "leiden", "fast_greedy", "walktrap", "infomap",
    "label_propagation", "edge_betweenness", "leading_eigenvector", "spinglass",
    "optimal", "fluid"),
  community = NULL,
  weights = NULL,
  resolution = 1,
  directed = NULL,
  seed = NULL,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, CographNetwork, cograph_network, or tna object.
method	Community detection algorithm. One of "louvain" (default; Louvain modularity optimization), "leiden" (Leiden algorithm), "fast_greedy" (greedy modularity optimization), "walktrap" (random walks), "infomap" (map equation), "label_propagation" (label propagation), "edge_betweenness" (Girvan-Newman), "leading_eigenvector" (leading eigenvector of the modularity matrix), "spinglass" (spinglass model), "optimal" (exact modularity maximization) or "fluid" (fluid communities).
community	Optional integer or character vector. If supplied, the returned data frame is filtered to rows whose community column matches one of the given values. Default NULL (keep all communities).
weights	Edge weights. NULL (default) uses the edge weights of the network when present and otherwise runs unweighted. NA runs unweighted.
resolution	Resolution parameter of the louvain and leiden methods. For louvain, higher values yield more communities. Default 1.
directed	Logical. Whether the edge betweenness method treats the network as directed. NULL (default) uses the direction of the network. The other methods ignore it.
seed	Random seed for reproducibility. It applies to the stochastic methods (louvain, leiden, infomap, label_propagation, spinglass).
...	Additional arguments passed to the community_*() function of the chosen method, for example no.of.communities for "fluid".

Details

The louvain, leiden, fast_greedy, leading_eigenvector and fluid methods require an undirected graph. For a directed input this function prints a message and runs "walktrap" instead. Called directly, community_louvain() and community_leiden() raise an igraph error on a directed graph, while community_fast_greedy(), community_leading_eigenvector() and community_fluid() collapse it to an undirected graph with summed weights.

Negative edge weights are replaced by their absolute values for all methods except spinglass and optimal, which receive the weights as they are.

Method	Typical use
louvain	Large undirected networks
leiden	Large undirected networks, well-connected communities
fast_greedy	Medium-sized networks, hierarchical merges
walktrap	Directed or undirected networks, hierarchical merges
infomap	Directed networks with flow structure
label_propagation	Very large networks
edge_betweenness	Small networks, hierarchical splits
leading_eigenvector	Undirected networks, hierarchical splits
spinglass	Small connected networks, negative weights
optimal	Networks of at most about 50 nodes

fluid	Connected networks with a known number of communities

Value

A `cograph_communities` data frame with one row per node and the columns

node Node label (character).

community Community number (numeric).

The attributes `"algorithm"` (method name), `"modularity"` (modularity of the partition, `NA` when igraph does not compute it), `"network"` (the input `x`) and `"igraph_result"` (the igraph `communities` object) hold the metadata. When `community` is supplied, only the matching rows are kept.

Printing and plotting

Printing the result shows the algorithm, the number of nodes and communities, the modularity, the community sizes and the node table. The result is itself a data frame. `plot()` on the result is documented in `plot-results`.

See Also

`community_louvain`, `community_leiden`, `community_fast_greedy`, `community_walktrap`, `community_infomap`, `community_label_propagation`, `community_edge_betweenness`, `community_leading_eigenvector`, `community_spinglass`, `community_optimal`, `community_fluid`

Examples

```
communities(regulation_net, method = "walktrap")
```

`community_consensus` *Consensus Community Detection*

Description

Runs a stochastic community detection algorithm repeatedly and derives consensus communities by thresholding the co-occurrence matrix of the runs.

Usage

```
community_consensus(
  x,
  method = c("louvain", "leiden", "infomap", "label_propagation", "spinglass"),
  n_runs = 100,
  threshold = 0.5,
  seed = NULL,
  ...
```

```
)

com_consensus(
  x,
  method = c("louvain", "leiden", "infomap", "label_propagation", "spinglass"),
  n_runs = 100,
  threshold = 0.5,
  seed = NULL,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object. The louvain and leiden methods require an undirected network.
method	Community detection algorithm, one of "louvain" (default), "leiden", "infomap", "label_propagation" or "spinglass". The current code runs louvain for "spinglass".
n_runs	Number of runs. Default 100.
threshold	Co-occurrence threshold. Default 0.5. Pairs of nodes that share a community in at least this proportion of runs are linked in the consensus graph.
seed	Optional seed for reproducibility. If provided, the RNG state is initialized once before repeated runs and restored on exit.
...	Ignored. Each run calls the igraph function with its own defaults and without weights or resolution settings. For leiden this is the CPM objective with resolution 1.

Details

The algorithm is run n_runs times on the current random number stream. The proportion of runs in which each pair of nodes shares a community forms the co-occurrence matrix. Pairs with a proportion of at least threshold are linked in an unweighted consensus graph, and walktrap on that graph gives the final communities.

Value

A cograph_communities data frame (columns node and community) holding the consensus membership. Its "algorithm" attribute is "consensus_<method>" and its "modularity" attribute is the modularity of the final walktrap partition computed on the consensus graph.

References

Lancichinetti, A., & Fortunato, S. (2012). Consensus clustering in complex networks. *Scientific Reports*, 2, 336.

See Also

communities, community_louvain

Examples

```
community_consensus(to_undirected(regulation_net), method = "louvain",
                    n_runs = 10, seed = 1)
```

`community_edge_betweenness`

Edge Betweenness Community Detection

Description

The Girvan-Newman algorithm. It repeatedly removes the edge with the highest edge betweenness and keeps the partition with the highest modularity. On a weighted graph igraph warns that the membership is selected by modularity.

Usage

```
community_edge_betweenness(
  x,
  weights = NULL,
  directed = TRUE,
  edge.betweenness = TRUE,
  merges = TRUE,
  bridges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)

com_eb(
  x,
  weights = NULL,
  directed = TRUE,
  edge.betweenness = TRUE,
  merges = TRUE,
  bridges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)
```

Arguments

`x`	Network input.
`weights`	Edge weights. `NULL` uses the network weights and `NA` runs unweighted. Negative weights are replaced by their absolute values.

directed	Logical. Whether edge directions are used. Default TRUE. NULL uses the direction of the network.
edge.betweenness	Logical. Whether igraph stores the edge betweenness values. Default TRUE.
merges	Logical. Whether igraph stores the merge matrix. Default TRUE.
bridges	Logical. Whether igraph stores the bridge edges. Default TRUE.
modularity	Logical. Whether igraph stores the modularity scores. Default TRUE.
membership	Logical. Whether igraph computes the membership vector. Default TRUE.
...	Passed to to_igraph. Its only other argument, directed, is already taken by this function, so any further argument raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Girvan, M., & Newman, M.E.J. (2002). Community structure in social and biological networks. *PNAS*, 99(12), 7821-7826.

Examples

```
community_edge_betweenness(igraph::make_graph("Zachary"))
```

---

community_fast_greedy *Fast Greedy Community Detection*

Description

Hierarchical agglomeration by greedy modularity optimization, which produces a dendrogram of community merges. A directed graph is collapsed to an undirected graph with summed edge weights.

Usage

```
community_fast_greedy(
  x,
  weights = NULL,
  merges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)
```

```
com_fg(
  x,
  weights = NULL,
  merges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
merges	Logical. Whether igraph stores the merge matrix. Default TRUE.
modularity	Logical. Whether igraph stores the modularity scores. Default TRUE.
membership	Logical. Whether igraph computes the membership vector. Default TRUE.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. The igraph communities result, including the merge dendrogram when merges = TRUE, is kept in the "igraph_result" attribute.

References

Clauset, A., Newman, M.E.J., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70, 066111.

Examples

```
community_fast_greedy(regulation_net)
```

---

community_fluid *Fluid Communities Detection*

Description

Fluid communities algorithm, in which a fixed number of communities expand and compete for nodes. A directed graph is collapsed to an undirected graph. A disconnected graph raises a warning and only its largest component is partitioned, so the result has one row per node of that component. Edge weights are not used.

Usage

```
community_fluid(x, no.of.communities, ...)

com_fl(x, no.of.communities, ...)
```

Arguments

x	Network input.
no.of.communities	Number of communities to detect. Required; a missing value raises an error.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Pares, F., Gasulla, D.G., Vilalta, A., Moreno, J., Ayguade, E., Labarta, J., Cortes, U., & Suzumura, T. (2018). Fluid communities: A competitive, scalable and diverse community detection algorithm. *Studies in Computational Intelligence*, 689, 229-240.

Examples

```
community_fluid(regulation_net, no.of.communities = 2)
```

---

community_infomap *Infomap Community Detection*

Description

Information-theoretic community detection based on random walks. The partition minimizes the map equation, the description length of a random walk on the network.

Usage

```
community_infomap(
  x,
  weights = NULL,
  v.weights = NULL,
  nb.trials = 10,
  modularity = TRUE,
  seed = NULL,
  ...
```

```
)

com_im(
  x,
  weights = NULL,
  v.weights = NULL,
  nb.trials = 10,
  modularity = TRUE,
  seed = NULL,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
v.weights	Vertex weights (teleportation weights).
nb.trials	Number of optimization trials. Default 10.
modularity	Logical. Whether modularity is computed. Default TRUE.
seed	Random seed for reproducibility. Default NULL.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Rosvall, M., & Bergstrom, C.T. (2008). Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4), 1118-1123.

Examples

```
community_infomap(regulation_net, nb.trials = 10, seed = 1)
```

community_label_propagation
Label Propagation Community Detection

Description

Label propagation community detection. Each node repeatedly adopts the most frequent label among its neighbors.

Usage

```
community_label_propagation(
  x,
  weights = NULL,
  mode = c("out", "in", "all"),
  initial = NULL,
  fixed = NULL,
  seed = NULL,
  ...
)

com_lp(
  x,
  weights = NULL,
  mode = c("out", "in", "all"),
  initial = NULL,
  fixed = NULL,
  seed = NULL,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
mode	Direction of label propagation in directed graphs, one of "out" (default), "in" or "all".
initial	Initial labels, an integer vector, or NULL for a unique label per node.
fixed	Logical vector marking the nodes whose labels are fixed.
seed	Random seed for reproducibility. Default NULL.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Raghavan, U.N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76, 036106.

Examples

```
community_label_propagation(regulation_net, seed = 1)
```

---

community_leading_eigenvector

Leading Eigenvector Community Detection

Description

Detects communities with the leading eigenvector of the modularity matrix, splitting the network divisively. A directed graph is collapsed to an undirected graph with summed edge weights.

Usage

```
community_leading_eigenvector(
  x,
  weights = NULL,
  steps = -1,
  start = NULL,
  options = igraph::arpack_defaults(),
  callback = NULL,
  extra = NULL,
  env = parent.frame(),
  ...
)

com_le(
  x,
  weights = NULL,
  steps = -1,
  start = NULL,
  options = igraph::arpack_defaults(),
  callback = NULL,
  extra = NULL,
  env = parent.frame(),
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
steps	Maximum number of split attempts. Default -1 (no limit).
start	Starting community structure (membership vector).
options	ARPACK options list. Default igraph::arpack_defaults().
callback	Optional function called after each split.
extra	Extra argument passed to callback.
env	Environment in which callback is evaluated.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Newman, M.E.J. (2006). Finding community structure using the eigenvectors of matrices. *Physical Review E*, 74, 036104.

Examples

```
community_leading_eigenvector(regulation_net)
```

---

community_leiden *Leiden Community Detection*

Description

The Leiden algorithm, a refinement of the Louvain algorithm that guarantees well-connected communities. It optimizes the Constant Potts Model (CPM) or modularity. The graph must be undirected; a directed graph raises an igraph error.

Usage

```
community_leiden(
  x,
  weights = NULL,
  resolution = 1,
  objective_function = c("CPM", "modularity"),
  beta = 0.01,
```

```
  initial_membership = NULL,
  n_iterations = 2,
  vertex_weights = NULL,
  seed = NULL,
  ...
)

com_ld(
  x,
  weights = NULL,
  resolution = 1,
  objective_function = c("CPM", "modularity"),
  beta = 0.01,
  initial_membership = NULL,
  n_iterations = 2,
  vertex_weights = NULL,
  seed = NULL,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
resolution	Resolution parameter. Default 1. With the default CPM objective and resolution 1, a network with weights below 1 is typically split into single-node communities.
objective_function	Optimization objective, "CPM" (default) or "modularity".
beta	Randomness parameter of the refinement step. Default 0.01.
initial_membership	Initial community assignments (optional).
n_iterations	Number of iterations. Default 2. A negative value iterates until the partition no longer changes.
vertex_weights	Vertex weights for the CPM objective.
seed	Random seed for reproducibility. Default NULL.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Traag, V.A., Waltman, L., & van Eck, N.J. (2019). From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9, 5233.

Examples

```
community_leiden(to_undirected(regulation_net), objective_function = "modularity",
                 seed = 1)
```

---

community_louvain *Louvain Community Detection*

Description

Multi-level modularity optimization with the Louvain algorithm. The graph must be undirected; a directed graph raises an igraph error, so a directed network is converted first, for example with `to_undirected()`.

Usage

```
community_louvain(x, weights = NULL, resolution = 1, seed = NULL, ...)

com_lv(x, weights = NULL, resolution = 1, seed = NULL, ...)
```

Arguments

`x`	Network input.
`weights`	Edge weights. `NULL` uses the network weights and `NA` runs unweighted. Negative weights are replaced by their absolute values.
`resolution`	Resolution parameter. Higher values yield more communities. Default 1 (standard modularity).
`seed`	Random seed for reproducibility. Default NULL.
`...`	Passed to `to_igraph`, whose only other argument is `directed`; anything else raises an "unused argument" error.

Value

A `cograph_communities` data frame with columns `node` and `community`. See `communities` for its attributes.

References

Blondel, V.D., Guillaume, J.L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, P10008.

Examples

```
community_louvain(to_undirected(regulation_net), seed = 1)
```

`community_optimal` *Optimal Community Detection*

Description

Finds the partition with maximum modularity by exact optimization. Exact modularity maximization is NP-hard, so the computation is feasible only for small networks. A network with more than 50 nodes raises a warning.

Usage

```
community_optimal(x, weights = NULL, ...)

com_op(x, weights = NULL, ...)
```

Arguments

`x`	Network input.
`weights`	Edge weights. `NULL` uses the network weights and `NA` runs unweighted. Weights are passed unchanged.
`...`	Passed to `to_igraph`, whose only other argument is `directed`; anything else raises an "unused argument" error.

Value

A `cograph_communities` data frame with columns `node` and `community`. See `communities` for its attributes.

References

Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2008). On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2), 172-188.

Examples

```
community_optimal(regulation_net)
```

community_sizes *Get Community Sizes*

Description

Get Community Sizes

Usage

```
community_sizes(x)
```

Arguments

x	A cograph_communities object.

Value

An unnamed integer vector of community sizes, ordered by community number.

Examples

```
comm <- community_walktrap(regulation_net)
community_sizes(comm)
```

community_spinglass *Spinglass Community Detection*

Description

Community detection based on the spinglass model of statistical mechanics, optimized by simulated annealing. Negative edge weights are supported with `implementation = "neg"`. A disconnected graph raises a warning and only its largest component is partitioned, so the result has one row per node of that component.

Usage

```
community_spinglass(
  x,
  weights = NULL,
  vertex = NULL,
  spins = 25,
  parupdate = FALSE,
  start.temp = 1,
  stop.temp = 0.01,
  cool.fact = 0.99,
```

```
  update.rule = c("config", "random", "simple"),
  gamma = 1,
  implementation = c("orig", "neg"),
  gamma.minus = 1,
  seed = NULL,
  ...
)

com_sg(
  x,
  weights = NULL,
  vertex = NULL,
  spins = 25,
  parupdate = FALSE,
  start.temp = 1,
  stop.temp = 0.01,
  cool.fact = 0.99,
  update.rule = c("config", "random", "simple"),
  gamma = 1,
  implementation = c("orig", "neg"),
  gamma.minus = 1,
  seed = NULL,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Weights are passed unchanged.
vertex	Vertex whose community is searched (single community mode). NULL (default) partitions the whole network.
spins	Number of spins, the upper limit on the number of communities. Default 25.
parupdate	Logical. Whether spins are updated in parallel. Default FALSE.
start.temp	Starting temperature. Default 1.
stop.temp	Stopping temperature. Default 0.01.
cool.fact	Cooling factor. Default 0.99.
update.rule	Null model of the update rule, one of "config" (default), "random" or "simple".
gamma	Weight of the null model term. Default 1.
implementation	"orig" (default) or "neg", the implementation that supports negative weights.
gamma.minus	Weight of the null model term for negative edges in the "neg" implementation. Default 1.
seed	Random seed for reproducibility. Default NULL.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Reichardt, J., & Bornholdt, S. (2006). Statistical mechanics of community detection. *Physical Review E*, 74, 016110.

Examples

```
community_spinglass(regulation_net, seed = 1)
```

---

community_walktrap *Walktrap Community Detection*

Description

Detects communities from short random walks. Nodes in the same community have short random walk distances.

Usage

```
community_walktrap(
  x,
  weights = NULL,
  steps = 4,
  merges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)

com_wt(
  x,
  weights = NULL,
  steps = 4,
  merges = TRUE,
  modularity = TRUE,
  membership = TRUE,
  ...
)
```

Arguments

x	Network input.
weights	Edge weights. NULL uses the network weights and NA runs unweighted. Negative weights are replaced by their absolute values.
steps	Length of the random walks. Default 4.
merges	Logical. Whether igraph stores the merge matrix. Default TRUE.
modularity	Logical. Whether igraph stores the modularity scores. Default TRUE.
membership	Logical. Whether igraph computes the membership vector. Default TRUE.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities data frame with columns node and community. See communities for its attributes.

References

Pons, P., & Latapy, M. (2006). Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2), 191-218.

Examples

```
community_walktrap(regulation_net, steps = 4)
```

---

compare_communities *Compare Community Structures*

Description

Compares two partitions of the same nodes with igraph's partition comparison measures.

Usage

```
compare_communities(
  comm1,
  comm2,
  method = c("vi", "nmi", "split.join", "rand", "adjusted.rand")
)
```

Arguments

comm1, comm2	Partitions to compare. Each is a cograph_communities data frame, an igraph communities object or a membership vector.
method	Comparison measure, one of "vi" (default; variation of information), "nmi" (normalized mutual information), "split.join" (split-join distance), "rand" (Rand index) or "adjusted.rand" (adjusted Rand index).

Value

A single numeric value. "vi" and "split.join" are distances (0 for identical partitions); "nmi", "rand" and "adjusted.rand" are similarities (1 for identical partitions).

Examples

```
walktrap <- community_walktrap(regulation_net)
fast_greedy <- community_fast_greedy(regulation_net)
compare_communities(walktrap, fast_greedy, method = "nmi")
```

---

complement_network *Complement of a Network*

Description

Every pair of distinct nodes that is not joined in x is joined in the complement, and every joined pair is absent from it.

Usage

```
complement_network(
  x,
  weight = 1,
  loops = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
weight	Numeric. Weight of every edge in the complement. Default 1. A weight of zero means no edge, so weight = 0 raises a cograph_bad_selection error.
loops	Logical. Include self-loops in the complement. Default FALSE.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network holding the complement, or the input format when keep_format = TRUE. Directedness is preserved.

See Also

to_undirected, binarize

Examples

```
complement_network(threshold_edges(regulation_net, minimum = 0.2))
```

`contract_nodes` *Contract Nodes into Groups*

Description

Replaces each group of nodes with a single node whose edges aggregate the edges of its members. The operation corresponds to `igraph::contract()` and tidygraph's `to_contracted()`, and it is the network form of the aggregation computed by `summarize_clusters()`.

Usage

```
contract_nodes(
  x,
  groups,
  weight = c("sum", "mean", "max", "min"),
  loops = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`groups`	Group assignment. Either a vector with one entry per node, in node order, or a named list mapping each group name to node labels. A list must assign every node. Malformed input raises a `cograph_bad_selection` error.
`weight`	How to aggregate the weights of the edges that fall between two groups. One of `"sum"` (default), `"mean"`, `"max"` or `"min"`. Within-group edges are aggregated the same way when `loops = TRUE`.
`loops`	Logical. Keep the within-group edges as self-loops on the contracted node. Default FALSE.
`keep_format`	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a `cograph_network` with a `cograph_no_format_roundtrip` warning. Default FALSE returns a `cograph_network`.
`directed`	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network with one node per group, labeled by group name, or the input format when keep_format = TRUE. The groups follow the factor levels of a vector (sorted for a character vector) or the order of a list.

See Also

summarize_clusters, detect_communities, split_components

Examples

```
contract_nodes(regulation_net, groups = rep(c("Plan", "Act"), each = 5))
```

---

core_periphery *Detect Core-Periphery Structure*

Description

Identifies core-periphery structure in a network with a continuous (Borgatti-Everett) or a discrete method. Core nodes are densely interconnected, and periphery nodes connect mainly to the core. Edge weights are ignored; the analysis uses the binary adjacency matrix.

Usage

```
core_periphery(
  x,
  method = c("continuous", "discrete"),
  directed = NULL,
  iter = 100,
  digits = NULL,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
method	Character string; either "continuous" (default, Borgatti-Everett model) or "discrete" (binary core/periphery assignment).
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
iter	Integer; maximum number of power iterations. Must be at least 1. Default 100.
digits	Integer or NULL. Number of decimal places for the coreness scores, the fitness and the densities. Default NULL (no rounding).
...	Passed to to_igraph, which accepts no further arguments. Any argument supplied here raises an error.

Details

Continuous method:

The Borgatti-Everett model compares the adjacency matrix with the rank-1 pattern matrix `outer(c, c)` of a coreness vector `c`. Here `c` is estimated from the dominant eigenvector of the adjacency matrix, refined by power iteration with rescaling to $[0, 1]$. The iteration stops when the largest change falls below 10^{-6} or after `iter` steps. The vector is not optimized for the correlation. The `"fitness"` attribute is the correlation between the off-diagonal entries of the adjacency matrix and those of the pattern matrix (the lower triangle for a symmetric matrix), and it is 0 when the correlation is undefined.

Discrete method:

The discrete method starts from the continuous solution split at the median and repeatedly flips the single node assignment that most increases `density(core) - density(periphery)`. It stops when no flip increases this quantity. The `"fitness"` attribute is the correlation between the adjacency matrix and the ideal block pattern of the final assignment.

Value

A data frame with class `"cograph_core_periphery"`, one row per node, and columns:

node Node label.

role Character: `"core"` or `"periphery"`. For the continuous method a node is core when its coreness is at or above the median coreness.

coreness Numeric continuous coreness score, rescaled to $[0, 1]$. Reported for both methods.

The attributes `"fitness"`, `"core_density"` and `"periphery_density"` hold the fit and the block densities, and `"network"` holds the original input.

Printing and plotting

Printing the result shows the core and periphery sizes, the fitness and the two block densities, followed by the node table. The result is a data frame and serves as the tidy table directly. `plot()` on the result is documented in `plot-results`.

References

Borgatti, S.P. & Everett, M.G. (2000). Models of core/periphery structures. *Social Networks*, 21(4), 375-395. doi:10.1016/S03788733(99)000192

See Also

`centrality`, `network_summary`

Examples

```
core_periphery(regulation_net)
```

csum *Cluster Summary Statistics*

Description

Aggregates node-level network weights to cluster-level summaries. The result holds the macro (cluster-to-cluster) network and one network of within-cluster connections per cluster.

Usage

```
csum(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "cooccurrence", "semi_markov", "raw"),
  directed = TRUE,
  compute_within = TRUE
)
```

Arguments

x	Network input. Accepts multiple formats: **matrix** Numeric adjacency/weight matrix. Row and column names are used as node labels. Values represent edge weights (e.g., transition counts, co-occurrence frequencies, or probabilities). **cograph_network** A cograph network object. Its weight matrix is used, and clusters can be auto-detected from node attributes. **tna** A tna object from the tna package. Its weight matrix is used and its sequence data are kept in `macro$data`. **cluster_summary** Returned unchanged.
clusters	Cluster/group assignments for nodes. Accepts multiple formats: **NULL** (default) Auto-detect from a cograph_network. The first node column named 'clusters', 'cluster', 'groups' or 'group' is used, then a 'cluster', 'group' or 'layer' column of the node groups. An error is raised when none is found, and for any other input `clusters` must be supplied. **vector** Cluster membership for each node, in the same order as the matrix rows/columns. Can be numeric (1, 2, 3) or character ("A", "B"). Cluster names are the unique values, sorted for a numeric vector and in order of appearance for a character vector or factor. Example: `c(1, 1, 2, 2, 3, 3)` assigns first two nodes to cluster 1. **data.frame** A data frame where the first column contains node names and the second column contains group/cluster names. Example: `data.frame(node = c("A", "B", "C"), group = c("G1", "G1", "G2"))`

named list Explicit mapping of cluster names to node labels. List names become cluster names, values are character vectors of node labels that must match matrix row/column names. Example: `list(Alpha = c("A", "B"), Beta = c("C", "D"))`

`method` Aggregation method for combining edge weights within/between clusters. Zero and `NA` weights are dropped before aggregation (see `aggregate_weights`):

"sum" (default) Sum of the edge weights. Suited to count data such as transition frequencies, because it preserves the total flow.

"mean" Mean edge weight. Suited to inputs that are already transition probabilities, because the result does not grow with cluster size.

"median" Median edge weight. Robust to outliers.

"max" Maximum edge weight. Captures strongest connection.

"min" Minimum edge weight. Captures weakest connection.

"density" Sum divided by the number of possible edges ($n_i n_j$ for clusters of sizes n_i and n_j).

"geomean" Geometric mean of positive weights. Useful for multiplicative processes.

`type` Post-processing applied to aggregated weights. Determines the interpretation of the resulting matrices:

"tna" (default) Row-normalize so each row sums to 1, which gives transition probabilities. Rows that sum to zero are left at zero.

"raw" No normalization. The aggregated weights are returned as computed.

"cooccurrence" Symmetrize the matrix as (A + t(A)) / 2.

"semi_markov" Row-normalize, identical to `"tna"`.

`directed` Logical. Default `TRUE`. With `FALSE` the node-level weights are symmetrized as $(A + A^T)/2$ before aggregation, so the direction of a tie is ignored, and `meta$directed` is `FALSE`. With `type = "cooccurrence"` the aggregated weights are symmetrized and `meta$directed` is `FALSE` whatever this value is.

`compute_within` Logical. If `TRUE` (default), compute per-cluster matrices, one $n_i \times n_i$ matrix of internal node-to-node weights per cluster. `FALSE` skips this step when only the macro summary is needed.

Details

The function is the matrix-based entry point to Multi-Cluster Multi-Level (MCML) analysis. `as_tna` converts the result to tna models.

Workflow:

A typical MCML analysis computes the summary, then plots it with `plot_mcml` or converts it to tna models with `as_tna`:

```
cs <- csum(net, clusters = clusters, type = "tna")
plot_mcml(cs)
as_tna(cs)
```

Between-Cluster Matrix Structure:

The macro weight matrix has clusters as both rows and columns. An off-diagonal cell (i, j) holds the aggregated weight from cluster i to cluster j. A diagonal cell (i, i) holds the aggregated weight of the edges inside cluster i. When `type = "tna"`, rows sum to 1 and the diagonal is the probability of staying inside the same cluster.

Choosing method and type:

Input data	Recommended	Reason
Edge counts	method="sum", type="tna"	Preserves total flow, normalizes to probabilities
Transition matrix	method="mean", type="tna"	Avoids cluster size bias
Frequencies	method="sum", type="raw"	Keep raw counts for analysis
Correlation matrix	method="mean", type="raw"	Average correlations

Value

A `cluster_summary` object (S3 class) containing:

macro A tna object representing the macro (cluster-level) network:

- **weights** k x k matrix of cluster-to-cluster weights, where k is the number of clusters. Row i, column j contains the aggregated weight from cluster i to cluster j. The diagonal contains the aggregated within-cluster weight, node self-loops included. Processing depends on `type`.
- **inits** Named numeric vector of length k, the column sums of the aggregated cluster matrix (before `type` processing) divided by their total. It is uniform when all weights are zero.
- **labels** Cluster names.
- **data** Sequence data of a tna input, otherwise `NULL`.

clusters A `group_tna` list with one tna object per cluster, each containing:

- **weights** n_i x n_i matrix for nodes inside that cluster. Shows internal transitions between nodes in the same cluster.
- **inits** Column sums of the within-cluster weights divided by their total.

NULL if `compute_within = FALSE`.

cluster_members Named list mapping cluster names to their member node labels. Example: `list(A = c("n1", "n2"), B = c("n3", "n4", "n5"))`

meta List of metadata:

- **type** The `type` argument used ("tna", "raw", etc.)
- **method** The `method` argument used ("sum", "mean", etc.)
- **directed** Logical, effective directedness of the stored weights (`FALSE` when `type = "cooccurrence"`, which symmetrizes them)
- **n_nodes** Total number of nodes in original network
- **n_clusters** Number of clusters
- **cluster_sizes** Named vector of cluster sizes

See Also

as_tna to convert results to tna objects, plot_mcml for two-layer visualization, plot_mtna for flat cluster visualization

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
csum(regulation_net, clusters = clusters, type = "tna")
```

---

degree_distribution *Degree Distribution Visualization*

Description

Plots a histogram or a complementary cumulative distribution of node degrees. When the degree range is at most 50, the default bins are integer-aligned, with one bar per degree value.

Usage

```
degree_distribution(
  x,
  mode = "all",
  directed = NULL,
  loops = TRUE,
  simplify = "sum",
  cumulative = FALSE,
  breaks = NULL,
  bins = NULL,
  bin_width = NULL,
  normalize = FALSE,
  log = "",
  main = "Degree Distribution",
  xlab = "Degree",
  ylab = NULL,
  col = "steelblue",
  border = "white",
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
mode	For directed networks: "all", "in", or "out". Default "all".
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

loops	Logical. If TRUE (default), keep self-loops. Set FALSE to remove them.
simplify	How to combine multiple edges between the same node pair. Options: "sum" (default), "mean", "max", "min", or FALSE/"none" to keep multiple edges.
cumulative	Logical. If TRUE, show CCDF (complementary cumulative distribution: P(degree >= k)) instead of frequency. Default FALSE.
breaks	Bin specification passed to hist. Can be a numeric vector of breakpoints, a single number giving the number of bins, or a character string naming an algorithm (e.g. "Sturges", "FD", "scott"). Overrides bins and bin_width. Default NULL (auto-detect).
bins	Integer. Number of equal-width bins spanning the degree range. Overrides bin_width. Default NULL.
bin_width	Numeric. Width of each bin. Default NULL, which uses a width of 1 when the degree range is at most 50 and the Freedman-Diaconis rule otherwise.
normalize	Logical. If TRUE, the y-axis shows proportions (bars sum to 1) instead of counts. Default FALSE.
log	Character. Axis log-scaling: "" (none, default), "x", "y", or "xy". Histogram plots apply y-axis log scaling for "y" or "xy"; cumulative plots support x, y, and xy scaling, and "xy" gives a log-log CCDF.
main	Character. Plot title. Default "Degree Distribution".
xlab	Character. X-axis label. Default "Degree".
ylab	Character. Y-axis label. Default auto-chosen based on normalize and cumulative.
col	Character. Bar fill or line color. Default "steelblue".
border	Character. Bar border color. Default "white".
...	Additional graphical arguments passed to barplot (histogram) or plot (cumulative).

Value

Invisibly returns a list with components:

degree Named numeric vector of per-node degrees.

table Table of degree frequencies.

breaks Breakpoints of the degree histogram.

counts Bin counts.

proportions Bin proportions (counts / sum(counts)).

The same five components are returned for the histogram and the cumulative plot. The bin components always describe the histogram bins.

Examples

```
cograph::degree_distribution(regulation_net)
```

`detect_communities` *Detect Communities in a Network*

Description

Detects communities (clusters) in a network using various community detection algorithms. Returns a data frame with node-community assignments.

Usage

```
detect_communities(x, method = "louvain", directed = NULL, weights = TRUE)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`method`	Community detection algorithm to use. One of: • `"louvain"`: Louvain modularity optimization (default) • `"walktrap"`: Walktrap algorithm based on random walks • `"fast_greedy"`: Fast greedy modularity optimization • `"label_prop"`: Label propagation algorithm • `"infomap"`: Infomap algorithm based on information flow • `"leiden"`: Leiden algorithm, run with the defaults of `igraph::cluster_leiden()`. Its default constant Potts model objective can place every node in its own community. `"louvain"`, `"leiden"` and `"fast_greedy"` require an undirected graph. A directed network is collapsed to an undirected one for these methods, with the weights of reciprocal edges averaged, and a message is issued for `"louvain"` and `"leiden"`. The `"louvain"`, `"leiden"`, `"label_prop"` and `"infomap"` methods use random numbers, so `set.seed()` makes their result reproducible.
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
`weights`	Logical. Use edge weights for community detection. Default TRUE.

Value

A `cograph_communities` object, which inherits from `data.frame` and has one row per node with columns:

- `node`: Node labels/names
- `community`: Numeric community membership

The algorithm name, the igraph community object, the modularity and the input network are carried as attributes for the `print`, `plot` and `modularity` methods.

Examples

```
detect_communities(regulation_net, method = "walktrap")
```

`disparity_filter` *Disparity Filter*

Description

Extracts the statistically significant backbone of a weighted network using the disparity filter method (Serrano, Boguna, & Vespignani, 2009).

Usage

```
disparity_filter(x, level = 0.05, ...)

## Default S3 method:
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'matrix'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'tna'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'cograph_network'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'igraph'
disparity_filter(x, level = 0.05, ...)
```

Arguments

`x`	A weight matrix, tna object, cograph_network, or igraph object.
`level`	Significance level (default 0.05). Lower values result in a sparser backbone (fewer edges retained).
`...`	Additional arguments (currently unused).

Details

The disparity filter identifies edges that carry a disproportionate fraction of a node's total weight, based on a null model where weights are distributed uniformly at random.

For each node i with degree k_i, and each edge (i, j) with normalized weight $p_{ij} = w_{ij}/s_i$ (where s_i is the node's strength), the p-value is:

$$p = (1 - p_{ij})^{(k_i - 1)}$$

The p-value is computed from the outgoing weights of the source node (row strength and out-degree) and from the incoming weights of the target node (column strength and in-degree). An edge is significant if the smaller of the two p-values is below `level`. Self-loops are never retained. For igraph input, unweighted edges get weight 1 and multiple edges are summed before filtering.

Value

For a matrix, an integer matrix of the same dimensions and dimnames with 1 for significant edges and 0 otherwise. For tna, cograph_network and igraph objects, a `tna_disparity` list with elements `significant` (the 0/1 matrix), `weights_orig`, `weights_filtered` (original weights times the 0/1 matrix), `level`, `n_edges_orig` and `n_edges_filtered`. Any other input raises an error.

References

Serrano, M. A., Boguna, M., & Vespignani, A. (2009). Extracting the multiscale backbone of complex weighted networks. Proceedings of the National Academy of Sciences, 106(16), 6483-6488.

See Also

`bootstrap` for bootstrap-based significance testing

Examples

```
disparity_filter(regulation_net, level = 0.3)
```

---

dispersion *Dispersion (Backstrom-Kleinberg 2014)*

Description

Computes the dispersion of a tie (Backstrom and Kleinberg 2014), a per-pair measure of tie strength. Edge weights are ignored, and directed networks use out-neighbors. For each pair (u, v) where v is a neighbor of u, the computation proceeds as follows.

Usage

```
dispersion(x, u = NULL, v = NULL, normalized = TRUE, alpha = 1, b = 0, c = 0)
```

Arguments

`x`	Network input (matrix, igraph, network, cograph_network, tna object).
`u`	Optional source node (1-based index or node name). If `NULL` (default), compute for all sources.
`v`	Optional target node. If `NULL`, compute for all neighbors of `u`.
`normalized`	Logical. If `TRUE` (default), return the normalized form; otherwise the raw count.
`alpha`	Numeric normalization exponent. Default 1.
`b`	Numeric bias added to dispersion before exponentiation. Default 0.
`c`	Numeric bias added to embeddedness in the denominator. Default 0.

Details

1. Let $S_T = N(u) \cap N(v)$ be their mutual friends (embeddedness).
2. Count pairs $(s, t) \subset S_T$ such that:
   - s and t are not directly connected, and
   - s and t share no common neighbor inside $N(u)$ other than u and v.
3. The raw dispersion is this count. When `normalized = TRUE`, the result is $(\text{dispersion} + b)^{\alpha}/(\text{embeddedness} + c)$ (normalization is skipped when `embeddedness + c == 0`).

Value

- Scalar if both `u` and `v` are specified.
- Named numeric vector if exactly one of `u`, `v` is given, one element per neighbor of that node; the names are the neighbors' 1-based node indices as character strings.
- A data frame with columns `from`, `to`, `dispersion` when neither `u` nor `v` is given, one row per ordered (node, neighbor) pair, with `from` and `to` given as 1-based integer node indices.
- `numeric(0)` for an empty graph.

References

Backstrom, L., & Kleinberg, J. (2014). Romantic partnerships and the dispersion of social ties: A network analysis of relationship status on Facebook. In *Proceedings of CSCW* (pp. 831-841). ACM. https://arxiv.org/pdf/1310.6753v1.pdf

Examples

```
dispersion(regulation_net, u = "Plan")
```

`dyad_census` *Dyad Census*

Description

Classifies every dyad (unordered pair of nodes) in a directed network into one of three mutually exclusive states: mutual (M, edges in both directions), asymmetric (A, an edge in exactly one direction), or null (N, no edge between the pair). The dyad census is the dyad-level counterpart of `triad_census` and is the basis of dyad-based reciprocity.

Usage

```
dyad_census(x, directed = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
...	Passed to to_igraph, which accepts no further arguments. Any argument supplied here raises an error.

Details

In an undirected network every present edge is counted as a mutual dyad and the asymmetric count is zero. The total number of dyads is $n(n-1)/2$ in both cases.

Value

A tidy data.frame of class "cograph_dyad_census" with one row per dyad type and columns:

type Character: "mutual", "asymmetric", or "null".

count Integer: number of dyads of that type.

proportion Numeric: count divided by the total number of dyads ($n(n-1)/2$).

The dyad-based reciprocity $2M/(2M+A)$ is attached as the "reciprocity" attribute (NA for a network without edges). The attributes "directed" and "n_dyads" record the directedness and the total number of dyads.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

triad_census, edge_reciprocity, network_summary

Examples

```
cograph::dyad_census(regulation_net)
```

edge_centrality *Calculate Edge Centrality Measures*

Description

Computes centrality measures for the edges of a network and returns a tidy data frame with one row per edge.

Usage

```
edge_centrality(
  x,
  measures = "all",
  weighted = TRUE,
  directed = NULL,
  cutoff = -1,
  invert_weights = NULL,
  alpha = 1,
  digits = NULL,
  sort_by = NULL,
  ...
)

edge_betweenness(x, ...)
```

Arguments

x	Network input (matrix, edge-list data frame, igraph, network, cograph_network, tna object).
measures	Which measures to calculate. Default "all" calculates all available edge measures. Options: "betweenness", "weight", "overlap", "simmelian", "reciprocity".
weighted	Logical. Use edge weights if available. Default TRUE.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
cutoff	Maximum path length for betweenness. Default -1 (no limit).
invert_weights	Logical or NULL. Whether edge betweenness inverts the weights, so that higher weights mean shorter paths. The default NULL is TRUE for tna objects and FALSE otherwise.
alpha	Numeric. Exponent of the inversion, which computes distances as 1 / weight^alpha. Default 1.
digits	Integer or NULL. Round numeric columns. Default NULL.
sort_by	Character or NULL. Column to sort by (descending). Default NULL.
...	For edge_centrality(), the graph-construction arguments loops and simplify (see centrality). For edge_betweenness(), arguments passed to edge_centrality().

Details

Edge measures available, with the column(s) each one adds:

betweenness Edge betweenness, the sum over node pairs of the share of their shortest paths that pass through the edge. Adds `betweenness`.

weight Original edge weight (1 for an unweighted input). Adds `weight`.

overlap Jaccard neighborhood overlap of the edge endpoints. Adds `overlap` and the raw count `shared_neighbors`.

simmelian Number of triangles the edge participates in. Adds `triangles` (there is no column called `simmelian`).

reciprocity Whether the reverse edge exists. Directed only: on an undirected input it warns and adds nothing. Adds `reciprocated`, `reverse_weight` and `weight_ratio`, the last two `NA` where the edge is not reciprocated.

`measures = "all"` requests every measure, dropping `reciprocity` on an undirected input.

Value

`edge_centrality()` returns a base `data.frame` with one row per edge, in the canonical (row-major) edge order of the input. The first two columns are `from` and `to` (character when the input carried node names, numeric indices otherwise); the remaining columns are those the requested measures contribute, as listed in Details. `measures = "all"` on an undirected input therefore gives `from`, `to`, `weight`, `betweenness`, `overlap`, `shared_neighbors` and `triangles`, and a directed input adds `reciprocated`, `reverse_weight` and `weight_ratio`.

`edge_betweenness()` returns a numeric vector of edge betweenness values named `"from->to"`, for directed and undirected inputs alike.

Examples

```
edge_centrality(regulation_net, measures = "betweenness")
```

`edge_reciprocity` *Edge Reciprocity*

Description

Convenience wrapper around `edge_centrality` that returns only reciprocity information for directed networks.

Usage

```
edge_reciprocity(x, top = NULL, directed = NULL, digits = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
top	Integer or NULL. Return only the top N edges. Default NULL.
directed	Logical or NULL. Default NULL (auto-detect).
digits	Integer or NULL. Round numeric columns. Default NULL.
...	Additional arguments passed to edge_centrality.

Value

A data frame with one row per directed edge and columns from, to, weight, reciprocated (logical), reverse_weight (NA when not reciprocated) and weight_ratio (reverse_weight / weight; NA when not reciprocated). Rows are ordered with reciprocated edges first, then by |weight_ratio| descending.

Errors

Raises an error when the resolved network is undirected: reciprocity is only defined for directed edges.

See Also

edge_centrality

Examples

```
cograph::edge_reciprocity(regulation_net, top = 5)
```

---

ego_networks *Ego-Network Metrics*

Description

Extracts the ego network of each requested node and computes a table of personal-network metrics with one row per ego. An ego network consists of the node, its neighbors up to a given order and the ties among them. The metrics are the network size, tie counts and densities, and Burt's structural-hole measures.

Usage

```
ego_networks(
  x,
  nodes = NULL,
  order = 1,
  mode = c("all", "out", "in"),
  directed = NULL,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
nodes	Character vector of node names or integer vector of node indices selecting which egos to report. NULL (default) uses every node. An unknown name or an index out of range raises an error.
order	Integer neighborhood order defining the ego network. 1 (default) is the standard ego network (ego and direct neighbors). Burt's effective_size and constraint are only defined for order = 1 and are returned as NA otherwise.
mode	For directed networks, which ties define the neighborhood: "all" (default), "out", or "in". The Burt measures do not depend on mode.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry.
...	Passed to to_igraph, which takes no further arguments, so any argument supplied here raises an error.

Details

Self-loops are dropped before ties are counted. In a directed network the tie counts are counts of directed edges, and the densities divide by $m(m-1)$ for m members.

effective_size and constraint are computed on the full network from all ties of each node, with the same implementations as centrality, so the values equal centrality(x, measures = c("effective_size", "constraint")). Effective size uses the unweighted ties. Constraint uses the edge weights.

Value

A data frame of class "cograph_ego_networks" with one row per ego and columns:

node Ego node name.

size Number of alters (ego-network size, excluding ego).

ego_ties Number of edges in the ego network (ego and alters).

ego_density Edge density of the ego network including ego. NA for an ego without alters.

alter_ties Number of edges among the alters only (excluding ego).

alter_density Edge density among the alters. NA for an ego with fewer than two alters. Low values indicate many structural holes and brokerage opportunities.

effective_size Burt's effective size of the ego network (order = 1 only).

constraint Burt's constraint (order = 1 only).

The arguments order and mode and the directedness of the network are stored as attributes. Printing the result shows order and mode above the table.

References

Burt, R.S. (1992). *Structural Holes: The Social Structure of Competition*. Harvard University Press.

See Also

centrality (for effective_size and constraint), dispersion, select_neighbors, neighborhood_overlap

Examples

```
cograph::ego_networks(regulation_net, nodes = "Plan")
```

---

estrada_index *Estrada Index*

Description

Computes the Estrada index, a graph-level spectral invariant

$$EE(G) = \sum_{i=1}^{n} e^{\lambda_i}$$

where λ_i are the eigenvalues of the binary adjacency matrix. Edge weights are ignored. For an undirected network the index equals $\sum_k M_k/k!$, where M_k is the number of closed walks of length k, and it is the sum of the subgraph centralities of all nodes. For a directed network the eigenvalues may be complex. They come in conjugate pairs, so the sum of their exponentials is real and again equals the trace of e^A, the weighted count of closed walks.

Usage

```
estrada_index(x)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).

Value

Numeric scalar: the Estrada index of the graph, or 0 for a graph with no nodes.

References

Estrada, E. (2000). Characterization of 3D molecular structure. *Chemical Physics Letters*, 319(5-6), 713-718.

See Also

centrality_subgraph for the per-node measure. On an undirected network its values sum to estrada_index(x).

Examples

```
estrada_index(regulation_net)
```

`extract_motifs` *Extract Motifs from Network Data*

Description

Extracts the triads of a network, classifies each one into the 16 MAN types of a directed network, and optionally tests each triad against a permutation null model. The analysis is individual-level for tna objects and grouped data, and aggregate for matrices and networks.

Usage

```
extract_motifs(
  x = NULL,
  data = NULL,
  id = NULL,
  level = NULL,
  edge_method = c("any", "expected", "percent"),
  edge_threshold = 1.5,
  pattern = c("triangle", "network", "closed", "all"),
  exclude_types = NULL,
  include_types = NULL,
  top = NULL,
  by_type = FALSE,
  min_transitions = 5,
  significance = FALSE,
  n_perm = 100,
  seed = NULL
)
```

Arguments

`x`	Input data. Can be: • A `tna` object (supports individual-level analysis) • A matrix (aggregate analysis only, unless `data` and `id` provided) • A `cograph_network` object • An `igraph` object
`data`	Optional data.frame containing transition data with an ID column for individual-level analysis. Required columns: `from`, `to`, and the column(s) specified in `id`. An optional `weight` column gives the transition weights (1 otherwise). `data` is used when `x` is not a tna object, and `x` is then ignored.
`id`	Column name(s) identifying individuals/groups in `data`. Can be a single string or character vector for multiple grouping columns. Required for individual-level analysis with non-tna inputs.
`level`	Analysis level: "individual" counts how many individuals show each triad, and "aggregate" analyzes the network summed over individuals. The default is "individual" for a tna object or when `data` and `id` are supplied, and "aggregate"

otherwise. Requesting "individual" without individual data raises a warning and the aggregate level is used.

`edge_method` Method for determining edge presence:

"any" Edge exists if count > 0.

"expected" Edge exists if observed/expected >= threshold.

"percent" Edge exists if edge/total >= threshold.

Default "any".

`edge_threshold` Threshold value for "expected" or "percent" methods. For "expected", a ratio (1.5 means 50\ For "percent", a proportion of the total triad weight (for example 0.15). The default 1.5 is intended for "expected", so a value for "percent" should be set explicitly. Ignored when edge_method = "any". Default 1.5.

`pattern` Pattern filter for which triads to include:

"triangle" All 3 node pairs must be connected (any direction). Types: 030C, 030T, 120C, 120D, 120U, 210, 300. Default.

"network" Excludes the empty triad and the sequential patterns 003, 012 and 021C. Stars and triangles are kept.

"closed" Excludes 003, 012, 021C and 120C.

"all" All 16 MAN types.

`exclude_types` Character vector of MAN types to explicitly exclude. Applied after pattern filter. E.g., c("300") to exclude cliques.

`include_types` Character vector of MAN types to include. When supplied, only these types are returned, and `pattern` and `exclude_types` are ignored.

`top` Number of rows to return, taken after sorting by observed count (by z-score when `significance = TRUE`). NULL returns all rows. Default NULL.

`by_type` If TRUE, the rows are sorted by MAN type and then by observed count. Default FALSE.

`min_transitions`

At individual level: minimum total transitions for a person to be included in the analysis. At aggregate level: minimum triad weight to count as present. Default 5.

`significance` Logical. Run permutation significance test? Default FALSE.

`n_perm` Number of permutations for the significance test. When `significance = TRUE`, must be a whole number of at least 2. Default 100.

`seed` Optional random seed. The caller's random number state is restored on exit.

Details

Self-loops are removed before the activity filter, the counting and the null model. Significance is assessed against a directed weighted stub-matching null model. Each weight is rounded to an integer number of stubs, and a positive weight that rounds to zero keeps one stub. The target stubs are shuffled, which preserves the in- and out-strengths of every unit. Loops and multiple edges created by the shuffle are collapsed to a simple graph before the triads are classified. The aggregate mode of `motifs()` uses the simple-graph rewiring null of `motif_census()` instead.

The selected edge_method is reapplied to each null replicate. Since small positive weights are rounded up to one stub, the total weight of a replicate can differ from the observed total under "percent" and "expected". Results without significance testing and results with edge_method = "any" do not depend on this rounding.

Value

A cograph_motif_analysis object (list) containing the elements below, or NULL with a warning when no triad passes the filters.

results Data frame with one row per node-triple and MAN type, the display label triad, unambiguous node1/node2/ node3 columns, its observed count, and (if significance = TRUE) expected count, z-score, empirical p-value, and significance marker. A node triple that has different types across individuals therefore appears in more than one row.

type_summary A table of triad counts by MAN type, summed over individuals and sorted in decreasing order.

params List of the settings used, with the number of individuals, the number of states and the node labels.

MAN Notation

The 16 triad types use MAN (Mutual-Asymmetric-Null) notation where:

- First digit: number of mutual (bidirectional) pairs.
- Second digit: number of asymmetric (one-way) pairs.
- Third digit: number of null (no edge) pairs.
- Letter suffix: subtype (C = cycle, T = transitive, D = down, U = up).

Pattern Types

Triangle patterns (all pairs connected): 030C (cycle), 030T (feed-forward), 120C (regulated cycle), 120D (two out-stars), 120U (two in-stars), 210 (mutual+asymmetric), 300 (clique)

Network patterns: 021D (out-star), 021U (in-star), 102 (mutual pair), 111D (out-star+mutual), 111U (in-star+mutual), 201 (two mutual pairs), plus all triangle patterns

Sequential patterns (chains): 012 (single edge), 021C (A->B->C chain)

Empty: 003 (no edges)

Printing and plotting

Printing the result shows the analysis settings, the MAN type distribution and the first 20 triads. plot() on the result is documented in plot-results.

See Also

motifs(), subgraphs(), extract_triads(), motif_census()

Other motifs: extract_triads(), get_edge_list(), motif_census(), motifs(), subgraphs(), triad_census()

Examples

```
extract_motifs(regulation_net, min_transitions = 0)
```

`extract_triads` *Extract Triads with Node Labels*

Description

Lists the triads of a network that contain at least one edge, with the labels of their nodes, so that the node combinations forming each motif pattern can be identified.

Usage

```
extract_triads(
  x,
  type = NULL,
  involving = NULL,
  threshold = 0,
  min_total = 5,
  directed = NULL
)
```

Arguments

`x`	A matrix, igraph object, tna, or cograph_network.
`type`	Character vector of MAN codes to filter by (e.g., "030T", "030C"). Default NULL returns all types.
`involving`	Character vector of node labels. Only return triads involving at least one of these nodes. Default NULL returns all triads.
`threshold`	Edge weight threshold. An edge counts as present for the triad type when its weight is greater than `threshold`. Default 0.
`min_total`	Minimum total weight across all 6 edges. Excludes trivial triads with low overall activity. Default 5.
`directed`	Logical or NULL. Whether the network is treated as directed. NULL (default) detects it from the input.

Details

The function complements `motif_census()` by showing the node combinations that form each motif pattern. The triad type is determined by edge presence (weight greater than `threshold`). The weight columns hold the edge weights themselves, which measure the strength of each triad.

Value

A data frame with one row per triad, in node index order, and the columns:

A, B, C Labels of the three nodes in the triad.

type MAN code (012, ..., 300). Triads of type 003 have no edge and are not returned.

weight_AB, weight_BA, weight_AC, weight_CA, weight_BC, weight_CB Edge weights of the 6 possible directed edges.

total_weight Sum of the 6 edge weights.

A network with fewer than 3 nodes gives a data frame with no rows.

See Also

`motifs()`, `subgraphs()`, `motif_census()`, `extract_motifs()`

Other motifs: `extract_motifs()`, `get_edge_list()`, `motif_census()`, `motifs()`, `subgraphs()`, `triad_census()`

Examples

```
extract_triads(regulation_net, type = "030T", min_total = 0)
```

---

filter_edges *Filter Edges by Metadata*

Description

Filter edges using dplyr-style expressions on any edge column. Returns a cograph_network object by default (universal format), or optionally a matrix, igraph, statnet network or tna object when `keep_format = TRUE` and the input used one of those formats.

Usage

```
filter_edges(
  x,
  ...,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL,
  .keep_isolates = NULL
)

subset_edges(
  x,
  ...,
  keep_isolates = TRUE,
```

```
  keep_format = FALSE,
  directed = NULL,
  .keep_isolates = NULL
)
```

Arguments

x	Network input: cograph_network, matrix, igraph, network, or tna object.
...	Filter expressions using any edge column (e.g., weight > 0.5, weight > mean(weight), abs(weight) > 0.3).
keep_isolates	Logical. Keep nodes that end up with no edges? Default TRUE, matching igraph::delete_edges() and tidygraph: filtering edges does not remove nodes. Set FALSE to drop them, or call remove_isolates() afterwards.
keep_format	Logical. If TRUE, matrix, igraph, statnet network and tna inputs are returned in that format. Default FALSE returns cograph_network (universal format).
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected. Only used for non-cograph_network inputs.
.keep_isolates	Deprecated. Use keep_isolates.

Value

A cograph_network object with filtered edges. If keep_format = TRUE, matrix, igraph, statnet network and tna inputs are converted back to that type. With keep_isolates = TRUE, a node that loses all its edges stays in the network and a cograph_isolates_created warning is raised.

See Also

filter_nodes, splot, subset_edges

Examples

```
filter_edges(regulation_net, weight > 0.15)
```

---

filter_nodes *Filter Nodes by Metadata or Centrality*

Description

Filter nodes using dplyr-style expressions on any node column or centrality measure. Returns a cograph_network object by default (universal format), or optionally a matrix, igraph, statnet network or tna object when keep_format = TRUE and the input used one of those formats.

Usage

```
filter_nodes(
  x,
  ...,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL,
  .keep_edges = NULL
)

subset_nodes(
  x,
  ...,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL,
  .keep_edges = NULL
)
```

Arguments

x	Network input: cograph_network, matrix, igraph, network, or tna object.
...	Filter expressions using any node column or centrality measure. Available variables include: **Node columns** All columns of the node table, such as `id`, `label`, `name`, `x`, `y` and any custom columns. **Centrality measures** `degree`, `indegree`, `outdegree`, `strength`, `instrength`, `outstrength`, `betweenness`, `closeness`, `eigenvector`, `pagerank`, `hub`, `authority`, `coreness`. Any other measure `centrality()` computes can be named too; see `list_centralities()`. **Structural context and predicates** The same vocabulary `select_nodes()` documents, for example `component`, `component_size`, `k_core`, `is_isolated`, `is_cut`, `local_transitivity`. Examples: `degree >= 3`, `label %in% c("A", "B")`, `pagerank > 0.1 & degree >= 2`. On a network with negative edge weights, `betweenness`, `closeness` and `pagerank` are undefined: they return `NA` with a `cograph_negative_weights` warning.
keep_edges	How to handle edges. One of: `"internal"` (default) Keep only edges between remaining nodes `"none"` Remove all edges
keep_format	Logical. If TRUE, matrix, igraph, statnet network and tna inputs are returned in that format. Default FALSE returns cograph_network (universal format).
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected. Only used for non-cograph_network inputs.
.keep_edges	Deprecated. Use `keep_edges`.

Value

A cograph_network object with filtered nodes. If `keep_format = TRUE`, matrix, igraph, statnet network and tna inputs are converted back to that type.

See Also

`filter_edges`, `splot`, `subset_nodes`

Examples

```
filter_nodes(regulation_net, degree >= 7)
```

---

fit_degree_distribution

Fit Statistical Distributions to Degree Sequence

Description

Fits one or more statistical distributions to the degree sequence of a network by maximum likelihood and evaluates goodness of fit with Kolmogorov-Smirnov statistics. The comparison table is sorted by AIC.

Usage

```
fit_degree_distribution(
  x,
  distributions = NULL,
  mode = "all",
  directed = NULL,
  xmin = NULL,
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object. Without igraph installed, only a numeric matrix is accepted.
`distributions`	Character vector of distributions to fit. Options: `"power_law"`, `"exponential"`, `"poisson"`, `"geometric"`. Default `NULL` fits all four. An unknown name raises an error.
`mode`	For directed networks: `"all"`, `"in"`, or `"out"`. Determines which degree to extract. Default `"all"`.
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

xmin	Minimum degree to include in fitting. For the power law, NULL triggers automatic estimation (Clauset et al. 2009 via igraph). For the other distributions, NULL defaults to 1. Each fit requires at least two degrees at or above xmin and raises an error otherwise.
...	Additional arguments (currently unused).

Details

The power-law model (Pareto type I) is $P(k) \sim k^{-\alpha}$. With igraph available and xmin = NULL, it is fitted with igraph::fit_power_law(), which implements the Clauset et al. (2009) method. Otherwise the simple MLE $\alpha = 1 + n/\sum \log(k/k_{min})$ is computed, with k_{min} equal to the supplied xmin or, without igraph, to the smallest degree (at least 1).

The exponential model is $P(k) \sim e^{-\lambda k}$, with MLE $\lambda = 1/\bar{k}$.

The Poisson model is $P(k) \sim \lambda^k e^{-\lambda}/k!$, with MLE $\lambda = \bar{k}$.

The geometric model is $P(k) \sim (1-p)^k p$, with MLE $p = 1/(1+\bar{k})$.

ks_p comes from stats::ks.test() for the exponential and Poisson fits and is NA for the power-law and geometric fits. The p-values are approximate because the test assumes a continuous distribution. For the exponential fit stats::ks.test() warns when degrees are tied. A power-law fit to degrees that all equal xmin is degenerate and returns NA for alpha, the likelihood and every statistic.

Each likelihood is computed over the degrees at or above the xmin of that fit. With xmin = NULL, the power-law fit uses the estimated xmin and the other fits use 1, so their AIC values are computed on different subsets of the degrees. AIC and BIC count one free parameter per distribution, so the power-law xmin is not penalized.

Value

An object of class "cograph_degree_fit", a list containing:

fits Named list, one entry per distribution, each with distribution, parameters (named list of fitted parameters), loglik, aic, bic, ks_stat and ks_p. The parameters are alpha and xmin for the power law, lambda for the exponential and Poisson fits, and p for the geometric fit.

comparison Data frame sorted by AIC with columns distribution, aic, bic, ks_stat, ks_p.

best Name of the best-fitting distribution (lowest AIC).

degree The named degree vector used for fitting.

Printing and plotting

Printing the result shows the best-fitting distribution, the comparison table and the fitted parameters of each distribution. plot() on the result is documented in plot-results.

References

Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). Power-law distributions in empirical data. *SIAM Review*, 51(4), 661–703.

See Also

degree_distribution, centrality

Examples

```
fit_degree_distribution(regulation_net,
  distributions = c("poisson", "geometric"))
```

`from_qgraph` *Convert a qgraph object to cograph parameters*

Description

Extracts the network, the layout and the plotting arguments from a qgraph object and passes them to a cograph plotting engine. Node, edge and graph settings are read from the resolved `graphAttributes` of the object. The colors `posCol`, `negCol` and the `theme` are read from its `Arguments`.

Usage

```
from_qgraph(
  qgraph_object,
  engine = c("splot", "soplot"),
  plot = TRUE,
  weight_digits = 2,
  show_zero_edges = FALSE,
  preserve_node_size = FALSE,
  ...
)
```

Arguments

`qgraph_object`	Return value of `qgraph::qgraph()`
`engine`	Which cograph renderer to use: `"splot"` or `"soplot"`. Default: `"splot"`.
`plot`	Logical. If TRUE (default), immediately plot using the chosen engine.
`weight_digits`	Number of decimal places to round edge weights to. Default 2. Edges whose weight rounds to zero at this precision are dropped unless `show_zero_edges = TRUE`.
`show_zero_edges`	Logical. A zero weight means that the edge is absent, so an edge whose weight rounds to zero at `weight_digits` is dropped. With `TRUE` each such edge is plotted at the smallest magnitude that `weight_digits` can express, with its original sign. Other weights are unchanged. Default: `FALSE`.
`preserve_node_size`	Logical. If TRUE, use the node sizes extracted from the qgraph object. Default FALSE uses cograph's standard sizing.
`...`	Overrides for any extracted parameter, given by cograph name (for example `threshold`). The qgraph names `minimum` and `cut` are translated to `threshold` and `edge_cutoff`.

Details

Parameter Mapping:

The following qgraph parameters are extracted and mapped to their cograph equivalents.

Node properties:

- `labels`/`names` -> `labels`
- `color` -> `node_fill`
- `width` -> `node_size` (scaled by 1.3x) when `preserve_node_size = TRUE`
- `shape` -> `node_shape` (mapped to cograph equivalents)
- `border.color` -> `node_border_color`
- `border.width` -> `node_border_width`
- `label.cex` -> `label_size`
- `label.color` -> `label_color`

Edge properties:

- `labels` -> `edge_labels`
- `label.cex` -> `edge_label_size` (scaled by 0.5x)
- `lty` -> `edge_style` (numeric to name conversion)
- `curve` -> `curvature` (only when qgraph resolved a single curvature for the whole graph)
- `asize` -> `arrow_size` (scaled by 0.3x)
- `edge.label.position` -> `edge_label_position`

Graph properties:

- `minimum` -> `threshold`
- `maximum` -> `maximum`
- `groups` -> `groups`
- `directed` -> `directed`
- `posCol`/`negCol` -> `edge_positive_color`/`edge_negative_color`
- `theme` -> `theme`

Pie and donut:

- `pie` values -> `donut_fill` with `donut_inner_ratio = 0.8` and `donut_empty = FALSE`
- `pieColor` -> `donut_color`

Settings that are not extracted:

The edge colors and widths are not extracted, because qgraph stores them with its `cut`-based fading applied. cograph styles the edges by weight instead. The `cut` value is not extracted either.

Layout:

The qgraph layout coordinates are kept with `rescale = FALSE`. When `layout` is supplied in `...`, `rescale` is removed and the new layout is rescaled.

Value

Invisibly, a named list of plotting parameters (the weight matrix `x`, `weight_digits`, the layout and the extracted node, edge and graph settings, after the overrides in `...`). It can be passed to `splot()` with `do.call()`. An input without an `Arguments` field that does not inherit from `"qgraph"` raises an error.

See Also

cograph for creating networks from scratch, splot and soplot for plotting engines, from_tna for tna object conversion

Examples

```
q <- qgraph::qgraph(regulation_net, DoNotPlot = TRUE)
from_qgraph(q)
```

---

from_tna *Convert a tna object to cograph parameters*

Description

Extracts the transition matrix, labels, and initial state probabilities from a tna object and plots the network with a cograph engine. Initial probabilities are mapped to donut fills.

Usage

```
from_tna(
  tna_object,
  engine = c("splot", "soplot"),
  plot = TRUE,
  weight_digits = NULL,
  show_zero_edges = FALSE,
  ...
)
```

Arguments

tna_object	A tna object from tna::tna()
engine	Which cograph renderer to use: "splot" or "soplot". Default: "splot".
plot	Logical. If TRUE (default), immediately plot using the chosen engine.
weight_digits	Number of decimal places to round edge weights to. Default NULL, which picks the number of digits from the matrix: 0 when every non-zero weight is a whole number (counts, as in ftna/ctna models) and 2 otherwise (probabilities). Edges whose weight rounds to zero at this precision are dropped unless show_zero_edges = TRUE.
show_zero_edges	Logical. A zero weight means that the edge is absent, so an edge whose weight rounds to zero at weight_digits is dropped. With TRUE each such edge is plotted at the smallest magnitude that weight_digits can express, with its original sign. Other weights are unchanged. Default: FALSE.
...	Additional parameters passed to the plotting engine (e.g., layout, node_fill, donut_color).

Details

Conversion Process:

The transition matrix (`weights`) supplies the edge weights, the state labels (`labels`) supply the node labels, and the initial state probabilities (`inits`) supply the `donut_fill` values. The donuts are plotted with `donut_inner_ratio = 0.8`.

Directedness is read from the tna object when it is recorded there. Otherwise a symmetric matrix is treated as undirected and an asymmetric matrix as directed.

TNA Visual Defaults:

The following defaults are applied. Each can be overridden through `...`.

- `layout = "oval"`.
- `node_fill`: the RColorBrewer Accent palette for up to 8 states, Set3 for 9 to 12 states, and a qualitative HCL palette for more.
- `node_size = 7`.
- `edge_color = "#003355"`.
- `edge_labels = TRUE`, with `edge_label_size = 0.4` and `edge_label_position = 0.7`.
- `edge_label_style = "estimate"` and `edge_label_leading_zero = FALSE`, so a label shows the weight alone without a leading zero (for example `.42`).
- `minimum = 0.01`, so transitions weaker than 0.01 are not plotted.
- For directed networks, `arrow_size = 0.61`, `edge_start_style = "dotted"` and `edge_start_length = 0.2` (the first 20 percent of each edge is dotted).

Value

Invisibly, a named list of plotting parameters (the weight matrix `x`, `labels`, `directed`, the donut settings and the visual defaults above, after the overrides in `...`). It can be passed to `splot()` with `do.call()`. An input that is not a `tna` object raises an error.

See Also

`cograph` for creating networks from scratch, `splot` and `soplot` for plotting engines, `from_qgraph` for qgraph object conversion

Examples

```
from_tna(tna::tna(regulation_net))
```

---

get_edge_list *Extract Raw Edge List from TNA Model*

Description

Extracts the individual-level transition counts of a tna object as an edge list.

Usage

```
get_edge_list(x, by_individual = TRUE, drop_zeros = TRUE)
```

Arguments

x	A tna object created by tna::tna(). Other inputs raise an error.
by_individual	Logical. If TRUE (default), returns edge list with individual IDs. If FALSE, aggregates across all individuals.
drop_zeros	Logical. If TRUE (default), excludes edges with zero count.

Value

A data frame with columns:

id Row number of the individual in the sequence data (only if by_individual = TRUE).

from Source state label.

to Target state label.

count Number of transitions.

With by_individual = FALSE, the rows are sorted by decreasing count.

See Also

extract_motifs() for motif analysis using edge lists

Other motifs: extract_motifs(), extract_triads(), motif_census(), motifs(), subgraphs(), triad_census()

Examples

```
get_edge_list(tna::tna(coding))
```

---

get_nodes *Access and Modify a Cograph Network*

Description

These functions read and replace the parts of a cograph_network object created by as_cograph. The getters return the node table, the edge table, the node labels, the node groups, the source type, the stored estimation data, the metadata list, and the counts of nodes and edges. The setters return a modified copy of the network.

Usage

```
get_nodes(x)

get_edges(x)

get_labels(x)

get_source(x)

get_data(x)

get_meta(x)

set_nodes(x, nodes_df)

set_edges(x, edges_df)

set_layout(x, layout_df)

set_groups(
  x,
  groups = NULL,
  type = c("group", "cluster", "layer"),
  nodes = NULL,
  layers = NULL,
  clusters = NULL
)

get_groups(x)

nodes(x)

is_directed(x)

n_nodes(x)

n_edges(x)
```

Arguments

x	A cograph_network object. is_directed() also accepts a CographNetwork or an igraph object.
nodes_df	A data frame of node information. A missing id column is filled with row numbers and a missing label column with the ids. The stored weight matrix is rebuilt from the new node table.
edges_df	A data frame with columns from and to (integer row numbers into the node table) and an optional weight column, which defaults to 1. Extra columns are

kept. Each edge may appear once, and an undirected network counts A-B and B-A as the same edge.

`layout_df` A data frame with `x` and `y` columns, or a matrix whose first two columns are used, with one row per node.

`groups` Node groupings in one of these formats:

- A character string naming a community detection method of `detect_communities` (`"louvain"`, `"walktrap"`, `"fast_greedy"`, `"label_prop"`, `"infomap"`, `"leiden"`), which requires the igraph package.
- A named list mapping each group name to a vector of node labels, for example `list(A = c("N1", "N2"), B = c("N3", "N4"))`.
- An unnamed vector with one group assignment per node, in node order.
- A data frame with a `node` (or `nodes`) column and one of `layer`, `cluster` or `group` (plural forms are accepted and normalized to the singular).
- `NULL`, in which case `nodes` and one of `layers` or `clusters` supply the grouping.

`type` Group type stored by `set_groups()`. One of `"group"` (default), `"cluster"` or `"layer"`. It is ignored when `layers` or `clusters` is given, since the type then follows from the argument used.

`nodes` Character vector of node labels, used with `layers` or `clusters` to give groupings as vectors. When `NULL`, the assignments follow the node order of the network.

`layers` Character or factor vector of layer assignments, the same length as `nodes`.

`clusters` Character or factor vector of cluster assignments, the same length as `nodes`.

Value

`get_nodes()`, `nodes()` The node table, with `id` and `label` columns plus layout coordinates or other metadata columns when present. `nodes()` is a deprecated alias of `get_nodes()` and signals a deprecation warning.

`get_edges()` A data frame with one row per edge and columns `from` and `to` (integer row numbers into the node table), `weight`, and any extra edge columns. An undirected network stores one row per unordered pair. `as.data.frame()` returns the same table with node labels as endpoints.

`get_labels()` A character vector of node labels.

`get_groups()` A data frame with a `node` column and one of `layer`, `cluster` or `group`, or `NULL` when no groups are set.

`get_source()` A character string naming the input type (for example `"matrix"`, `"tna"`, `"igraph"`, `"edgelist"`), or `"unknown"`.

`get_data()` The original estimation data (for example the sequence data of a tna model), or `NULL` when none is stored.

`get_meta()` A list with component `source` (input type). Networks built from a tna model also carry `tna` (type, group name and group index).

`n_nodes()`, `n_edges()` An integer count. Each undirected edge counts once.

`is_directed()` A single logical value.

`set_nodes()`, `set_edges()`, `set_layout()`, `set_groups()` The modified `cograph_network`. `set_layout()` writes the coordinates into the `x` and `y` columns of the node table. `set_groups()` stores the grouping as `node_groups` for use by the group-aware plot functions. It stops with an error when a node is assigned twice, when a node is missing from the assignment or unknown to the network, and when fewer than two groups result. `set_edges()` raises an error of class `cograph_bad_selection` for endpoints outside the node table and for duplicated edges.

See Also

`as_cograph`, `splot`, `detect_communities`

Examples

```
net <- as_cograph(regulation_net)
get_edges(net)
```

---

ggplot_robustness *Compare Network Robustness (ggplot2)*

Description

Plots robustness curves for one or more networks with ggplot2, with one facet per network and one line per attack strategy.

Usage

```
ggplot_robustness(
  ...,
  networks = NULL,
  measures = c("betweenness", "degree", "random"),
  strategy = "sequential",
  colors = NULL,
  title = NULL,
  n_iter = 1000,
  seed = NULL,
  type = "vertex",
  ncol = NULL,
  free_y = FALSE
)
```

Arguments

`...`	Networks, with network names as argument names. Unnamed networks are labelled "Network 1", "Network 2", and so on.
`networks`	Named list of networks, used when `...` is empty.
`measures`	Attack strategies to compare. Default c("betweenness", "degree", "random").

strategy	Character string; "sequential" (default) recalculates centrality after each removal, "static" uses initial centrality ranking throughout.
colors	Vector of colors named "Betweenness", "Degree" and "Random". Default NULL uses red, green and blue.
title	Overall title. Default NULL. With a single network the title is replaced by ": sequential removal of nodes".
n_iter	Iterations for random. Default 1000.
seed	Random seed. Default NULL.
type	Removal type. Default "vertex".
ncol	Columns in facet. Default NULL (auto).
free_y	If TRUE, allow different y-axis scales per facet. Default FALSE.

Value

A ggplot object.

Examples

```
ggplot_robustness(regulation = regulation_net, n_iter = 20, seed = 1)
```

---

group_centrality *Group Centrality (Everett-Borgatti 1999)*

Description

Computes the centrality of a set of nodes $C \subseteq V$. Distances are unweighted hop counts in the direction of the edges. The measures are defined as follows.

Usage

```
group_centrality(
  x,
  nodes,
  measure = c("betweenness", "closeness", "degree"),
  mode = c("all", "out", "in"),
  normalized = TRUE
)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
nodes	Integer vector of node indices (1-based) or character vector of node names identifying the group C.
measure	One of "betweenness" (default), "closeness", or "degree".

`mode` — For directed graphs with `measure = "degree"`: `"all"` (both directions, default), `"out"` (outgoing), or `"in"` (incoming). Ignored for undirected graphs and other measures.

`normalized` — Logical, for `"betweenness"` only. If `TRUE` (default), divide by $(|V|-|C|)(|V|-|C|-1)$.

Details

betweenness $GBC(C) = \sum_{s,t \in V \setminus C, s \neq t} \sigma(s,t \mid C)/\sigma(s,t)$, where $\sigma(s,t)$ is the number of shortest s-t paths and $\sigma(s,t \mid C)$ is the number of those paths passing through at least one node in C. Normalized by $1/((|V|-|C|)(|V|-|C|-1))$.

closeness $GCC(C) = (|V|-|C|)/\sum_{v \in V \setminus C} d(v,C)$, where $d(v,C) = \min_{c \in C} d(v,c)$ is the shortest distance from v to any group member. Unreachable nodes contribute 0 to the denominator sum. For directed graphs, $d(v,c)$ follows the edges from v to c.

degree $GDC(C) = |N(C) \setminus C|/(|V|-|C|)$, the fraction of non-group nodes adjacent to at least one group member. For directed graphs, `mode` selects the neighborhood.

Value

Numeric scalar: the group centrality of the set `nodes`. Unknown node names and out-of-range indices raise an error.

Group betweenness

Group betweenness is computed directly from the Everett and Borgatti definition, counting the shortest paths that pass through at least one node in C. On some graphs the result differs from `networkx.group_betweenness_centrality`, which uses the iterative algorithm of Puzis, Elovici and Dolev.

References

Everett, M. G., & Borgatti, S. P. (1999). The centrality of groups and classes. *Journal of Mathematical Sociology*, 23(3), 181-201.

Puzis, R., Elovici, Y., & Dolev, S. (2007). Fast algorithm for successive computation of group betweenness centrality. *Physical Review E*, 76, 056709. doi:10.1103/PhysRevE.76.056709.

See Also

`centrality` for per-node measures.

Examples

```
group_centrality(regulation_net, nodes = c("Plan", "Monitor"), measure = "betweenness")
```

`hai_datasets` *Human-AI Interaction Coding Sequences*

Description

Coded sequences of human-AI programming interactions from 34 projects across 429 sessions. Actions are coded at two granularity levels (broad categories and fine-grained codes) and split by actor (human, AI, or both combined). The AI datasets contain 428 sessions. Each row is one session and every column is a time step: the columns are named T1, T2, ... Tn and hold the sequential actions. `NA` indicates the session ended before that time step.

Usage

```
coding

coding_detailed

ai_coding

ai_detailed

human_ai

human_ai_detailed
```

Format

coding 429 x 164 data.frame. Human actions by category (9 states: Command, Correct, Frustrate, Inquire, Interrupt, Refine, Request, Specify, Verify).

coding_detailed 429 x 164 data.frame. Human actions by fine-grained code (15 states: Accept, Arguing, Ask, Command, Context, Correction, Direct, Frustration, Interrupt, Refinement, Reject, Request, Specification, Thinking, Verification).

ai_coding 428 x 138 data.frame. AI actions by category (8 states: Ask, Delegate, Execute, Explain, Investigate, Plan, Repair, Report).

ai_detailed 428 x 138 data.frame. AI actions by fine-grained code (18 states: Acknowledge, Apologize, Ask, Comply, Delegate, Diagnose, Escape, Execute, Explain, Hedge, Investigate, Plan, Refuse, Report, Retry, Scaffold, Suggest, Warn).

human_ai 429 x 287 data.frame. Both actors combined, by category (17 states).

human_ai_detailed 429 x 287 data.frame. Both actors combined, by fine-grained code (32 states).

An object of class `data.frame` with 429 rows and 164 columns.

An object of class `data.frame` with 429 rows and 164 columns.

An object of class `data.frame` with 428 rows and 138 columns.

An object of class `data.frame` with 428 rows and 138 columns.

An object of class `data.frame` with 429 rows and 287 columns.

An object of class `data.frame` with 429 rows and 287 columns.

Value

Each dataset is a `data.frame` with one row per session and one character column per time step (T1, T2, ... Tn). It has no identifier columns.

Source

Human-AI programming interaction study, 34 projects, 429 sessions.

Examples

```
data(coding)
str(coding, list.len = 6)
dim(coding)
```

---

invert_weights *Invert Edge Weights (Similarity to Distance and Back)*

Description

Turns strong ties into short distances. Path-based measures treat weights as costs, so similarity weights are inverted before such measures are computed.

Usage

```
invert_weights(
  x,
  method = c("reciprocal", "max_minus", "reflect"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`method`	How to invert:
	`"reciprocal"` (default) `1 / w`, the standard similarity-to-distance map.
	`"max_minus"` `max(w) - w`. The strongest edge becomes zero and is therefore dropped; a `cograph_edges_dropped` warning says how many.
	`"reflect"` `max(w) + min(w) - w`. Reverses the order of the weights and keeps every edge when all weights are positive.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with inverted weights, or the input format when `keep_format = TRUE`.

See Also

normalize_weights, shortest_paths

Examples

```
invert_weights(regulation_net, method = "reciprocal")
```

---

is_bipartite *Check if a Matrix Could Be Bipartite*

Description

Tests whether a matrix could represent a bipartite incidence matrix. A non-square matrix is considered bipartite by default. For square matrices, checks whether the corresponding graph has bipartite structure (i.e., nodes can be partitioned into two groups with edges only between groups).

Usage

```
is_bipartite(x)
```

Arguments

x	A numeric matrix.

Details

For non-square matrices, returns TRUE since they naturally represent two-mode data (rows and columns are distinct node types).

For square matrices, the function checks whether the corresponding undirected graph is bipartite by attempting a two-coloring via igraph::bipartite_mapping() when igraph is available. Without igraph, it uses a breadth-first two-coloring. Positive entries define the edges, edge direction is ignored, and the diagonal (self-loops) is dropped before the check.

Value

Logical. TRUE if the matrix could represent a bipartite network, FALSE otherwise.

Examples

```
inc <- matrix(c(1, 0, 1, 1, 1, 0), 2, 3)
cograph::is_bipartite(inc)
```

is_tna_network *Check if Network is TNA-based*

Description

Checks whether a cograph_network was created from a tna object, such as one model of a group_tna object.

Usage

```
is_tna_network(x)
```

Arguments

x	A CographNetwork or cograph_network object.

Value

Logical. TRUE if the network was created from a tna object, FALSE otherwise, including for any input that is not a network.

See Also

as_cograph

Examples

```
is_tna_network(as_cograph(regulation_net))
```

k_shortest_paths *Find K Shortest Loopless Paths (Yen's Algorithm)*

Description

Computes up to k shortest loopless paths between two nodes using Yen's algorithm. Each path is a sequence of distinct nodes from source to target.

Usage

```
k_shortest_paths(x, from, to, k = 3, weights = NULL, directed = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
from	Character or numeric node identifier for the source node.
to	Character or numeric node identifier for the target node.
k	Integer; number of shortest paths to find. Default 3.
weights	Edge weight handling: NULL (default) auto-detects from edge attributes, NA forces unweighted distances, or a numeric vector of custom weights.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
...	Not used. Any argument supplied here raises an "unused argument" error.

Details

Yen's algorithm finds the k shortest loopless (simple) paths in a graph. It works by:

1. Finding the shortest path via Dijkstra's algorithm
2. For each subsequent path, systematically exploring deviations from previously found paths by temporarily removing edges, finding spur paths, and selecting the shortest candidate

The algorithm may return fewer than k paths if fewer distinct loopless paths exist between the two nodes.

Value

A list with class "cograph_k_paths" containing:

paths List of up to k character vectors, each containing node names in path order

distances Numeric vector of path lengths (sum of edge weights or hop count)

from Source node name

to Target node name

k Number of paths requested

References

Yen, J.Y. (1971). Finding the K shortest loopless paths in a network. *Management Science*, 17(11), 712-716. doi:10.1287/mnsc.17.11.712

See Also

shortest_paths

Examples

```
cograph::k_shortest_paths(regulation_net, from = "Plan", to = "Share", k = 3)
```

layer_degree_correlation *Degree Correlation Between Layers*

Description

Measures the consistency of hubs across layers as the Pearson correlation of node degrees between layers.

Usage

```
layer_degree_correlation(layers, mode = c("total", "in", "out"))

ldegcor(layers, mode = c("total", "in", "out"))
```

Arguments

layers	List of adjacency matrices of the same dimensions
mode	Degree type: "total" (default, row plus column sums), "in" (column sums) or "out" (row sums). The sums use the edge weights, so on a weighted layer the degree is the node strength.

Value

An L x L Pearson correlation matrix of the layer degree sequences, with the layer names (or "Layer1", "Layer2", ...) as dimnames.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
layer_degree_correlation(layers, mode = "total")
```

layer_similarity *Layer Similarity*

Description

Computes similarity between two network layers.

Usage

```
layer_similarity(
  A1,
  A2,
  method = c("jaccard", "overlap", "hamming", "cosine", "pearson")
)

lsim(A1, A2, method = c("jaccard", "overlap", "hamming", "cosine", "pearson"))
```

Arguments

A1	First adjacency matrix
A2	Second adjacency matrix, with the same dimensions as A1
method	Comparison method: "jaccard" (default), "overlap", "hamming", "cosine" or "pearson"

Details

"jaccard", "overlap" and "hamming" compare edge presence (A > 0) and ignore weights. "cosine" and "pearson" are computed on the cell values, diagonal included. Matrices of different dimensions raise an error.

Value

A single numeric value. All methods except "hamming" return a similarity, where higher values mean more alike layers. "hamming" returns a distance, the number of matrix cells whose edge presence differs between the two layers. Lower values then mean more alike layers, and the value is not bounded by 1. NA is returned when the denominator is undefined ("jaccard" with no edges in either layer, "overlap" with an empty layer, "cosine" with an all-zero layer).

Examples

```
layer_similarity(regulation_net, t(regulation_net), method = "cosine")
```

---

layer_similarity_matrix

Pairwise Layer Similarities

Description

Computes the similarity of every pair of layers with layer_similarity.

Usage

```
layer_similarity_matrix(
  layers,
  method = c("jaccard", "overlap", "cosine", "pearson")
)

lsim_matrix(layers, method = c("jaccard", "overlap", "cosine", "pearson"))
```

Arguments

layers	Named list of adjacency matrices (one per layer); at least two are required.
method	Comparison method: "jaccard" (default), "overlap", "cosine" or "pearson". The "hamming" distance is not accepted.

Value

A symmetric L x L matrix of pairwise similarities with 1 on the diagonal. The dimnames are the layer names, or "Layer1", "Layer2", ... for an unnamed list.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
layer_similarity_matrix(layers, method = "cosine")
```

---

layout_circle *Circular Layout*

Description

Places the nodes at evenly spaced positions on a circle of radius 0.4 centered at (0.5, 0.5). A single node is placed at the center.

Usage

```
layout_circle(network, order = NULL, start_angle = pi/2, clockwise = TRUE, ...)
```

Arguments

network	A CographNetwork or cograph_network object.
order	Optional vector of node indices or labels. Its i-th element is the node placed at the i-th position. Labels are matched only for a CographNetwork object. An order of the wrong length, or with unmatched labels, raises a warning and the default order is used.
start_angle	Angle in radians of the reference position. Default pi/2 (top of the circle).
clockwise	Logical. Default TRUE places the positions clockwise, with the last position at start_angle and the first position one step clockwise of it. FALSE places the first position at start_angle and continues counterclockwise.
...	Ignored.

Value

A data frame with columns x and y and one row per node, in node order.

Examples

```
layout_circle(CographNetwork$new(regulation_net))
```

layout_groups *Group-based Layout*

Description

Places the nodes by group membership. The group centers lie on a circle around (0.5, 0.5), starting at the top and proceeding counterclockwise in the order of the group levels. A single group is centered at (0.5, 0.5). The nodes of each group lie on a circle around their group center, and a group with one node is placed at its center.

Usage

```
layout_groups(
  network,
  groups,
  group_positions = NULL,
  inner_radius = 0.15,
  outer_radius = 0.35
)
```

Arguments

network	A CographNetwork or cograph_network object.
groups	Vector of group memberships, one per node (numeric, character, or factor). A length different from the number of nodes raises an error.
group_positions	Optional list or data frame with columns x and y giving the group centers, one row per group level.
inner_radius	Radius of the circle of nodes within each group. Default 0.15.
outer_radius	Radius of the circle of group centers. Default 0.35.

Value

A data frame with columns x and y and one row per node, in node order.

Examples

```
layout_groups(CographNetwork$new(regulation_net),
  groups = rep(c("A", "B"), each = 5))
```

`layout_oval` *Oval Layout*

Description

Arrange nodes evenly spaced around an ellipse. This creates an oval-shaped network layout that is wider than it is tall (or vice versa depending on ratio).

Usage

```
layout_oval(
  network,
  ratio = 1.5,
  order = NULL,
  start_angle = pi/2,
  clockwise = TRUE,
  rotation = 0,
  ...
)
```

Arguments

`network`	A CographNetwork or cograph_network object.
`ratio`	Aspect ratio (width/height). Values > 1 create horizontal ovals, values < 1 create vertical ovals. Default 1.5.
`order`	Optional vector specifying node order, as node indices or as node labels. Labels are matched for both `CographNetwork` and `cograph_network` input.
`start_angle`	Starting angle in radians (default: pi/2 for top).
`clockwise`	Logical. Arrange nodes clockwise? Default TRUE.
`rotation`	Rotation angle in radians to tilt the entire oval. Default 0.
`...`	Additional arguments (ignored).

Value

Data frame with x, y coordinates.

Examples

```
layout_oval(CographNetwork$new(regulation_net), ratio = 1.5)
```

layout_registry *Layout Registry*

Description

Layout algorithms are stored in a registry and selected by name through the `layout` argument of `splot`, `soplot` and `sn_layout`. `register_layout()` adds a layout function, `get_layout()` returns the function of a registered layout, and `list_layouts()` returns the names of all registered layouts. Registering an existing name replaces that layout for the rest of the session.

Usage

```
register_layout(name, layout_fn)

get_layout(name)

list_layouts()
```

Arguments

`name`	Character. The name of the layout.
`layout_fn`	A function that computes node positions. It receives the network as the argument `network`, followed by any layout parameters, and returns a matrix or data frame with `x` and `y` columns.

Value

`register_layout()` returns `NULL` invisibly. `get_layout()` returns the layout function, or `NULL` if no layout has that name. `list_layouts()` returns a character vector of layout names.

See Also

`layout_circle`, `layout_spring`, `layout_groups`, `layout_oval`

Examples

```
list_layouts()
splot(regulation_net, layout = "circle")
```

layout_saqr *Saqr Layout (Start/End transition flow)*

Description

Places the nodes of a directed transition network in rows between a Start and an End node (Saqr et al., LAK25). The Start node is alone on the top row and the End node, when present, is alone on the bottom row. The other nodes are ranked by the weight of the edge they receive from Start, with the strongest nearest Start. They are split into two middle rows when there are at most 10 of them and into three rows otherwise. A sine envelope narrows the rows near Start and End, which gives the layout a lens shape. The first middle row is offset in a zig-zag pattern.

Usage

```
layout_saqr(network, start = "Start", end = "End", jitter = 0.32, ...)
```

Arguments

network	A CographNetwork or cograph_network object.
start	Label of the Start node (default "Start"). When the label is not found, the node with the largest sum of outgoing weights is used.
end	Label of the End node (default "End"). The End row is omitted when the label is not found.
jitter	Numeric in [0, 1]. Zig-zag amount applied to the first middle row, as a fraction of the row spacing (default 0.32).
...	Additional arguments (ignored).

Value

Data frame with x, y coordinates, one row per node.

Examples

```
layout_saqr(CographNetwork$new(regulation_net), start = "Explore",
  end = "Share")
```

layout_spring *Fruchterman-Reingold Spring Layout*

Description

Computes node positions with the Fruchterman-Reingold force-directed algorithm. Nodes connected by edges attract each other and all nodes repel each other. Attraction along an edge is scaled by the absolute edge weight. Duplicate edges between the same pair of nodes, in either direction, are merged by summing their weights.

Usage

```
layout_spring(
  network,
  iterations = 200,
  cooling = 0.95,
  repulsion = 1.5,
  attraction = 1,
  seed = NULL,
  initial = NULL,
  max_displacement = NULL,
  anchor_strength = 0,
  area = 1.5,
  gravity = 0,
  init = c("random", "circular"),
  cooling_mode = c("exponential", "vcf", "linear"),
  ...
)
```

Arguments

network	A CographNetwork or cograph_network object.
iterations	Number of iterations (default: 200).
cooling	Factor by which the temperature is multiplied after each iteration when cooling_mode = "exponential" (default: 0.95).
repulsion	Repulsion constant (default: 1.5).
attraction	Attraction constant (default: 1).
seed	Random seed for the random initial positions. NULL (default) uses the current random state. When a seed is given, the caller's random state is restored on exit.
initial	Optional initial coordinates, a matrix or a data frame with columns x and y. When supplied, init is ignored. For animations, pass the previous frame's layout to obtain smooth transitions.
max_displacement	Maximum distance a node can move from its initial position (default: NULL, no limit). Applies only when initial is supplied. In that case the coordinates are returned without the final rescaling. Values such as 0.05 to 0.1 suit animations.

anchor_strength
: Strength of the force pulling nodes toward their initial positions (default: 0). Higher values (for example 0.5 to 2) keep nodes closer to their starting positions. Applies only when `initial` is supplied.

area
: Area parameter (default: 1.5). It sets the ideal edge length $k = \sqrt{area/n}$ and the initial temperature $\sqrt{area}/10$. Because the final coordinates are rescaled, `area` changes the relative arrangement of the nodes and leaves the overall extent unchanged.

gravity
: Strength of the force pulling nodes toward the centroid of the layout (default: 0). Higher values (for example 0.5 to 2) give a more compact layout.

init
: Initialization method, `"random"` (default) or `"circular"`.

cooling_mode
: Cooling schedule. `"exponential"` (default) multiplies the temperature by `cooling` after each iteration. `"vcf"` (variable cooling factor) multiplies it by 0.9 when the mean displacement is below $0.1k$ and by 0.99 otherwise. `"linear"` multiplies it by $1 - t/T$ at iteration t of T.

...
: Additional arguments (ignored).

Value

A data frame with columns `x` and `y` and one row per node. The coordinates are centered on their mean and scaled by a common factor into $[0.05, 0.95]$, which preserves the aspect ratio of the layout. A network without edges returns the initial positions, and a single node is placed at (0.5, 0.5).

Examples

```
layout_spring(CographNetwork$new(regulation_net), seed = 42)
```

---

layout_target *Target Layout (focal-node, topological)*

Description

Adapts the `flow()` layout of qgraph. One node of interest (the `target`) is placed alone, and every other node is placed in successive levels ordered by its unweighted graph distance (number of hops) from the target. Edge direction is ignored. The layout shows how the target node connects to the rest of the network.

Usage

```
layout_target(network, target = NULL, horizontal = TRUE, equalize = TRUE, ...)
```

Arguments

network	A CographNetwork or cograph_network object.
target	Node of interest, given as a label (character) or 1-based index. When NULL (default) the node with the most neighbors is used. A label that is not found or an index out of range raises an error.
horizontal	Logical. If TRUE (default) levels flow left to right with the target node on the left; if FALSE they flow top to bottom.
equalize	Logical. If TRUE (default) nodes are evenly spaced within each level.
...	Additional arguments (ignored).

Details

Weights are binarized for layering, so only connectivity matters. Nodes that cannot be reached from the target are placed in one extra level after the last. qgraph raises an error for such nodes.

Value

Data frame with x, y coordinates, one row per node.

Examples

```
layout_target(CographNetwork$new(regulation_net), target = "Plan")
```

---

list_centralities *Catalogue of the Centrality Measures*

Description

Returns a table of every measure centrality can compute. For each measure the table records which end of the scale marks a prominent node, whether the measure accepts mode, whether it requires a community partition, whether it uses edge weights, and whether type = "all" holds it back because its cost grows steeply with network size.

Usage

```
list_centralities(orientation = NULL, costly = NULL, needs_membership = NULL)
```

Arguments

orientation	Keep only measures with this orientation: "higher" or "lower". Default NULL keeps both.
costly	Keep only costly measures (TRUE) or only the rest (FALSE). Default NULL keeps both.
needs_membership	Keep only measures that require a partition (TRUE) or only those that do not (FALSE). Default NULL keeps both.

Details

For some measures a low value marks the more central node, so a descending sort of their column places the most peripheral nodes first. `orientation = "lower"` lists them.

Value

A `data.frame` with one row per measure and the columns `measure` (the name to pass to `centrality(measures = )`), `orientation` (`"higher"` or `"lower"`, which end of the scale marks a prominent node), `mode_aware` (whether the measure accepts `mode` and its column carries a mode suffix), `needs_membership`, `uses_weights`, and `costly` (held back from `type = "all"`; add it with `include = `). Rows are ordered by measure name.

See Also

`centrality` to compute them, `centrality_degree` and the other one-measure verbs.

Examples

```
list_centralities(orientation = "lower")
```

`membership` *Get Community Membership*

Description

Extracts a named membership vector from a `cograph_communities` data frame or an igraph `communities` object.

Usage

```
membership(x)
```

Arguments

`x`	A `cograph_communities` or igraph `communities` object.

Value

A numeric vector of community numbers named by node. For an igraph object it is the igraph `membership` vector.

Examples

```
comm <- community_walktrap(regulation_net)
membership(comm)
```

`motif_census` *Network Motif Analysis*

Description

Counts the subgraph classes (motifs) of size 3 or 4 in a network and tests their frequencies against random networks from a null model. Edge weights are ignored, and self-loops and multiple edges are removed before counting.

Usage

```
motif_census(
  x,
  size = 3,
  n_random = 100,
  method = c("configuration", "gnm"),
  directed = NULL,
  seed = NULL
)
```

Arguments

`x`	A matrix, igraph object, or cograph_network.
`size`	Motif size: 3 (triads) or 4 (tetrads). Default 3.
`n_random`	Number of random networks for the null model. Must be a whole number of at least 2. Default 100.
`method`	Null model method. `"configuration"` (default) rewires the graph with degree-preserving edge swaps. `"gnm"` draws random graphs with the same numbers of nodes and edges.
`directed`	Logical or NULL. Whether the network is treated as directed. NULL (default) treats a matrix as directed when it is not symmetric and takes the directedness of an igraph or cograph_network input. A value that conflicts with an igraph or cograph_network input raises an error.
`seed`	Random seed for reproducibility. Default NULL. When supplied, the caller's RNG state is saved and restored.

Details

Printing the result shows the motif table with the null-model settings and the number of over- and under-represented motifs. The result is a data frame and serves as the tidy table directly. `plot()` on the result is documented in `plot-results`.

Value

A cograph_motifs data frame with one row per motif class and columns:

motif Motif class name. Directed triads use the 16 MAN codes, undirected triads the classes empty, edge, wedge and triangle, and size 4 the igraph isomorphism class labels M1 to M218 (directed) or M1 to M11 (undirected).

count Observed number of that motif in the network.

null_mean, null_sd Mean and standard deviation of the count across the n_random null graphs.

z_score (count - null_mean) / null_sd. When null_sd = 0, it is 0 if the count equals the null mean and NA otherwise.

p_value Two-sided empirical permutation p-value with add-one correction, based on the absolute deviation from the null mean.

significant Logical, p_value < 0.05.

The motif size ("size"), directed flag ("directed"), null-model method ("method"), and number of random networks ("n_random") are stored as attributes.

See Also

motifs() for the unified API, extract_motifs() for detailed triad extraction, plot-results for plotting

Other motifs: extract_motifs(), extract_triads(), get_edge_list(), motifs(), subgraphs(), triad_census()

Examples

```
motif_census(regulation_net, n_random = 20, seed = 1)
```

---

motifs *Network Motif Analysis*

Description

Classifies the node triples of a network into the 16 directed MAN triad types and tests their frequencies against a permutation null. The function has two modes.

- In census mode (named_nodes = FALSE, the default), the result counts the triads of each MAN type. Nodes are exchangeable.
- In instance mode (named_nodes = TRUE, or subgraphs()), the result lists the node triples that form each type.

Usage

```
motifs(
  x,
  named_nodes = FALSE,
  actor = NULL,
  window = NULL,
  window_type = c("rolling", "tumbling"),
  pattern = c("triangle", "network", "closed", "all"),
  include = NULL,
  exclude = NULL,
  significance = TRUE,
  n_perm = 1000L,
  cores = 1L,
  min_count = if (named_nodes) 5L else NULL,
  edge_method = c("any", "expected", "percent"),
  edge_threshold = 1.5,
  min_transitions = 5,
  top = NULL,
  seed = NULL
)

## S3 method for class 'cograph_motif_result'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  what = c("results", "types")
)
```

Arguments

x	Input data: a tna object, cograph_network, matrix, igraph object or edge-list data frame. For `as.data.frame()`, a `cograph_motif_result` object returned by `motifs()` or `subgraphs()`.
named_nodes	Logical. If FALSE (default), the MAN type census is computed. If TRUE, the individual node triples are listed. `subgraphs()` sets this to TRUE.
actor	Character. Name of the edge-list column that identifies the units. If NULL (default), the first column named `session_id`, `session`, `actor`, `user`, `participant`, `individual` or `id` (in that order, ignoring case) is used. Without such a column the analysis is aggregate.
window	Numeric. Window size for edge-list input. Each actor's transitions are split into windows of this size. NULL (default) applies no windowing.
window_type	Character. `"rolling"` (default) or `"tumbling"`. Used only when `window` is set.
pattern	Which MAN triad types to include in the analysis:
	`"triangle"` (default) The 7 closed triangle types 030C, 030T, 120C, 120D, 120U, 210 and 300.

"network" All types except 003 (empty), 012 (single edge) and 021C (chain).
"closed" All types except 003, 012, 021C and 120C.
"all" All 16 MAN types.

include — Character vector of MAN types to keep. When supplied, pattern and exclude are ignored.

exclude — Character vector of MAN types to drop in addition to those removed by pattern.

significance — Logical. If TRUE (default), a permutation significance test is run. In instance mode the test requires individual data, and for aggregate input it is skipped with a warning.

n_perm — Number of permutations for significance. When significance = TRUE, must be a whole number of at least 2. Default 1000.

cores — Number of worker processes for the permutation null. Default 1 runs serially and draws all replicates from a single RNG stream. cores > 1 gives each replicate its own L'Ecuyer-CMRG stream, so the result depends on seed alone and is the same for every worker count. The serial and parallel streams differ, so a serial run and a parallel run with the same seed give different p-values. Forking is used where available, and Windows uses a PSOCK cluster. Only the individual-level census null is parallelized. Values above parallel::detectCores() are capped with a cograph_cores_capped warning.

min_count — Inclusive minimum count for a row to be kept. In census mode it filters the count column, the number of triads of each MAN type. In instance mode it filters the observed column. At individual level this is the number of units showing the triad, and at aggregate level it is the weighted edge mass of the triad (the sum of its 6 directed edge weights). Default 5 in instance mode and NULL (no filter) in census mode.

edge_method — Method for determining edge presence: "any" (default; any positive edge), "expected" (ratio of the observed weight to the weight expected from the row and column totals), or "percent" (edge weight divided by the six-edge triad total).

edge_threshold — Threshold for "expected" or "percent" methods. For "expected", 1.5 means 50 percent above expected. For "percent", values at or below 1 are proportions and values above 1 are percentages. Default 1.5.

min_transitions — Minimum total edge weight for a unit to be included. Default 5. At aggregate level the network is the only unit, so a network whose weights sum to less than this value gives NULL.

top — Integer or NULL. Only the first top rows of the results are kept. NULL (default) keeps all rows.

seed — Random seed. Default NULL. When supplied, the caller's RNG state is restored on exit.

row.names, optional — Standard as.data.frame arguments. row.names replaces the default row names; optional is ignored.

... — Unused.

what — Which table as.data.frame() returns, either "results" (default) or "types". The Value section lists the columns of each.

Details

The input type and the analysis level are detected automatically. Inputs that carry per-unit data are analyzed per unit. These are tna objects, networks that store tna sequence data, and edge lists (or networks built from edge lists) with an actor column. Matrices, igraph objects and other networks are analyzed as one aggregate network. The adjacency is always classified as directed. The four-class undirected census is computed by `motif_census(..., directed = FALSE)`.

For aggregate inputs, significance is computed by `motif_census()` with its degree-preserving rewiring null on the simple loop-free graph. Individual weighted inputs use a directed stub-matching null. Each positive edge weight is converted to at least one integer stub, and target stubs are shuffled within each unit so that its integer in- and out-margins are preserved. The resulting multigraph, which may contain loops or parallel edges, is classified through its simple loop-free triad projection. Self-loops are removed before counting and before the null is constructed.

With `edge_method = "percent"`, edge presence is computed within each node triple. The weight of an edge is divided by the sum of the six possible directed edge weights of that triple. A threshold above 1 is read as a percentage (1.5 means 1.5 percent), and a threshold at or below 1 is read as a proportion.

With an `edge_method` other than `"any"`, the significance test has limits. For aggregate census input, the observed counts use the threshold while the null tests the unthresholded network, and a warning is raised. For individual census input, the threshold is reapplied to each stub-null replicate. For individual instance input, the null classifies raw stub presence and does not reapply `edge_method` or `edge_threshold`. In every weighted individual null, a positive fractional weight keeps at least one stub. This preserves the support but can change the weight scale used by `"percent"` and `"expected"`. Descriptive results with `significance = FALSE` or `edge_method = "any"` are unaffected.

Value

A `cograph_motif_result` object, or NULL with a message when no motif passes the filters. The object is a list with the elements below.

results Data frame of results. In census mode it has one row per retained, observed MAN type and the columns `type` and `count`. In instance mode it has one row per node triple and MAN type and the columns `triad`, `node1`, `node2`, `node3`, `type` and `observed`. At individual level, `observed` is the number of units in which the triple has that type, so one triple can occupy several rows. With `significance = TRUE`, the columns `expected`, `z`, `p` and `sig` are added and the rows are sorted by decreasing absolute z-score. Otherwise the rows are sorted by decreasing count.

type_summary Named `table` of counts per MAN type, sorted in decreasing order. In census mode it holds the `count` column. In instance mode it holds the number of node triples of each type.

level `"individual"` when the input carried per-unit data, otherwise `"aggregate"`.

named_nodes The value of the `named_nodes` argument.

n_units Number of units analyzed. 1 at aggregate level.

params List of the analysis settings: `labels`, `n_states`, `pattern`, `edge_method`, `edge_threshold`, `significance`, `n_perm`, `min_count`, `window`, `window_type` and `actor`.

`as.data.frame()` returns one of these tables as a plain data frame. With `what = "results"` (default) it returns the `results` table described above. With `what = "types"` it returns one row per

MAN type with columns `type` and `count`, where `count` is the number of triads of that type in a census or the number of node triples of that type in instance mode.

Printing and plotting

Printing the result shows the analysis settings, the MAN type distribution and the first 20 rows of the results table. `as.data.frame()` returns the tidy tables. `plot()` on the result is documented in `plot-results`.

See Also

`subgraphs()`, `motif_census()`, `extract_motifs()`

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `subgraphs()`, `triad_census()`

Examples

```
census <- motifs(regulation_net, significance = FALSE)
as.data.frame(census, what = "types")
```

---

mutate_edges *Add or Change Edge Attributes*

Description

Evaluates expressions against the edge table and stores the results as edge columns.

Usage

```
mutate_edges(
  x,
  ...,
  community = "louvain",
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`...`	Named expressions evaluated against the edge table, with the same metrics and predicates `select_edges()` offers, for example `strong = abs_weight > 0.5` or `scaled = weight / max(weight)`.
`community`	Community detection method used when an expression refers to `same_community`, `from_community` or `to_community`. Default `"louvain"`.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A cograph_network whose edge table has the new columns, or the input format when keep_format = TRUE.

See Also

mutate_nodes, select_edges

Examples

```
as.data.frame(mutate_edges(regulation_net, strong = weight > 0.2))
```

---

mutate_nodes *Add or Change Node Attributes*

Description

Evaluates expressions against the node table, with the same centrality and structural vocabulary that select_nodes() offers, and stores the results as node columns.

Usage

```
mutate_nodes(x, ..., keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
...	Named expressions, for example hub = degree > 3 or score = pagerank * 100. Available names are the existing node columns plus every measure and predicate listed under select_nodes.
keep_format	Logical. Return the input format when TRUE. Note that only igraph and cograph_network formats can carry node attributes; a matrix cannot, and the new columns are lost.
directed	Logical or NULL. If NULL (default), auto-detect.

Value

A cograph_network whose node table has the new columns, or the input format when keep_format = TRUE.

See Also

mutate_edges, select_nodes, centrality

Examples

```
as.data.frame(mutate_nodes(regulation_net, deg = degree), what = "nodes")
```

n_communities *Get Number of Communities*

Description

Get Number of Communities

Usage

```
n_communities(x)
```

Arguments

x	A cograph_communities object.

Value

A single integer, the number of distinct communities.

Examples

```
comm <- community_walktrap(regulation_net)
n_communities(comm)
```

neighborhood_overlap *Neighborhood Overlap (Jaccard) for Each Edge*

Description

Convenience wrapper around edge_centrality that returns only the overlap measure sorted by overlap descending.

Usage

```
neighborhood_overlap(x, top = NULL, directed = NULL, digits = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
top	Integer or NULL. Return only the top N edges. Default NULL.
directed	Logical or NULL. Default NULL (auto-detect).
digits	Integer or NULL. Round numeric columns. Default NULL.
...	Additional arguments passed to edge_centrality.

Value

A data frame sorted by `overlap` (descending) with columns: `from`, `to`, `weight`, `overlap`, `shared_neighbors`.

See Also

`edge_centrality`, `simmelian_strength`

Examples

```
cograph::neighborhood_overlap(regulation_net, top = 5)
```

---

network_bridges *Bridge Edges*

Description

Finds edges whose removal would disconnect the network. These are critical edges for network connectivity.

Usage

```
network_bridges(x, count_only = FALSE, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object
`count_only`	Logical. If TRUE, return only the count. Default FALSE.
`...`	Passed to `to_igraph`, whose only other argument is `directed`; anything else raises an "unused argument" error.

Value

If `count_only = FALSE`, a data frame with one row per bridge and columns `from` and `to` (node names, or integer indices when the graph has no names). If `count_only = TRUE`, an integer count.

Examples

```
strong <- filter_edges(regulation_net, weight > 0.3, keep_isolates = FALSE)
network_bridges(strong)
```

network_clique_size *Largest Clique Size*

Description

Computes the size of the largest clique (complete subgraph) in the network, also called the clique number or omega of the graph.

Usage

```
network_clique_size(x, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Details

A clique is defined on undirected ties, so a directed network is read with each pair of nodes joined when either direction is present, and loops and repeated edges are dropped before counting.

Value

Numeric scalar: the size of the largest clique.

Examples

```
network_clique_size(regulation_net)
```

network_cut_vertices *Cut Vertices (Articulation Points)*

Description

Finds nodes whose removal would disconnect the network. These are critical nodes for network connectivity.

Usage

```
network_cut_vertices(x, count_only = FALSE, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
count_only	Logical. If TRUE, return only the count. Default FALSE.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

If count_only = FALSE, a character vector of node names, or an integer vector of node indices when the graph has no names. If count_only = TRUE, an integer count.

Examples

```
strong <- filter_edges(regulation_net, weight > 0.3, keep_isolates = FALSE)
network_cut_vertices(strong)
```

---

network_girth *Network Girth (Shortest Cycle Length)*

Description

Computes the girth of a network, the length of its shortest cycle. Edge direction, self-loops and repeated undirected edges are ignored. A pair of reciprocal directed edges counts as a cycle of length 2, so a directed network with any mutual tie has girth 2. An undirected forest has girth Inf.

Usage

```
network_girth(x, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

Numeric scalar: the length of the shortest cycle, or Inf if the graph has no cycle.

Examples

```
network_girth(regulation_net)
```

`network_global_efficiency` *Global Efficiency*

Description

Computes the global efficiency of a network, the average of the inverse shortest path lengths between all ordered pairs of distinct nodes. Higher values indicate more efficient global communication. Unreachable pairs contribute 0. A graph with fewer than two nodes returns `NA`.

Usage

```
network_global_efficiency(
  x,
  directed = NULL,
  weights = NULL,
  invert_weights = NULL,
  alpha = 1,
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object
`directed`	Logical or NULL. Consider edge direction? Default NULL, which follows the directedness of the converted graph.
`weights`	Numeric vector of edge weights. Default NULL uses the graph's `weight` attribute when present. NA ignores weights, so every edge has length 1, whatever `invert_weights` is.
`invert_weights`	Logical or NULL. If TRUE, weights are converted to distances as $1/w^{\alpha}$, so stronger ties give shorter paths. If FALSE, weights are used as distances. Default NULL uses TRUE for tna objects and FALSE otherwise.
`alpha`	Numeric. Exponent for weight inversion. Default 1.
`...`	Passed to `to_igraph`, which accepts no arguments besides `directed`.

Value

Numeric scalar: the global efficiency. For unweighted simple graphs it lies in $[0, 1]$. Weighted graphs can exceed 1 when edge distances are below 1.

Examples

```
network_global_efficiency(regulation_net)
```

network_local_efficiency
Local Efficiency

Description

Computes the average local efficiency across all nodes with `igraph::average_local_efficiency()`. For each node, igraph removes the node and measures the distances between its neighbors through the rest of the network. The value can therefore exceed the Latora and Marchiori (2001) form, which restricts those distances to the subgraph induced on the neighbors. `centrality(x, measures = "local_efficiency")` reports the induced-subgraph form. The two agree when the neighbors have no path outside their induced subgraph.

Usage

```
network_local_efficiency(
  x,
  weights = NULL,
  invert_weights = NULL,
  alpha = 1,
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object
`weights`	Numeric vector of edge weights. Default NULL uses the graph's `weight` attribute when present. NA ignores weights, so every edge has length 1, whatever `invert_weights` is.
`invert_weights`	Logical or NULL. If TRUE, weights are converted to distances as $1/w^{\alpha}$, so stronger ties give shorter paths. If FALSE, weights are used as distances. Default NULL uses TRUE for tna objects and FALSE otherwise.
`alpha`	Numeric. Exponent for weight inversion. Default 1.
`...`	Passed to `to_igraph`, whose only other argument is `directed`; anything else raises an "unused argument" error.

Value

Numeric scalar: the average local efficiency, or `NA` for a graph with fewer than two nodes. For unweighted simple graphs it lies in $[0, 1]$. Weighted graphs can exceed 1 when edge distances are below 1.

Examples

```
network_local_efficiency(regulation_net)
```

network_radius *Network Radius*

Description

Computes the radius of a network, the minimum eccentricity across all nodes. The eccentricity of a node is its largest shortest-path distance to any other node. Edge weights are used as distances, and directed networks use outgoing paths.

Usage

```
network_radius(x, directed = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
directed	Logical or NULL. Consider edge direction? Default NULL, which follows the directedness of the converted graph.
...	Passed to to_igraph, which accepts no arguments besides directed; anything else raises an "unused argument" error.

Value

Numeric scalar: the network radius.

Examples

```
network_radius(regulation_net)
```

network_rich_club *Rich Club Coefficient*

Description

Computes the rich club coefficient for a degree threshold k, the density of ties among nodes with degree above k. It measures the tendency of high-degree nodes to connect to each other. Degrees are taken on the undirected simple skeleton of the network.

Usage

```
network_rich_club(x, k = NULL, normalized = FALSE, n_random = 10, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
k	Degree threshold. Only nodes with degree greater than k are included. Default NULL uses the median degree.
normalized	Logical. If TRUE, divide by the mean coefficient of random graphs with the same degree sequence. Default FALSE.
n_random	Number of random graphs for normalization. Default 10.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

Numeric scalar: the rich club coefficient. A normalized value above 1 indicates a rich club effect. NA when fewer than two nodes exceed k.

Reproducibility

When normalized = TRUE the null graphs are drawn from the caller's RNG stream; this function takes no seed argument and does not save or restore .Random.seed. Call set.seed() beforehand for a reproducible result. rich_club() offers a seed argument, confidence intervals, and the full rich club curve.

Examples

```
network_rich_club(regulation_net, k = 5)
```

---

network_small_world *Small-World Coefficient (Sigma)*

Description

Computes the small-world coefficient

$$\sigma = \frac{C/C_{rand}}{L/L_{rand}}$$

where C is the global clustering coefficient, L is the mean shortest path length, and C_{rand} and L_{rand} are their means over Erdos-Renyi graphs with the same numbers of nodes and edges. A directed network is collapsed to undirected first.

Usage

```
network_small_world(x, n_random = 10, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
n_random	Number of Erdos-Renyi comparison graphs (same n and m as the observed graph). Default 10.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Details

Values above 1 indicate small-world structure.

Value

Numeric: small-world coefficient sigma. NA when the graph has fewer than 4 nodes, no edges, or an undefined/zero mean path length.

Reproducibility

The comparison graphs are drawn from the caller's RNG stream; this function takes no seed argument and does not save or restore .Random.seed. Call set.seed() beforehand for a reproducible result. A larger n_random gives a more stable estimate.

Examples

```
set.seed(1)
network_small_world(regulation_net, n_random = 5)
```

---

network_summary *Network-Level Summary Statistics*

Description

Computes network-level statistics and returns them as a one-row data frame. The basic set covers size, density, connectivity, path lengths, centralization, transitivity, reciprocity and degree assortativity.

Usage

```
network_summary(
  x,
  directed = NULL,
  weighted = TRUE,
  mode = "all",
  loops = TRUE,
  simplify = "sum",
  detailed = FALSE,
```

```
  extended = FALSE,
  digits = 3,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
weighted	Logical. Use edge weights for strength, shortest-path, and centrality calculations where the underlying igraph routine accepts them. Default TRUE.
mode	For directed networks: "all", "in", or "out". Used by the degree, strength and closeness statistics added by `detailed = TRUE`. Default "all".
loops	Logical. If TRUE (default), keep self-loops. Set FALSE to remove them.
simplify	How to combine multiple edges between the same node pair. Options: "sum" (default), "mean", "max", "min", or FALSE/"none" to keep multiple edges.
detailed	Logical. If TRUE, add 11 summary statistics of node-level centralities to the 16 basic metrics. Default FALSE.
extended	Logical. If TRUE, add 8 structural metrics (girth, radius, vertex connectivity, clique size, cut vertices, bridges, global and local efficiency). Default FALSE.
digits	Integer. Round numeric results to this many decimal places. Default 3. NULL skips rounding.
...	Additional arguments (currently unused)

Value

A data frame with one row. The basic measures are always computed:

node_count Number of nodes in the network

edge_count Number of edges in the network

density Edge density (proportion of possible edges)

component_count Number of connected components

diameter Longest shortest path in the network

mean_distance Average shortest path length

min_cut Minimum number of edges whose removal disconnects the network. Edge weights are not used.

centralization_degree Degree centralization over all ties (0-1)

centralization_in_degree In-degree centralization (directed only)

centralization_out_degree Out-degree centralization (directed only)

centralization_betweenness Betweenness centralization (0-1)

centralization_closeness Closeness centralization (0-1)

centralization_eigen Eigenvector centralization (0-1)

transitivity Global clustering coefficient

reciprocity Proportion of mutual edges (directed only)

assortativity_degree Degree assortativity coefficient

The extended measures are added when `extended = TRUE`:

girth Length of shortest cycle (Inf if acyclic)

radius Minimum eccentricity over all nodes

vertex_connectivity Minimum nodes to remove to disconnect graph

largest_clique_size Size of the largest complete subgraph

cut_vertex_count Number of articulation points (cut vertices)

bridge_count Number of bridge edges

global_efficiency Average inverse shortest path length

local_efficiency Average local efficiency across nodes

The detailed measures are added when `detailed = TRUE`:

mean_degree, sd_degree, median_degree Degree distribution statistics

mean_strength, sd_strength Weighted degree statistics

mean_betweenness Average betweenness centrality

mean_closeness Average closeness centrality

mean_eigenvector Average eigenvector centrality

mean_pagerank Average PageRank

mean_constraint Average Burt's constraint

mean_local_transitivity Average local clustering coefficient

Examples

```
network_summary(regulation_net)
```

---

network_vertex_connectivity

Network Vertex Connectivity

Description

Computes the vertex connectivity of a network, the minimum number of vertices whose removal disconnects the graph or leaves a single vertex. Higher values indicate a more robust structure.

Usage

```
network_vertex_connectivity(x, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Value

Numeric scalar: the minimum vertex cut size, or NA when igraph cannot compute it.

Examples

```
network_vertex_connectivity(regulation_net)
```

---

network_wrangling *Network Wrangling Verbs*

Description

cograph provides verbs for reshaping a network. Each verb accepts any supported input (matrix, edge list, igraph, statnet network, tna model, cograph_network) and takes its options as named arguments. The result is a cograph_network, or the input format when keep_format = TRUE. as.data.frame() returns the edge or node table of a result.

Value

Each verb returns a cograph_network, except split_components(), which returns a list of them. With keep_format = TRUE a matrix, igraph, statnet network or tna input comes back in that format. An edge-list data frame comes back as a cograph_network with a warning.

Selecting

filter_nodes(), select_nodes() Keep nodes by expression, name, index, top-N, neighborhood or component.

filter_edges(), select_edges() Keep edges by expression, endpoints, bridges, mutuality or top-N.

select_neighbors(), select_component(), select_top(), select_k_core() Named shorthands for the common selections.

split_components() One network per connected component.

Weights

threshold_edges() Keep edges by weight, count, proportion or density.

binarize() Replace weights with 0/1.

symmetrize() Combine opposite arcs into one edge.

normalize_weights() Rescale by row, column, maximum, total, or to [0, 1].

invert_weights() Turn similarities into distances.

Structure

`to_undirected()`, `to_directed()`, `reverse_edges()` Change directedness.

`remove_isolates()` Drop nodes with no edges.

`contract_nodes()` Collapse groups of nodes into one.

`spanning_tree()`, `complement_network()` Derived graphs.

`reorder_nodes()`, `rename_nodes()` Change node order or labels without changing the network.

`simplify()` Merge duplicate edges and drop loops. It returns the input format.

Editing

`add_nodes()`, `remove_nodes()`, `add_edges()`, `remove_edges()` Add and remove.

`mutate_nodes()`, `mutate_edges()` Compute and store attributes.

`bind_networks()` Union, intersection or difference of two networks.

Conversion and access

`as_cograph()`, `to_matrix()`, `to_igraph()`, `to_network()`, `to_df()`, and `as.data.frame()` on a `cograph_network` (see `as.data.frame.cograph_network`).

Semantics

- Filtering edges does not remove nodes, as in `igraph::delete_edges()` and tidygraph. Nodes left without edges raise a `cograph_isolates_created` warning. They are dropped by `remove_isolates()` or by `keep_isolates = FALSE` in the verbs that have that argument.
- The weight matrix of an undirected result is symmetric, so the result is still detected as undirected downstream.
- Node groups, estimation data, layout coordinates and the original source type are carried through the verbs.
- Unknown node names, out-of-range or fractional indices, unknown measure names and a malformed `between` raise a `cograph_bad_selection` error. The `name` argument of `select_nodes()` is an exception. Unknown names in it are skipped, and a warning is given when no node matches.

Related verbs elsewhere

`ego_networks()`, `shortest_paths()`, `disparity_filter()`, `detect_communities()`, `summarize_clusters()`, `aggregate_layers()`.

Examples

```
as.data.frame(threshold_edges(regulation_net, minimum = 0.2))
```

`normalize_weights` *Normalize Edge Weights*

Description

Rescales the edge weights. Row normalization turns a transition count matrix into the transition probabilities used by TNA models.

Usage

```
normalize_weights(
  x,
  method = c("row", "column", "max", "sum", "minmax"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`method`	How to rescale:
	`"row"` (default) each row sums to 1
	`"column"` each column sums to 1
	`"max"` divide by the largest absolute weight
	`"sum"` divide by the sum of the weight matrix, in which each undirected edge appears twice
	`"minmax"` rescale the non-zero weights to [0, 1]
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Details

A row or column whose total is zero is left at zero. Zero totals, and a zero denominator for `"max"` or `"sum"`, raise a `cograph_zero_norm` warning.

`"minmax"` maps the weakest edge to `.Machine$double.eps`. A weight of exactly 0 would remove the edge, because 0 stores "no edge". When all weights are equal they all become 1.

`"max"`, `"sum"` and `"minmax"` rescale each edge independently and keep any extra edge columns. `"row"` and `"column"` scale an edge by a total that differs at its two endpoints. They break symmetry, so an undirected input is returned as a directed network.

Value

A `cograph_network` with rescaled weights, or the input format when `keep_format = TRUE`.

See Also

binarize, invert_weights, symmetrize

Examples

```
normalize_weights(regulation_net, method = "row")
```

---

overlay_communities *Overlay Community Blobs on a Network Plot*

Description

Plots a network with splot and overlays smooth blob shapes that mark node communities.

Usage

```
overlay_communities(
  x,
  communities,
  blob_colors = NULL,
  blob_alpha = 0.25,
  blob_linewidth = 0.7,
  blob_line_alpha = 0.8,
  ...
)
```

Arguments

x	A network object passed to splot: tna, matrix, igraph, or cograph_network.
communities	Community assignments in any of these formats: • the name of an igraph cluster_* algorithm (e.g., "walktrap", "louvain", "leiden", "edge_betweenness"), matched partially; directed networks are collapsed to undirected before detection; • a numeric or factor membership vector in node order (e.g., c(1, 1, 2, 2, 3)), or named by node; • a named list of character vectors of node names; • a cograph_communities, igraph communities or tna_communities object.
blob_colors	Character vector of fill colors for blobs. Recycled if shorter than the number of communities. Default NULL uses the built-in blob palette.
blob_alpha	Numeric. Fill transparency (0-1). Default 0.25.
blob_linewidth	Numeric. Border line width. Default 0.7.
blob_line_alpha	Numeric. Border line transparency (0-1). Default 0.8.
...	Additional arguments passed to splot.

Value

Invisibly, the cograph_network object returned by splot. Called for the side effect of plotting.

Examples

```
comm <- cograph::communities(regulation_net, method = "walktrap")
overlay_communities(regulation_net, comm)
```

---

palettes *Color Palettes*

Description

The palette_*() functions generate a vector of n colors for nodes or edges. They are registered under their short names (for example "colorblind"), which sn_palette accepts, and list_palettes() returns the registered names.

palette_rainbow() Rainbow hues.

palette_colorblind() The colorblind-safe colors of Wong.

palette_pastel() Soft pastel colors.

palette_viridis() The viridis family, chosen by option.

palette_blues(), palette_reds() Sequential blue or red shades.

palette_diverging() Blue to red through midpoint.

Usage

```
list_palettes()

palette_rainbow(n, alpha = 1)

palette_colorblind(n, alpha = 1)

palette_pastel(n, alpha = 1)

palette_viridis(n, alpha = 1, option = "viridis")

palette_blues(n, alpha = 1)

palette_reds(n, alpha = 1)

palette_diverging(n, alpha = 1, midpoint = "white")
```

Arguments

n	Number of colors to generate.
alpha	Transparency, from 0 (transparent) to 1 (opaque).
option	Viridis option, one of "viridis", "magma", "plasma", "inferno", "cividis". Any other value gives the "viridis" colors.
midpoint	Color of the midpoint of the diverging palette.

Value

The palette_*() functions return a character vector of n colors. list_palettes() returns a character vector of palette names.

Examples

```
splot(regulation_net, node_fill = palette_colorblind(n = 10))
```

---

panel_layout *Configure a custom multi-panel layout*

Description

Sets up a multi-panel device layout for use with cograph plotting functions called with combined = FALSE. The previous par() settings are returned so that the caller can restore the device state.

Usage

```
panel_layout(spec, mar = c(2, 2, 3, 1), widths = NULL, heights = NULL)
```

Arguments

spec	Either a length-2 vector of positive integers c(nrow, ncol) for a uniform grid, or a numeric matrix of non-negative panel numbers with at least one positive cell, passed to graphics::layout().
mar	Numeric vector of length 4 giving panel margins. Default c(2, 2, 3, 1).
widths, heights	Optional numeric vectors of column widths and row heights, passed to graphics::layout(). They are valid only when spec is a matrix. Supplying them with a length-2 spec is an error.

Details

A length-2 spec = c(nrow, ncol) creates a uniform grid through graphics::par(mfrow = ...). A matrix spec creates a non-uniform layout through graphics::layout(). The matrix values number the panel cells and are read in column-major order, so matrix(c(1, 1, 2, 3), 2, 2) gives one tall cell in the left column and two stacked cells in the right column.

Value

Invisibly, a list of the previous `par()` settings (`mar` and `mfrow`). Passing it to `graphics::par()` restores the prior device state and also clears a layout set by `graphics::layout()`.

Combined-flag scope

The `combined = FALSE` argument applies to the multi-panel plot functions `plot_netobject_group()`, `plot_netobject_ml()`, `plot_net_bootstrap_group()`, `plot_group_permutation()`, `plot_difference()`, `splot.net_mlvar(type = "all")`, `plot_network_evolution()`, `plot.cograph_motifs(type = "network")`, `plot.cograph_motif_result(type = "patterns")`, `plot.cograph_motif_analysis(type = "patterns")`, `plot.tna_disparity(type = "comparison")`, and `splot()` on `group_tna` and other list inputs. A single-network `splot()` call plots one panel and ignores `combined`.

Examples

```
op <- panel_layout(c(1, 2))
splot(regulation_net, combined = FALSE)
splot(regulation_net, layout = "circle", combined = FALSE)
graphics::par(op)
```

---

plot-results *Plot Analysis Results*

Description

Plot methods and plotting functions for the result objects returned by the analysis functions of cograph and by the tna and Nestimate packages. Each function accepts one class of result.

`plot()` **or** `plot_motifs()` **on a** `cograph_motif_result` From `motifs()` or `subgraphs()`. Plots triad diagrams, MAN type frequencies, z-scores or MAN pattern diagrams.

`plot()` **on a** `cograph_motif_analysis` From `extract_motifs()`. Plots the same four views as above.

`plot()` **on a** `cograph_motifs` From `motif_census()`. Plots motif z-scores colored by significance direction, a z-score heatmap or diagrams of the selected motifs.

`plot()` **on a** `cograph_communities` From `communities()`. Plots the network with nodes grouped by community through `splot()`.

`plot()` **on a** `cograph_core_periphery` From `core_periphery()`. Plots the network with core nodes enlarged and periphery nodes reduced.

`plot()` **on a** `cograph_rich_club` From `rich_club()`. Plots the rich club curve with the null model band, or the club members on the network.

`plot()` **on a** `cograph_degree_fit` From `fit_degree_distribution()`. Plots a histogram of the observed degrees with the fitted distribution curves overlaid.

`plot()` **on a** `cograph_vulnerability` From `vulnerability()`. Plots the node vulnerability scores as a bar chart.

`plot()` **and** `splot()` **on a** `tna_disparity` From `disparity_filter()`. `plot()` plots the backbone or the original and backbone networks side by side. `splot()` plots the full network with backbone edges solid and the remaining edges dashed and faded.

`splot()` **on a** `tna_bootstrap` From `tna::bootstrap()`. Plots the network with significant and non-significant edges styled differently.

`plot_permutation()` **or** `splot()` **on a** `tna_permutation` From `tna::permutation_test()`. Plots the edge differences between two networks, colored by sign and styled by significance.

`plot_group_permutation()` **or** `splot()` **on a** `group_tna_permutation` From `tna::permutation_test()` on a `group_tna` model. Plots one panel per pairwise comparison.

`plot_netobject_group()` **or** `plot()` **on a** `netobject_group` A named list of Nestimate networks. Plots one panel per group.

`plot_net_bootstrap_group()` **or** `plot()` **on a** `net_bootstrap_group` A list of Nestimate `net_bootstrap` results. Plots one panel per group with significance styling.

`plot_netobject_ml()` **or** `plot()` **on a** `netobject_ml` A multilevel Nestimate network. Plots the between-person and within-person networks side by side.

`plot_net_stability()` **on a** `net_stability` From `Nestimate::centrality_stability()`. Plots the mean correlation of each centrality measure with the original against the proportion of cases dropped.

Usage

```
## S3 method for class 'cograph_communities'
plot(x, network = NULL, ...)

## S3 method for class 'cograph_core_periphery'
plot(
  x,
  core_color = "#E41A1C",
  periphery_color = "#377EB8",
  core_size = 12,
  periphery_size = 6,
  ...
)

## S3 method for class 'tna_disparity'
plot(x, type = c("backbone", "comparison"), combined = TRUE, ...)

splot.tna_disparity(
  x,
  show = c("styled", "backbone", "full"),
  edge_style_sig = 1,
  edge_style_nonsig = 2,
  alpha_nonsig = 0.3,
  ...
)

## S3 method for class 'cograph_degree_fit'
```

```
plot(
  x,
  which = NULL,
  log = "",
  cols = NULL,
  lwd = 2,
  main = "Degree Distribution Fit",
  ...
)

## S3 method for class 'cograph_motif_result'
plot(
  x,
  type = c("triads", "types", "significance", "patterns"),
  n = 15,
  ncol = 5,
  colors = c("#2166AC", "#B2182B"),
  node_size = 5,
  label_size = 11,
  title_size = 12,
  stats_size = 13,
  legend_size = 13,
  legend = TRUE,
  motif_color = "#800020",
  spacing = 1,
  base_size = 12,
  combined = TRUE,
  ...
)

plot_motifs(
  x,
  type = c("triads", "types", "significance", "patterns"),
  n = 15,
  ncol = 5,
  colors = c("#2166AC", "#B2182B"),
  node_size = 5,
  label_size = 11,
  title_size = 12,
  stats_size = 13,
  legend_size = 13,
  legend = TRUE,
  motif_color = "#800020",
  spacing = 1,
  base_size = 12,
  ...
)
```

```
## S3 method for class 'cograph_motif_analysis'
plot(
  x,
  type = c("triads", "types", "significance", "patterns"),
  n = 20,
  colors = c("#2166AC", "#B2182B"),
  res = 72,
  node_size = 5,
  label_size = 7,
  title_size = 7,
  stats_size = 5,
  ncol = 5,
  legend = TRUE,
  color = "#800020",
  spacing = 1,
  combined = TRUE,
  ...
)

## S3 method for class 'cograph_motifs'
plot(
  x,
  type = c("bar", "heatmap", "network"),
  show_nonsig = FALSE,
  top_n = NULL,
  colors = c("#2166AC", "#F7F7F7", "#B2182B"),
  combined = TRUE,
  ...
)

splot.tna_bootstrap(
  x,
  display = c("styled", "significant", "full", "ci"),
  edge_style_sig = 1,
  edge_style_nonsig = 2,
  color_nonsig = "#888888",
  show_ci = FALSE,
  show_stars = TRUE,
  width_by = NULL,
  inherit_style = TRUE,
  ...
)

plot_netobject_group(
  x,
  nrow = NULL,
  ncol = NULL,
  common_scale = TRUE,
```

```
  title_prefix = NULL,
  combined = TRUE,
  ...
)

plot_netobject_ml(
  x,
  layout = NULL,
  common_scale = TRUE,
  titles = c("Between-person", "Within-person"),
  combined = TRUE,
  ...
)

plot_net_bootstrap_group(
  x,
  nrow = NULL,
  ncol = NULL,
  common_scale = TRUE,
  combined = TRUE,
  ...
)

plot_net_stability(x, ...)

splot.tna_permutation(x, ...)

splot.group_tna_permutation(x, ...)

plot_permutation(
  x,
  show_nonsig = FALSE,
  edge_positive_color = "#009900",
  edge_negative_color = "#C62828",
  edge_nonsig_color = "#888888",
  edge_nonsig_style = 2,
  show_stars = TRUE,
  show_effect = FALSE,
  edge_nonsig_alpha = 0.4,
  ...
)

plot_group_permutation(x, i = NULL, combined = TRUE, ...)

## S3 method for class 'cograph_rich_club'
plot(x, type = c("curve", "network"), k = NULL, col = "#E41A1C", ...)

## S3 method for class 'cograph_vulnerability'
```

```
plot(x, top = NULL, col = "steelblue", ...)
```

Arguments

x	The result object. The Description lists the class each function accepts.
network	The network the communities were detected on. It is required only when the result does not store the network.
...	Additional arguments passed to the underlying plotting call. Network plots pass them to splot(); plot_group_permutation() passes them to plot_permutation() and plot_net_bootstrap_group() to the splot() method for net_bootstrap (for example display = "significant"). The rich club curve passes them to plot, the degree fit to hist, the vulnerability plot to barplot, plot_net_stability() to plot, and cograph_motifs with type = "network" to the per-motif igraph plot calls. The ggplot2 motif views do not use them.
core_color, periphery_color	Node colors of core and periphery nodes.
core_size, periphery_size	Node sizes of core and periphery nodes.
type	Plot type. The values for each class are listed in Details.
combined	Logical. When TRUE (default), a multi-panel plot is arranged in an internal grid through graphics::par(mfrow = ...). When FALSE, the panels are plotted into a layout the caller has already set up, for example with panel_layout(). It applies to type = "network" for cograph_motifs, to type = "patterns" (and type = "triads" on census results) for the other motif results, to type = "comparison" for tna_disparity, to plot_group_permutation() when i is NULL, and to the group and multilevel panel functions.
show	Network shown by splot() on a tna_disparity. "styled" (default) shows the full network with backbone styling, "backbone" the backbone only and "full" the full network without styling.
edge_style_sig	Line type of significant (or backbone) edges. Default 1 (solid).
edge_style_nonsig	Line type of non-significant (or non-backbone) edges. Default 2 (dashed).
alpha_nonsig	Transparency of non-backbone edges. Default 0.3.
which	Character vector of fitted distributions to show. NULL (default) shows all fitted distributions.
log	Log-scale axes for the degree fit, one of "" (default), "x", "y" or "xy". Only "y" and "xy" set a logarithmic histogram axis. The values containing "x" only remove non-positive fitted curve values.
cols	Colors of the fitted distribution curves, named or unnamed. NULL uses a built-in palette.
lwd	Line width of the fitted distribution curves.
main	Title of the degree-fit plot.
n	Maximum number of triads, patterns or z-score bars plotted. For cograph_motif_analysis with type = "significance", the n lowest and n highest z-scores are plotted.

`ncol`, `nrow` Number of columns and rows of the panel grid. A `NULL` value is computed from the number of panels.

`colors` Colors of the significance scale in motif plots. For `cograph_motif_result` and `cograph_motif_analysis`, a vector of two colors. The first fills items that are significantly under-represented (`p < .05` and `z < 0`) and the second fills items that are significantly over-represented (`p < .05` and `z > 0`). All other items are filled neutral grey (`"#9E9E9E"`). When no per-type significance is available, the first color is used as a single fill. For `cograph_motifs`, a vector of three colors for under-represented, neutral and over-represented motifs.

`node_size` Relative size of the nodes in triad diagrams.

`label_size` Font size of the node labels in triad diagrams.

`title_size` Font size of the panel titles in triad diagrams.

`stats_size` Font size of the statistics caption of each triad panel (for example `n=34 z=-55.3 p<.001`).

`legend_size` Font size of the legend below the triad grid.

`legend` Logical. Whether to show the legend of node-label abbreviations below the triad grid.

`motif_color`, `color` Color of the nodes, edges and labels in triad diagrams. `motif_color` applies to `cograph_motif_result` and `color` to `cograph_motif_analysis`.

`spacing` Spacing multiplier for triad diagrams. Values above 1 pull the three nodes of each panel inward and values below 1 push them apart.

`base_size` Base font size of the ggplot2 theme used by `type = "types"` and `type = "significance"`.

`res` Unused. It is kept for backward compatibility.

`show_nonsig` Logical. Whether non-significant items are shown. For `cograph_motifs` these are motifs; for `plot_permutation()` they are edges, plotted dashed and grey. Default `FALSE`.

`top_n` Number of motifs with the largest absolute z-scores to plot. `NULL` (default) plots all.

`display` Display mode of `splot()` on a `tna_bootstrap`. `"styled"` (default) shows all edges with significance styling, `"significant"` the significant edges only, `"full"` all edges without significance styling and `"ci"` all edges with confidence interval bounds in the labels and an underlay whose width reflects the interval width relative to the edge weight.

`color_nonsig` Accepted for compatibility. The styled bootstrap plot uses a fixed pink color for non-significant edges.

`show_ci` Logical. Whether confidence interval bounds are added to the edge labels. `display = "ci"` adds them as well.

`show_stars` Logical. Whether significance stars (`*`, `**`, `***`) are added to the edge labels.

`width_by` Set to `"cr_lower"` to plot the lower bounds of the consistency range as the edge weights, with widths scaled by these bounds and the significance styling removed. `NULL` (default) leaves the edges unchanged.

`inherit_style` Logical. Whether the labels, node colors and initial-state donuts of the original TNA model are reused, with the oval layout as the default layout.

`common_scale` Logical. Whether all panels share the same maximum edge weight. Default `TRUE`.

`title_prefix` Optional text placed before each group name in the panel titles.

`layout` Layout algorithm of the multilevel panels. `NULL` (default) uses `"oval"`.

`titles` Character vector of length 2 with the titles of the between-person and within-person panels.

`edge_positive_color`, `edge_negative_color`
Colors of significant positive (`x > y`) and negative (`x < y`) edge differences.

`edge_nonsig_color`, `edge_nonsig_style`, `edge_nonsig_alpha`
Color, line type and transparency of non-significant edge differences.

`show_effect` Logical. Whether the absolute effect size is added in parentheses to the labels of significant edges.

`i` Index or name of a single comparison to plot. `NULL` (default) plots all comparisons.

`k` Prominence threshold whose club members are highlighted with `type = "network"` for `cograph_rich_club`. `NULL` uses the threshold with the highest `phi_norm` (or `phi` when the result is not normalized).

`col` Color of the rich club curve and club members, or of the vulnerability bars.

`top` Number of most vulnerable nodes to plot. `NULL` (default) plots all.

Details

Plot types: For `cograph_motif_result` and `cograph_motif_analysis`, `type` is one of the following values.

`"triads"` (default) Network diagrams of node triples arranged in a grid. A census result without named nodes falls back to `"patterns"`. Each diagram shows a canonical representative of the MAN class, so the node labels identify the participating nodes and their positions do not encode observed source or sink roles. Panel titles read `"<MAN code>: <description>"`, and the caption gives the count and, when significance was tested, the z-score and p-value.

`"types"` Bar chart of MAN type frequencies. For a census tested for significance the bars are colored by significance direction. Instance results and `cograph_motif_analysis` use a single fill, because per-type significance would require aggregating several node-triple rows of the same type.

`"significance"` Z-score bars, one per MAN type for a census and one per node triple for instance results. It requires the analysis to have been run with `significance = TRUE`.

`"patterns"` Abstract MAN pattern diagrams of each triad type. For a census tested for significance the nodes are filled by significance direction, and the panel titles add the z-score and a significance star (* p<.05, ** p<.01, *** p<.001). Instance results use a single fill.

For `cograph_motifs`, `type` is `"bar"` (default; motif z-scores colored by over- or under-representation), `"heatmap"` (z-scores across motif types, labelled with the observed and expected counts) or `"network"` (one diagram per motif that passes the `show_nonsig` and `top_n` filters). The network view requires a directed 3-node census and otherwise falls back to the bar chart with a

message. For cograph_rich_club, type is "curve" (default; the coefficient across thresholds with null model bands) or "network" (club members at threshold k). For tna_disparity, type is "backbone" (default) or "comparison" (original and backbone side by side).

Bootstrap and permutation input: splot() on a tna_bootstrap reads the original weights from weights (or weights_orig), the significant weights from weights_sig or the p_values matrix, the confidence bounds from ci_lower and ci_upper, the significance level from a level element and the styling from model. Results of tna::bootstrap() store no level element, so their edges are styled at a level of 0.05. In styled mode significant edges are solid dark blue with bold starred labels and are plotted on top, and non-significant edges are dashed pink with plain labels.

plot_permutation() reads the edge differences (x - y) from edges$diffs_true, the significant differences from edges$diffs_sig and the edge statistics from edges$stats. Significant positive differences are solid green and significant negative differences solid red, both with bold starred labels.

plot_net_bootstrap_group() plots each group through the splot() method for net_bootstrap, so every panel keeps the solid and dashed significance styling.

Value

Each function is called for its plot. The returned value depends on the class.

Motif results A ggplot2 object, printed and returned invisibly, for type = "types", "significance", "bar" and "heatmap". type = "triads" and "patterns" return the input invisibly for cograph_motif_result and NULL invisibly for cograph_motif_analysis. cograph_motifs with type = "network" returns NULL invisibly. Any cograph_motifs plot returns NULL invisibly with a message when no motif passes the show_nonsig and top_n filters.

Network plots The cograph_network built by splot(), invisibly, for cograph_communities, tna_disparity, tna_bootstrap and tna_permutation. plot_permutation() returns NULL invisibly with a message when no edge remains to plot. plot_group_permutation() returns the selected panel's network when i is given and NULL invisibly otherwise.

Group panels The input invisibly for netobject_group and net_bootstrap_group. With a single group the network of that panel is returned, and with no groups NULL.

Other results The input invisibly for cograph_core_periphery, cograph_rich_club, cograph_vulnerability, netobject_ml and net_stability, and NULL invisibly for cograph_degree_fit.

See Also

splot() for the network plots of single networks.

Examples

```
census <- motifs(regulation_net, significance = FALSE)
plot(census, type = "types")
```

plot_alluvial *Plot Alluvial Diagram*

Description

Creates an alluvial (Sankey) diagram of aggregated flows between states. It calls `plot_transitions()` with `track_individuals = FALSE`.

Usage

```
plot_alluvial(
  x,
  from_title = "From",
  to_title = "To",
  title = NULL,
  from_colors = NULL,
  to_colors = NULL,
  flow_fill = "#888888",
  flow_alpha = 0.4,
  flow_color_by = NULL,
  flow_border = NA,
  flow_border_width = 0.5,
  node_width = 0.08,
  node_border = NA,
  node_spacing = 0.02,
  label_size = 3.5,
  label_position = c("beside", "inside", "above", "below", "outside"),
  label_halo = TRUE,
  label_color = "black",
  label_fontface = "plain",
  label_nudge = 0.02,
  title_size = 5,
  title_color = "black",
  title_fontface = "bold",
  curve_strength = 0.6,
  show_values = FALSE,
  value_position = c("center", "origin", "destination", "outside_origin",
    "outside_destination"),
  value_size = 3,
  value_color = "black",
  value_halo = NULL,
  value_fontface = "bold",
  value_nudge = 0.03,
  value_min = 0,
  show_totals = FALSE,
  total_size = 4,
  total_color = "white",
```

```
  total_fontface = "bold",
  conserve_flow = TRUE,
  min_flow = 0,
  threshold = 0,
  value_digits = 2,
  column_gap = 1
)
```

Arguments

x	Input data in one of these formats: • A transition matrix (rows = from, columns = to, values = counts). • A vector of "before" states, with the vector of "after" states passed as the second argument (`from_title`). Both vectors must have the same length, greater than 2, and their contingency table is computed. • A data frame with two columns of raw observations, whose contingency table is computed. • A data frame with three or more columns of raw observations, one column per time point, plotted as a multi-step diagram. • A data frame with columns `from`, `to` and `count`. • A list of matrices for multi-step transitions. • A `tna` object. Its sequence data are used as a data frame of time points (rows with missing values are dropped), or its weight matrix when no sequence data are stored.
from_title	Title for the left column. Default "From". For multi-step and individual-tracking plots, a vector with one title per column. Data frame input then uses the column names by default, and a vector shorter than the number of columns is replaced by "T1", "T2", ...
to_title	Title for the right column. Default "To". Ignored for multi-step plots.
title	Optional plot title.
from_colors	Colors for the left-side nodes. In multi-step and individual-tracking plots, the colors of all states. Default NULL uses the built-in palette.
to_colors	Colors for the right-side nodes in two-column plots. Default NULL uses the built-in palette.
flow_fill	Fill color for flows. Default "#888888" (grey). In multi-step plots it is replaced by the state colors when `flow_color_by` is set. Individual-tracking lines do not use it.
flow_alpha	Alpha transparency for flows. Default 0.4.
flow_color_by	Color flows by state. Multi-step aggregate plots accept `"source"` or `"destination"`. Individual-tracking plots also accept `"first"` and `"last"`. Default NULL uses `flow_fill`. Two-column aggregate plots ignore this argument.
flow_border	Border color for flows. Default NA (no border).
flow_border_width	Line width for flow borders. Default 0.5.

node_width	Width of node rectangles (0-1 scale). Default 0.08.
node_border	Border color for nodes. Default NA (no border).
node_spacing	Vertical spacing between nodes (0-1 scale). Default 0.02.
label_size	Size of node labels. Default 3.5.
label_position	Position of node labels: "beside" (default), "inside", "above", "below", "outside". In multi-step and individual-tracking plots, "beside" and "outside" label the first and last columns only. See mid_label_position for middle columns.
label_halo	Logical: add a white halo around labels and column titles? Default TRUE.
label_color	Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots. Two-column aggregate plots use black external labels and white inside labels.
label_fontface	Font face of state name labels ("plain", "bold", "italic", "bold.italic"). Default "plain". Applied to multi-step and individual-tracking plots.
label_nudge	Distance between node edge and label (in plot units). Default 0.02. Used by multi-step and individual-tracking plots.
title_size	Size of column titles. Default 5.
title_color	Color of column title text. Default "black". Applied to multi-step and individual-tracking plots.
title_fontface	Font face of column titles. Default "bold". Applied to multi-step and individual-tracking plots.
curve_strength	Controls bezier curve shape (0-1). Default 0.6.
show_values	Logical: show transition counts on flows? Default FALSE.
value_position	Position of flow values: "center", "origin", "destination", "outside_origin", "outside_destination". Default "center". Individual-tracking plots use "center", "origin" and "destination".
value_size	Size of value labels on flows. Default 3.
value_color	Color of value labels. Default "black".
value_halo	Logical: add halo around flow value labels? Default NULL uses label_halo. Applied to multi-step and individual-tracking plots.
value_fontface	Font face of flow value labels. Default "bold". Applied to multi-step and individual-tracking plots.
value_nudge	Distance of value labels from node edge when using "origin" or "destination" positions. Default 0.03.
value_min	Minimum count to show a flow value label in multi-step and individual-tracking plots. Default 0 (show all). Two-column aggregate plots show every nonzero value label when show_values = TRUE.
show_totals	Logical: show total counts on nodes? Default FALSE.
total_size	Size of total labels. Default 4.
total_color	Color of total labels. Default "white".
total_fontface	Font face of total labels. Default "bold". Applied to multi-step and individual-tracking plots.

conserve_flow	Logical. When TRUE (default), node heights on both sides of a two-column plot are proportions of the same total flow. When FALSE, each side is scaled to its own total. Ignored for multi-step and individual-tracking plots.
min_flow	Minimum flow value to display in aggregate plots. Default 0 (show all).
threshold	Minimum flow value to display in aggregate plots. Flows below max(threshold, min_flow) are removed. Default 0.
value_digits	Number of decimal places for flow value labels and node totals. Default 2.
column_gap	Horizontal spread of columns (0-1) for multi-step and individual-tracking plots. Default 1 uses the full width. Smaller values (e.g., 0.6) bring the columns closer together.

Value

A ggplot object.

See Also

plot_transitions, plot_trajectories

Examples

```
plot_alluvial(regulation_net)
```

---

plot_bootstrap_forest *Forest Plot for Bootstrap Network Results*

Description

Plots bootstrap results of net_bootstrap, net_bootstrap_group, tna_bootstrap and boot_glasso objects as a ggplot2 forest plot. Each row is one non-zero network edge. A square marks the point estimate, a horizontal bar spans the selected interval, and a dashed reference line marks zero. Significant edges are plotted in sig_color and non-significant edges in a faded nonsig_color.

Usage

```
plot_bootstrap_forest(x, ...)

## S3 method for class 'net_bootstrap'
plot_bootstrap_forest(
  x,
  alpha = NULL,
  layout = c("linear", "circular", "grouped"),
  interval = c("ci", "cr", "both"),
  show_nonsig = TRUE,
  sort_by = c("estimate", "significance", "name"),
```

```
  n_top = NULL,
  node_colors = NULL,
  sig_color = "#2C6E8A",
  cr_color = "#D4829A",
  nonsig_color = "#CCCCCC",
  ring_color = "#C8C8C8",
  median_color = "#AAAAAA",
  label_size = NULL,
  label_color = NULL,
  point_size = NULL,
  r_inner = NULL,
  r_outer = NULL,
  gap_rad = NULL,
  label_offset = NULL,
  src_label_size = NULL,
  margins = c(0.1, 0.1, 0.1, 0.1),
  scale = 1,
  title = NULL,
  subtitle = NULL,
  ...
)

## S3 method for class 'tna_bootstrap'
plot_bootstrap_forest(
  x,
  alpha = NULL,
  layout = c("linear", "circular", "grouped"),
  interval = c("ci", "cr", "both"),
  show_nonsig = TRUE,
  sort_by = c("estimate", "significance", "name"),
  n_top = NULL,
  node_colors = NULL,
  sig_color = "#2C6E8A",
  cr_color = "#D4829A",
  nonsig_color = "#CCCCCC",
  ring_color = "#C8C8C8",
  median_color = "#AAAAAA",
  label_size = NULL,
  label_color = NULL,
  point_size = NULL,
  r_inner = NULL,
  r_outer = NULL,
  gap_rad = NULL,
  label_offset = NULL,
  src_label_size = NULL,
  margins = c(0.1, 0.1, 0.1, 0.1),
  scale = 1,
  title = NULL,
```

```
  subtitle = NULL,
  ...
)

## S3 method for class 'boot_glasso'
plot_bootstrap_forest(
  x,
  alpha = NULL,
  layout = c("linear", "circular", "grouped"),
  interval = c("ci", "cr", "both"),
  show_nonsig = TRUE,
  sort_by = c("estimate", "significance", "name"),
  n_top = NULL,
  node_colors = NULL,
  sig_color = "#2C6E8A",
  cr_color = "#D4829A",
  nonsig_color = "#CCCCCC",
  ring_color = "#C8C8C8",
  median_color = "#AAAAAA",
  label_size = NULL,
  label_color = NULL,
  point_size = NULL,
  r_inner = NULL,
  r_outer = NULL,
  gap_rad = NULL,
  label_offset = NULL,
  src_label_size = NULL,
  margins = c(0.1, 0.1, 0.1, 0.1),
  scale = 1,
  title = NULL,
  subtitle = NULL,
  ...
)

## S3 method for class 'net_bootstrap_group'
plot_bootstrap_forest(
  x,
  layout = c("linear", "circular"),
  interval = c("ci", "cr", "both"),
  show_nonsig = TRUE,
  n_top = NULL,
  all_edges = FALSE,
  pos_color = NULL,
  title = NULL,
  subtitle = NULL,
  label_size = 2.8,
  ...
)
```

Arguments

x	A tna_bootstrap (from tna::bootstrap), net_bootstrap, net_bootstrap_group, or boot_glasso object.
...	Currently unused.
alpha	Significance threshold. Default NULL, which inherits from the object: $ci_level for net_bootstrap, $level for tna_bootstrap, $alpha for boot_glasso, each falling back to 0.05.
layout	"linear" (default) gives the standard forest plot; "circular" arranges each edge as a spoke around a circle, with the inner ring at the smallest lower bound and the outer ring just beyond the largest upper bound; "grouped" arranges edges in sectors by source node. The net_bootstrap_group method supports "linear" and "circular" only.
interval	Which interval to display: "ci" (bootstrap confidence interval, default), "cr" (consistency range, stability inference only), or "both" (CI as outer bar, CR as inner bar).
show_nonsig	Logical. Include non-significant edges. Default TRUE.
sort_by	Order of edges on the y-axis in the linear layout: "estimate" (default, ascending), "significance" (most significant at top), or "name" (alphabetical). When n_top is set, the retained edges are ordered by estimate. The circular layout orders edges alphabetically, clockwise from the top.
n_top	Integer. Keeps only the n_top edges with the largest absolute estimate. Applied after significance filtering. Default NULL.
node_colors	Optional named vector of node colors for the grouped layout. When NULL, node colors stored in the original network or tna model are used if present.
sig_color	Color for significant CI bars and points (linear and circular layouts). Default "#2C6E8A" (teal-blue).
cr_color	Color for the consistency range bar (interval = "cr" or "both"). Default "#D4829A".
nonsig_color	Color for non-significant edges (linear and circular layouts). Default "#CCCCCC".
ring_color	Color for the reference rings (circular and grouped layouts). Default "#C8C8C8".
median_color	Color for the dashed median ring (circular and grouped layouts). Default "#AAAAAA".
label_size	Text size for edge labels (circular and grouped layouts). Default NULL, which gives 2.9 in the circular layout and automatic sizing in the grouped layout. The net_bootstrap_group method defaults to 2.8.
label_color	Fixed color for edge labels (circular and grouped layouts). NULL (default) uses the edge color.
point_size	Size of the estimate square. Default NULL, which gives 3 (linear), 2 (circular), or automatic sizing (grouped).
r_inner	Inner ring radius (grouped layout). Default NULL (auto).
r_outer	Outer ring radius (grouped layout). Default NULL (auto).
gap_rad	Gap in radians between sectors (grouped layout). Default NULL (auto).
label_offset	Distance between outer ring and labels (grouped layout). Default NULL (auto).

src_label_size	Text size for source node labels in the center (grouped layout). Default NULL (auto, label_size * 0.80).
margins	Margins as c(bottom, left, top, right) fractions (grouped layout). Default c(0.1, 0.1, 0.1, 0.1).
scale	Scaling factor applied to all text and point sizes (grouped layout). Default 1. Use values > 1 for high-DPI output, < 1 for small devices.
title	Plot title. Default NULL.
subtitle	Plot subtitle. Default NULL.
all_edges	For net_bootstrap_group, show the union of group edges instead of only edges common to all groups. Default FALSE.
pos_color	Currently unused by the net_bootstrap_group method.

Details

Objects from stability inference (net_bootstrap and tna_bootstrap) carry both a bootstrap confidence interval and a consistency range, and interval = "both" overlays the two. When the consistency range is requested but absent, a message is issued and the confidence interval is shown. For boot_glasso, an edge is significant when its inclusion proportion is at least 1 - alpha.

The net_bootstrap_group method plots the groups side by side in the linear layout, with the edges ordered by their mean estimate across groups. Its interval = "both" shows the confidence interval only, and its circular layout shows the first group only.

Value

A ggplot object.

Examples

```
boot <- tna::bootstrap(tna::tna(coding), iter = 50)
plot_bootstrap_forest(boot)
```

---

plot_centrality *Plot Centrality*

Description

Plots one or more centrality measures, one facet per measure. Accepts the data frame from centrality or any network input.

Usage

```
plot_centrality(
  x,
  measures = NULL,
  style = c("line", "bar", "lollipop", "dot"),
  orientation = c("horizontal", "vertical"),
  scale = c("raw", "normalized", "z", "rank"),
  order_by = NULL,
  top_n = NULL,
  highlight = 0L,
  cluster = NULL,
  palette = "cograph",
  ncol = NULL,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

x	Output of centrality, or any network input (matrix, igraph, cograph_network, tna, netobject).
measures	Character vector of measure names. When x is a network and measures is NULL (default), degree, strength, betweenness, closeness and eigenvector are computed. When x is a centrality data frame, NULL keeps all its measure columns.
style	Character: "line" (default), "bar", "lollipop", or "dot".
orientation	Character: "horizontal" (default, nodes on y-axis) or "vertical" (nodes on x-axis).
scale	Character: "raw" (default, native units with a free value axis per facet), "normalized" (min-max scaled to [0, 1] within measure), "z" (standardized within measure), or "rank" (1 = highest value within measure).
order_by	Character. Name of the measure column that sorts the nodes (e.g., "degree_all"), or "alpha" for alphabetical order. Defaults to the first measure. In the "bar", "lollipop" and "dot" styles an unknown name raises an error; in the "line" style it falls back to the first measure.
top_n	Optional integer. Keeps only the top-N nodes by order_by (by the first measure when order_by = "alpha").
highlight	Integer. In the "bar", "lollipop" and "dot" styles, the top-N nodes per measure are plotted in full color and the rest are muted. Default 0 (no highlighting). Ignored by the "line" style.
cluster	Optional cluster assignment of the nodes, given as the name of a column of x, a vector in node order, or a vector named by node. Colors the nodes by cluster in the "bar", "lollipop" and "dot" styles.
palette	Currently unused. Cluster colors are taken from the built-in cograph palette.

ncol	Number of facet columns. Default NULL uses up to three columns for eight or fewer measures and four otherwise.
title	Plot title. Default NULL.
subtitle	Plot subtitle. Default NULL.
...	Passed to centrality when x is a network.

Details

Four styles are available:

"line" Points connected by a line within each measure, with nodes in the order set by order_by.

"bar" Bars of the measure values.

"lollipop" Segments ending in a dot.

"dot" Dots only.

Value

A ggplot object.

Examples

```
plot_centrality(regulation_net)
```

---

plot_centrality_compare

Plot Centrality Comparison

Description

Plots one centrality measure across two or more groups as stacked, faceted, grouped, dumbbell, line, or pyramid charts. The "pyramid" style is a back-to-back horizontal bar chart for exactly two groups. Groups are aligned on the node names they share.

Usage

```
plot_centrality_compare(
  ...,
  measure = NULL,
  style = c("stacked", "facet", "grouped", "dumbbell", "line", "pyramid"),
  group_labels = NULL,
  group_colors = NULL,
  node_colors = NULL,
  sort_by = c("max", "delta", "first", "alpha"),
  top_n = NULL,
  scale = c("raw", "normalized"),
  show_values = TRUE,
```

```
  size_by_value = FALSE,
  size_range = c(2, 9),
  orientation = c("horizontal", "vertical"),
  ncol = NULL,
  title = NULL,
  subtitle = NULL,
  centrality_args = list()
)
```

Arguments

`...`	Two or more centrality data frames (from `centrality`) or network inputs. When every argument is named, the names are used as group labels if `group_labels` is NULL.
`measure`	Character, a single centrality measure to compare. For network inputs, a name such as "strength" also matches a shared column with an "_all", "_in" or "_out" suffix when the match is unique. For centrality data frames, it must be an exact column name. If NULL, the first shared measure is used.
`style`	Character: `"stacked"` (default), `"facet"`, `"grouped"`, `"dumbbell"`, `"line"`, or `"pyramid"` (2 groups only).
`group_labels`	Character vector with one label per group. Default `c("Group 1", "Group 2", ...)`.
`group_colors`	Character vector of colors, one per group. Default cycles through the cograph palette.
`node_colors`	Colors of the nodes in `style = "facet"`. Either a named character vector mapping node name to color, an unnamed vector of colors applied in node order, or the name of a palette (`"cograph"`, `"okabe"`, `"viridis"`). NULL (default) uses the node colors stored in the first network when available and the cograph palette otherwise.
`sort_by`	`"max"` (default) ranks nodes by highest value across groups; `"delta"` by range; `"first"` by first group; `"alpha"` alphabetically.
`top_n`	Show top N nodes (by `sort_by`). Default: all.
`scale`	`"raw"` (default, native values) or `"normalized"` (min-max scaled to [0, 1] within each group).
`show_values`	Logical. Print the value of each bar or point. Default TRUE.
`size_by_value`	Logical. For `"dumbbell"` style, scale dot size by centrality value. Default FALSE.
`size_range`	Numeric vector of length 2 giving the min and max dot size (mm) when `size_by_value = TRUE`. Default `c(2, 9)`.
`orientation`	Character: `"horizontal"` (default, nodes on y-axis) or `"vertical"` (nodes on x-axis). Ignored by the `"pyramid"` style.
`ncol`	Number of facet columns for `style = "facet"`. Default NULL chooses automatically.
`title`	Plot title. NULL (default) gives "Centrality comparison:" followed by the measure name.

subtitle	Plot subtitle. When NULL, the "pyramid" style shows the two group labels and the other styles show none.
centrality_args	Named list of additional arguments passed to centrality when inputs are networks.

Value

A ggplot object.

Examples

```
plot_centrality_compare(Full = regulation_net,
  Strong = threshold_edges(regulation_net, minimum = 0.1), measure = "strength")
```

---

plot_centrality_distribution

Plot Centrality Distribution

Description

Histogram or density plot of one centrality measure. The input is either the data frame returned by centrality or a network, for which the measure is computed first.

Usage

```
plot_centrality_distribution(
  x,
  measure = "degree_all",
  type = c("histogram", "density"),
  normalize = FALSE,
  bins = NULL,
  log = "",
  col = "steelblue",
  border = "white",
  main = NULL,
  xlab = NULL,
  ...
)
```

Arguments

x	A data frame from centrality, or a network input (matrix, igraph, cograph_network, tna).
measure	Character. Column of the centrality output to plot, for example "degree_all" (default) or "strength_in". An unknown name is an error that lists the available columns.

type	Character. "histogram" (default) or "density".
normalize	Logical. Show proportions instead of counts. Default FALSE.
bins	Integer or NULL. Number of equal-width histogram bins. The default NULL uses the Freedman-Diaconis rule.
log	Character. "y" or "xy" log-scales the y-axis. The x-axis is never log-scaled, and any other value gives linear axes. Default "".
col	Fill color (line color for the density). Default "steelblue".
border	Bar border color for the histogram. Default "white".
main	Plot title. The default NULL builds a title such as "Degree Distribution" from the measure name.
xlab	X-axis label. The default NULL uses the measure name.
...	Additional arguments passed to barplot or plot.

Value

Invisibly, a numeric vector of the finite centrality values plotted.

Examples

```
cograph::plot_centrality_distribution(regulation_net, measure = "strength_all")
```

---

plot_centrality_heatmap

Plot Centrality Heatmap

Description

Plots a heatmap of nodes (rows) by centrality measures (columns). Cell fill is the z-score of each value within its measure, mapped to a diverging color scale. Optional row clustering places nodes with similar centrality profiles next to each other.

Usage

```
plot_centrality_heatmap(
  x,
  measures = NULL,
  cluster_rows = TRUE,
  order_by = NULL,
  show_values = FALSE,
  value_digits = 1L,
  low = "#2171B5",
  mid = "white",
  high = "#CB181D",
  limits = c(-2.5, 2.5),
  title = NULL,
```

```
  subtitle = "z-scored within measure",
  ...
)
```

Arguments

x	Centrality data frame (from centrality) or a network input.
measures	Character vector of measure names. When x is a network and measures is NULL (default), degree, strength, betweenness, closeness and eigenvector are computed. When x is a centrality data frame, NULL keeps all its measure columns.
cluster_rows	Logical. Order rows by hierarchical clustering (Euclidean distance, average linkage) of the z-scored profiles. Applied when there are more than two nodes. Default TRUE.
order_by	Used when rows are not clustered. Name of the measure that sorts rows in descending order. NULL (default) or an unknown name uses the first measure.
show_values	Logical. Print the raw centrality values in the cells. Default FALSE.
value_digits	Decimal places for cell values. Default 1.
low, mid, high	Color stops for the diverging scale. Defaults to blue, white and red.
limits	Numeric c(min, max) z-score range. Values outside are squished to the endpoints. Default c(-2.5, 2.5).
title, subtitle	Plot title and subtitle. The default subtitle is "z-scored within measure".
...	Passed to centrality when x is a network.

Value

A ggplot object.

Examples

```
plot_centrality_heatmap(regulation_net)
```

---

plot_chord *Chord Diagram*

Description

Plots a chord diagram in which nodes are arcs on the outer ring and edges are curved ribbons (chords) connecting them. Arc length is proportional to the total absolute weight of each node's edges, and chord width is proportional to the absolute edge weight.

Usage

```
plot_chord(
  x,
  directed = NULL,
  segment_colors = NULL,
  segment_border_color = "white",
  segment_border_width = 1,
  segment_pad = 0.02,
  segment_width = 0.08,
  chord_color_by = "source",
  chord_alpha = 0.5,
  chord_border = NA,
  self_loop = TRUE,
  labels = NULL,
  label_size = 1,
  label_color = "black",
  label_offset = 0.05,
  label_threshold = 0,
  threshold = 0,
  ticks = FALSE,
  tick_interval = NULL,
  tick_labels = TRUE,
  tick_size = 0.6,
  tick_color = "grey30",
  start_angle = pi/2,
  clockwise = TRUE,
  title = NULL,
  title_size = 1.2,
  background = NULL,
  ...
)
```

Arguments

x	A weight matrix, cograph_network, tna, igraph, or list with a matrix weights component.
directed	Logical. If NULL (default), auto-detected from matrix symmetry.
segment_colors	Colors for the outer ring segments, recycled to the number of nodes. NULL uses a built-in palette of 12 colors, interpolated when there are more nodes.
segment_border_color	Border color for segments.
segment_border_width	Border width for segments.
segment_pad	Gap between segments in radians.
segment_width	Radial thickness of the outer ring as a fraction of the radius.
chord_color_by	How to color chords. "target" uses the target segment color, and any other single string (default "source") uses the source segment color. A vector of

	colors is recycled to the number of chords, which are ordered by source node and then target node.
chord_alpha	Alpha transparency for chords.
chord_border	Border color for chords. NA for no border.
self_loop	Logical. Ignored. Self-loop chords are always shown.
labels	Node labels. NULL uses row names, FALSE suppresses labels.
label_size	Text size multiplier for labels.
label_color	Color for labels.
label_offset	Radial offset of labels beyond the outer ring.
label_threshold	Hide labels for nodes whose share of the total flow is below this value.
threshold	Minimum absolute weight to show a chord.
ticks	Logical. Add tick marks along the outer ring to indicate magnitude?
tick_interval	Spacing between major ticks in the same units as the weight matrix. Minor ticks are placed at half this spacing. NULL (default) selects an interval from the scale of the weights.
tick_labels	Logical. Show numeric labels at major ticks?
tick_size	Text size multiplier for tick labels.
tick_color	Color for tick marks and labels.
start_angle	Starting angle in radians (default pi/2, top).
clockwise	Logical. Lay out segments clockwise?
title	Optional plot title.
title_size	Text size multiplier for the title.
background	Background color for the plot. NULL (default) uses the current device background.
...	Ignored.

Details

The diagram is plotted with base R graphics. Segments and chords are polygons, and each ribbon follows quadratic Bezier curves through the center.

For directed networks, each segment is split into an outgoing half and an incoming half, and chords attach to the matching half. For undirected networks each edge forms one chord and the full segment arc is shared. Nodes without edges receive a small minimum arc so that they remain visible.

Value

Invisibly, a list with two data frames. segments has one row per node with columns node, start, end, mid (angles in radians) and flow. chords has one row per chord with columns from, to (node indices), from_start, from_end, to_start, to_end (attachment angles) and weight (absolute edge weight).

Examples

```
plot_chord(regulation_net)
```

`plot_compare` *Plot Network Difference (alias of plot_difference)*

Description

`plot_compare()` is an alias of `plot_difference()` and calls the same implementation. `tna::plot_compare()` calls it by name. `plot_difference()` is the preferred name.

Usage

```
plot_compare(x, ...)
```

Arguments

`x`	First network (see `plot_difference`).
`...`	Arguments passed to `plot_difference`.

Value

Invisibly, the value of `plot_difference`.

See Also

`plot_difference`

Examples

```
plot_compare(regulation_net, t(regulation_net))
```

`plot_comparison_heatmap`
Plot Comparison Heatmap

Description

Plots a heatmap of the difference between two weight matrices, or of either matrix alone. Rows are source nodes and columns are target nodes.

Usage

```
plot_comparison_heatmap(
  x,
  y = NULL,
  type = c("difference", "x", "y"),
  name_x = "x",
  name_y = "y",
  low_color = "blue",
  mid_color = "white",
  high_color = "red",
  limits = NULL,
  show_values = FALSE,
  value_size = 3,
  digits = 2,
  title = NULL,
  xlab = "Target",
  ylab = "Source"
)
```

Arguments

x	First network: matrix, cograph_network, tna, igraph, or list with a matrix weights component.
y	Second network, of the same type and dimensions as x. Required for type = "difference" and type = "y"; it may be NULL for type = "x".
type	What to display: "difference" (x - y), "x", or "y".
name_x	Label for the first network in the default title.
name_y	Label for the second network in the default title.
low_color	Color for low (negative) values.
mid_color	Color for zero.
high_color	Color for high (positive) values.
limits	Color scale limits. NULL uses the data range. Use c(-1, 1) for normalized values.
show_values	Logical. Display values in cells?
value_size	Text size for cell values.
digits	Decimal places for cell values.
title	Plot title. NULL builds one from type, name_x and name_y.
xlab	X-axis label.
ylab	Y-axis label.

Value

A ggplot object. The color scale is a diverging gradient with its midpoint at 0.

Examples

```
plot_comparison_heatmap(regulation_net, t(regulation_net))
```

`plot_degree_correlation` *Plot Degree-Degree Correlation*

Description

Scatter plot of each node's degree against the average degree of its neighbors. A positive slope indicates assortative mixing and a negative slope indicates disassortative mixing. When more than two nodes have neighbors, a least-squares line is added and the Pearson correlation is printed in the top margin.

Usage

```
plot_degree_correlation(
  x,
  mode = "all",
  directed = NULL,
  col = "steelblue",
  main = "Degree-Degree Correlation",
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna.
`mode`	Character. Degree type and neighborhood used for directed networks: `"all"` (default), `"in"`, or `"out"`.
`directed`	Logical or NULL. Default NULL (auto-detect).
`col`	Point color. Default `"steelblue"`.
`main`	Title. Default `"Degree-Degree Correlation"`.
`...`	Additional arguments passed to `plot`.

Value

Invisibly, a data frame with one row per node and columns `node`, `degree` and `avg_neighbor_degree`. The average neighbor degree is `NA` for nodes without neighbors.

See Also

`centrality`, `degree_distribution`, `network_summary`

Examples

```
cograph::plot_degree_correlation(student_interactions)
```

`plot_difference` *Plot Network Difference*

Description

Plots the difference between two networks (x - y) with `splot`. Positive differences (x > y) are shown in `pos_color` and negative differences (x < y) in `neg_color`. Node-level differences, such as initial probabilities, can be shown as donut charts.

Usage

```
plot_difference(
  x,
  y = NULL,
  i = NULL,
  j = NULL,
  pos_color = "#009900",
  neg_color = "#C62828",
  labels = NULL,
  title = NULL,
  inits_x = NULL,
  inits_y = NULL,
  show_inits = NULL,
  donut_inner_ratio = 0.8,
  force = FALSE,
  combined = TRUE,
  difference = FALSE,
  ...
)
```

Arguments

`x`	First network: matrix, `cograph_network`, `tna`, `igraph`, list with a matrix `weights` component, plain list of networks, or `group_tna`. For a `group_tna` with two groups, the groups are compared directly. With more groups and `i` and `j` both `NULL`, all pairwise comparisons are plotted.
`y`	Second network, of the same type as `x`. Ignored when `x` is a list or `group_tna`.
`i`	Index or name of the first group when `x` is a `group_tna` or a plain list. `NULL` selects the first element, except that all pairs are plotted for a `group_tna` of more than two groups when `j` is also `NULL`.
`j`	Index or name of the second group, with the same rules as `i`. `NULL` selects the second element.

pos_color	Color for positive differences (x > y).
neg_color	Color for negative differences (x < y).
labels	Node labels. NULL uses the row names of the weight matrix, or node indices when there are none.
title	Plot title. NULL uses "Network Difference (x - y)", or the two group names when they are available.
inits_x	Node values for x (e.g., initial probabilities). NULL extracts them from a tna object.
inits_y	Node values for y. NULL extracts them from a tna object.
show_inits	Logical. Show node differences as donuts? NULL (default) shows them when node values are available for both networks.
donut_inner_ratio	Inner radius ratio for donut (0-1).
force	Logical. Plot all pairs for a group_tna with more than four groups. Without it, such input stops with an error.
combined	Logical. When TRUE (default) and x is a multi-group input that triggers all-pairs plotting, panels are arranged in a grid via graphics::par(mfrow = ...). FALSE plots into a layout the caller has already configured (e.g. via panel_layout()). It has no effect for a single pair.
difference	Logical. If TRUE, x is treated as an already-subtracted difference network and y is ignored with a warning. A tna_comparison object (from tna::compare()) or a netdifference object is detected automatically and its difference matrix is used.
...	Additional arguments passed to splot. They override the defaults set here (layout = "oval", minimum = 0 and the TNA or psychometric styling preset).

Details

The weight matrices are subtracted element-wise. Both networks must have the same dimensions and, when present, the same node labels; otherwise the function stops with an error. A directed difference (or tna input) is styled with the TNA preset and an undirected difference with the psychometric preset.

When node values (inits) are given or extracted from tna objects, each node is shown as a donut whose filled fraction is the absolute difference, capped at 1, colored pos_color when x is higher and neg_color when y is higher.

plot_compare() is an alias of plot_difference().

Value

Invisibly, a list with elements weights (the element-wise difference matrix x - y) and inits (the node-value difference, or NULL when no node values were available). For the group_tna all-pairs path, a named list of such lists with one element per pair, named "<group_i>_vs_<group_j>".

See Also

plot_compare, plot_comparison_heatmap

Examples

```
plot_difference(regulation_net, t(regulation_net))
```

`plot_edge_diff_forest` *Forest Plot for Bootstrap Edge Differences*

Description

Plots pairwise edge weight differences from a `boot_glasso` object. Each row (linear) or spoke (circular) is one edge pair. The square marks the mean bootstrap difference, the bar spans its bootstrap percentile interval at level `1 - alpha`, and a dashed line or ring marks zero. Significant pairs are plotted in `pos_color` when the first edge is larger and in `neg_color` when the second edge is larger.

Usage

```
plot_edge_diff_forest(x, ...)

## S3 method for class 'boot_glasso'
plot_edge_diff_forest(
  x,
  alpha = NULL,
  layout = c("linear", "circular", "chord", "tile"),
  show_nonsig = FALSE,
  nonzero_only = FALSE,
  sort_by = c("estimate", "significance", "name"),
  n_top = NULL,
  pos_color = "#C0392B",
  neg_color = "#2C6E8A",
  nonsig_color = "#AAAAAA",
  ring_color = "#C8C8C8",
  label_size = 2.3,
  label_color = NULL,
  point_size = if (match.arg(layout) == "circular") 2 else 3,
  r_inner = 0.38,
  r_outer = 0.72,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

`x`	A `boot_glasso` object with `$boot_edges` and `$edge_diff_p`.
`...`	Currently unused.

alpha	Significance threshold. Default NULL, which inherits x$alpha, falling back to 0.05. The tile layout always uses x$alpha.
layout	"linear" (default), "circular", "chord", or "tile". The chord layout places all edge names on a unit circle and connects significant pairs with Bezier arcs whose width and color encode the mean bootstrap difference. The tile layout plots the full pairwise-difference matrix and ignores show_nonsig, nonzero_only, sort_by, n_top and the label and point arguments.
show_nonsig	Logical. Include non-significant pairs. Default FALSE.
nonzero_only	If TRUE, restrict to edges that are non-zero in the original network ($original_pcor). Without $original_pcor, edges whose absolute mean bootstrap weight is at least 10 percent of the largest are kept. Default FALSE.
sort_by	"estimate" (default), "significance", or "name" (linear layout only). When n_top is set, the retained pairs are ordered by estimate.
n_top	Restrict to top N pairs by absolute difference.
pos_color	Color when edge1 > edge2. Default "#C0392B" (crimson).
neg_color	Color when edge1 < edge2. Default "#2C6E8A" (teal-blue).
nonsig_color	Color for non-significant pairs. Default "#AAAAAA".
ring_color	Ring color (circular and chord layouts). Default "#C8C8C8".
label_size	Text size of edge labels (circular and chord layouts). Default 2.3.
label_color	Fixed label color (circular and chord layouts). NULL (default) uses the pair color in the circular layout and dark grey in the chord layout.
point_size	Size of estimate square (linear and circular layouts). Default 2 for layout = "circular" and 3 otherwise.
r_inner	Inner ring radius (circular). Default 0.38.
r_outer	Outer ring radius (circular). Default 0.72.
title	Plot title. Default NULL.
subtitle	Plot subtitle. Default NULL.

Value

A ggplot object.

Examples

```
set.seed(1)
data1 <- as.data.frame(matrix(rnorm(60), 20, 3, dimnames = list(NULL, c("A","B","C"))))
bg <- Nestimate::boot_glasso(data1, iter = 50, cs_iter = 25,
                             centrality = c("strength", "expected_influence"))
plot_edge_diff_forest(bg)
```

plot_edge_weights *Plot Edge Weight Distribution*

Description

Histogram of edge weights in a network. The number of edges and the mean and standard deviation of the weights are printed in the top margin.

Usage

```
plot_edge_weights(
  x,
  normalize = FALSE,
  bins = NULL,
  log = "",
  directed = NULL,
  col = "steelblue",
  border = "white",
  main = "Edge Weight Distribution",
  xlab = "Weight",
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna.
normalize	Logical. Show proportions. Default FALSE.
bins	Integer or NULL. Number of equal-width bins. With the default NULL, integer weights spanning at most 30 units get one bin per integer and other weights use the Freedman-Diaconis rule.
log	Character. "y" or "xy" log-scales the y-axis. Any other value gives linear axes. Default "".
directed	Logical or NULL. Default NULL (auto-detect).
col	Fill color. Default "steelblue".
border	Border color. Default "white".
main	Title. Default "Edge Weight Distribution".
xlab	X-axis label. Default "Weight".
...	Additional arguments passed to barplot.

Value

Invisibly, a numeric vector of edge weights (all 1 for an unweighted network).

Examples

```
cograph::plot_edge_weights(regulation_net)
```

`plot_heatmap` *Plot Network as Heatmap*

Description

Visualizes a network weight matrix as a heatmap. Single networks, clustered networks (blocks along the diagonal), and multi-group networks (`group_tna`, as a supra-adjacency matrix) are supported.

Usage

```
plot_heatmap(
  x,
  cluster_list = NULL,
  cluster_spacing = 0,
  show_legend = TRUE,
  legend_position = "right",
  legend_title = "Weight",
  colors = "viridis",
  limits = NULL,
  midpoint = NULL,
  na_color = "grey90",
  show_values = FALSE,
  value_size = 2.5,
  value_color = "black",
  value_fontface = "plain",
  value_fontfamily = "sans",
  value_halo = NULL,
  value_digits = 2,
  show_diagonal = TRUE,
  diagonal_color = NULL,
  cluster_labels = TRUE,
  cluster_borders = TRUE,
  border_color = "black",
  border_width = 0.5,
  row_labels = NULL,
  col_labels = NULL,
  show_axis_labels = TRUE,
  axis_text_size = 8,
  axis_text_angle = 45,
  title = NULL,
  subtitle = NULL,
  xlab = NULL,
```

```
  ylab = NULL,
  threshold = 0,
  aspect_ratio = 1,
  ...
)
```

Arguments

x	Network input: matrix, CographNetwork, cograph_network, tna, igraph, group_tna, or a list-like object with a $weights matrix.
cluster_list	Optional list of character vectors of node names defining node clusters. The matrix is reordered so that clusters form blocks along the diagonal. Nodes not listed are dropped, and a listed name that is not in the matrix is an error.
cluster_spacing	Gap size between clusters (in cell units). Default 0.
show_legend	Logical: display color legend? Default TRUE.
legend_position	Position: "right" (default), "left", "top", "bottom", "none".
legend_title	Title for legend. Default "Weight".
colors	Color palette: a vector of colors for the gradient, a palette name ("viridis", "heat", "blues", "reds", "greens", "diverging"), or a single color name, which gives a gradient from white to that color. With a diverging scale the first three colors are used as low, mid and high. Two colors are used as low and high around a white midpoint, and a single color name is used as the high end, with the blue low end and the light gray midpoint of the "diverging" palette. Default "viridis".
limits	Numeric vector c(min, max) for color scale. NULL for auto.
midpoint	Midpoint of a diverging scale. Supplying it makes the scale diverging. With the default NULL, the scale is diverging around 0 when the values span negative and positive numbers.
na_color	Color for NA values. Default "grey90".
show_values	Logical: display values in cells? Default FALSE.
value_size	Text size for cell values. Default 2.5.
value_color	Color for cell value text. Default "black".
value_fontface	Font face for values: "plain", "bold", "italic", "bold.italic". Default "plain".
value_fontfamily	Font family for values: "sans", "serif", "mono". Default "sans".
value_halo	Halo color behind value labels for readability on dark cells. Set to a color (e.g., "white") to enable, or NULL (default) to disable.
value_digits	Decimal places for values. Default 2.
show_diagonal	Logical: show diagonal values? Default TRUE.
diagonal_color	Currently unused. Diagonal cells use the fill scale unless they are hidden with show_diagonal = FALSE.

cluster_labels	Logical: show cluster or group labels? With a cluster_list, labels are shown only when the list is named. For group_tna input the group names are shown. Default TRUE.
cluster_borders	Logical: plot borders around clusters? Default TRUE.
border_color	Color for cluster borders. Default "black".
border_width	Width of cluster borders. Default 0.5.
row_labels	Row labels for a single-network heatmap. NULL uses the row names or indices.
col_labels	Column labels for a single-network heatmap. NULL uses the column names or indices.
show_axis_labels	Logical: show axis tick labels? Default TRUE. A clustered heatmap never shows axis tick labels.
axis_text_size	Size of axis labels. Default 8.
axis_text_angle	Angle for x-axis labels. Default 45.
title	Plot title. Default NULL, which gives no title, or "Supra-Adjacency Heatmap" for group_tna input.
subtitle	Plot subtitle. Default NULL.
xlab	X-axis label. Default NULL.
ylab	Y-axis label. Default NULL.
threshold	Minimum absolute value to display. Values with abs(value) < threshold are set to zero. It is not applied to group_tna input. Default 0.
aspect_ratio	Aspect ratio. Default 1 (square cells).
...	Additional arguments (currently unused).

Details

For group_tna objects, each group network becomes a diagonal block of a supra-adjacency matrix, with cells labelled group:node. The off-diagonal blocks are NA and take na_color. The node labels are taken from the first group.

Value

A ggplot object.

Examples

```
plot_heatmap(regulation_net)
```

plot_htna *Plot Heterogeneous TNA Network (Multi-Group Layout)*

Description

Plots a TNA model with nodes arranged in groups by a geometric layout. The circular layout (the default for `layout = "auto"`) places each group on an arc of a circle. The bipartite layout places exactly two groups in two columns, two rows, or facing arrangements. The polygon layout places three or more groups along the sides of a regular polygon with one side per group.

Usage

```
plot_htna(
  x,
  node_list = NULL,
  community = NULL,
  layout = "auto",
  use_list_order = TRUE,
  jitter = FALSE,
  jitter_amount = 0.8,
  jitter_side = "first",
  orientation = "vertical",
  group1_pos = -2,
  group2_pos = 2,
  group_spacing = NULL,
  node_spacing = NULL,
  columns = 1,
  column_spacing = NULL,
  layout_margin = 0.15,
  curvature = 0.4,
  group1_color = "#4FC3F7",
  group2_color = "#fbb550",
  group1_shape = "circle",
  group2_shape = "square",
  group_colors = NULL,
  group_shapes = NULL,
  angle_spacing = 0.15,
  edge_colors = NULL,
  intra_curvature = NULL,
  legend = TRUE,
  legend_position = "bottom",
  legend_horiz = NULL,
  legend_ncol = NULL,
  legend_size = 0.8,
  extend_lines = FALSE,
  scale = 1,
  nodes = NULL,
```

```
  label_abbrev = NULL,
  ...
)

htna(
  x,
  node_list = NULL,
  community = NULL,
  layout = "auto",
  use_list_order = TRUE,
  jitter = FALSE,
  jitter_amount = 0.8,
  jitter_side = "first",
  orientation = "vertical",
  group1_pos = -2,
  group2_pos = 2,
  group_spacing = NULL,
  node_spacing = NULL,
  columns = 1,
  column_spacing = NULL,
  layout_margin = 0.15,
  curvature = 0.4,
  group1_color = "#4FC3F7",
  group2_color = "#fbb550",
  group1_shape = "circle",
  group2_shape = "square",
  group_colors = NULL,
  group_shapes = NULL,
  angle_spacing = 0.15,
  edge_colors = NULL,
  intra_curvature = NULL,
  legend = TRUE,
  legend_position = "bottom",
  legend_horiz = NULL,
  legend_ncol = NULL,
  legend_size = 0.8,
  extend_lines = FALSE,
  scale = 1,
  nodes = NULL,
  label_abbrev = NULL,
  ...
)
```

Arguments

x	A tna object, weight matrix, or cograph_network.
node_list	Node groups can be specified as: • A list of character vectors (node names per group)

- A column name of the node table of a cograph_network (e.g., "groups")
- NULL, in which case the first node-table column named groups, group, clusters, cluster, community, module or layer is used, with a message

Groups must not overlap, every name must be a node of x, and at least two groups are required.

community — Community detection method to use for auto-grouping. If specified, overrides node_list. See detect_communities for available methods: "louvain", "walktrap", "fast_greedy", "label_prop", "infomap", "leiden".

layout — Layout type: "auto" (default, the circular layout), "bipartite" (exactly two groups), "polygon" (three or more groups), or "circular". The values "triangle", "rectangle", "pentagon" and "hexagon" are aliases for "polygon".

use_list_order — Logical. Use node_list order (TRUE) or weight-based order (FALSE). Only applies to the bipartite layout with orientation = "vertical".

jitter — Controls horizontal spread of nodes. Options:

- FALSE (default) or 0: No jitter (nodes aligned in columns)
- TRUE: Auto-compute jitter based on edge connectivity
- Numeric (0-1): Amount of jitter (0.3 = spread nodes 30\
- Named list: Manual per-node offsets by label (e.g., list(Wrong = -0.2))

Only applies to the bipartite layout with orientation set to "vertical" or "horizontal".

jitter_amount — Base jitter amount when jitter=TRUE. Default 0.8. Higher values spread nodes more toward the center. Only applies to bipartite layout.

jitter_side — Which side(s) to apply jitter: "first" (or "left"), "second" (or "right"), "both", or "none". Default "first" (only first group nodes are jittered toward center). Only applies to bipartite layout.

orientation — Layout orientation for bipartite: "vertical" (two columns, default), "horizontal" (two rows), "facing" (both groups on same horizontal line, group1 left, group2 right, tip-to-tip), or "circular" (two facing semicircles with a gap between them). Ignored for non-bipartite layouts.

group1_pos — Position of the first group in the bipartite layout, an x position for vertical and a y position for horizontal orientation. Overridden by group_spacing if specified.

group2_pos — Position of the second group in the bipartite layout. Overridden by group_spacing if specified.

group_spacing — Numeric. Distance between the two groups in bipartite layout. Overrides group1_pos/group2_pos. For example, group_spacing = 6 places groups at x = -3 and x = 3. Default NULL (uses group1_pos/group2_pos).

node_spacing — Numeric. Vertical (or horizontal) gap between nodes within a group. Default NULL (computed from the largest number of rows in a group). Increase for more space between nodes (e.g., 0.5 or 0.8).

columns — Integer or vector of length 2. Number of sub-columns per group. A single value applies to both groups. A vector of 2 sets columns per group independently (e.g., c(2, 1) puts the first group in 2 columns). Nodes are distributed evenly across sub-columns. Default 1.

`column_spacing` Numeric. Horizontal distance between sub-columns within a group. Default NULL (auto: `node_spacing * 2`).

`layout_margin` Margin around the layout (0-1). Default 0.15. Increase if labels or self-loops are clipped at the edges.

`curvature` Edge curvature amount.

`group1_color` Color for first group nodes.

`group2_color` Color for second group nodes.

`group1_shape` Shape for first group nodes.

`group2_shape` Shape for second group nodes.

`group_colors` Vector of colors for each group. Overrides group1_color/group2_color. If NULL, two-group layouts use group1_color/group2_color and 3+ group layouts cycle through a built-in palette of 12 colors. The length must equal the number of groups.

`group_shapes` Vector of shapes for each group. Overrides group1_shape/group2_shape. If NULL, two-group layouts use group1_shape/group2_shape and 3+ group layouts cycle through a built-in palette of 8 shapes. The length must equal the number of groups.

`angle_spacing` Controls empty space at corners (0-1). Default 0.15. Higher values create larger gaps in polygon and circular layouts. For circular auto layout, the default is increased to 0.35 unless explicitly set.

`edge_colors` Vector of colors for edges by source group. If NULL (default), uses darker versions of group_colors. Set to FALSE to use the default edge color.

`intra_curvature`

Numeric. Curvature amount for intra-group edges (edges between nodes in the same group). When set, intra-group edges are removed from the main plot and added separately as curves that arc away from the opposing group. Default NULL (intra-group edges are plotted by splot).

`legend` Logical. Whether to show a legend of the groups.

`legend_position`

Position for legend: "topright", "topleft", "bottomright", "bottomleft", "right", "left", "top", "bottom". Side positions place the legend in a reserved margin band, and corner positions place it inside the plot region.

`legend_horiz` Logical. Force horizontal (TRUE) or vertical (FALSE) legend. NULL (default) auto-selects: horizontal for "top"/"bottom" positions, vertical otherwise.

`legend_ncol` Integer. Number of columns when the legend is vertical. NULL (default) lets `graphics::legend` pick. Ignored when the legend is horizontal.

`legend_size` Legend text size (`cex`), as in `splot()`. The legend symbols are sized from it, and it is divided by `sqrt(scale)`. A value that is not a single positive number raises an error of class `cograph_bad_legend_size`.

`extend_lines` Logical or numeric. Add extension lines from nodes. Only applies to bipartite layout.

- FALSE (default): No extension lines
- TRUE: Lines extending toward the other group (length 0.1)

- Numeric: Length of extension lines

scale	Scaling factor for high-resolution output (e.g., scale = 4 for 300 dpi). The legend text and symbols are divided by `sqrt(scale)`, and the extension lines are plotted with line width `1 / sqrt(scale)`. The node positions do not depend on `scale`, because the layout is normalized after it is computed.
nodes	Node metadata. `NULL` (default) uses the node table of a `cograph_network`. A data frame replaces it, with one row per node in node order. Display labels are taken from its `labels` column, or from its `label` column when `labels` is absent.
label_abbrev	Label abbreviation: NULL (none), integer (max chars), or "auto" (adaptive based on node count). See `abbrev_label`.
...	Additional parameters passed to `tplot()`.

Value

Invisibly, the `cograph_network` object returned by `tplot()`. The function is called for its plot.

Examples

```
clusters <- list(Plan = c("Explore", "Plan", "Monitor", "Adapt", "Reflect"),
                 Act = c("Discuss", "Synthesize", "Evaluate", "Create", "Share"))
plot_htna(regulation_net, node_list = clusters)
```

---

plot_mcml *Plot Multi-Cluster Multi-Layer Network*

Description

Produces a two-layer hierarchical visualization of a clustered network. The bottom layer shows every node inside an elliptical cluster shell, with the within-cluster and between-cluster edges of the individual nodes. The top layer shows one summary node per cluster, with edges carrying the aggregated between-cluster weights. By default the colored slice of a summary node is the cluster's share of the initial state distribution (see `summary_pie`). Dashed lines connect each detail node to its summary node.

Usage

```
plot_mcml(
  x,
  cluster_list = NULL,
  expand = NULL,
  mode = c("weights", "tna"),
  theme = c("classic", "rich", "light"),
  layer_spacing = NULL,
  spacing = 3,
  shape_size = 1.2,
  summary_size = 4,
```

```
skew_angle = 60,
aggregation = c("sum", "mean", "max"),
minimum = 0,
colors = NULL,
legend = TRUE,
show_labels = TRUE,
nodes = NULL,
label_size = NULL,
label_abbrev = NULL,
node_size = 2.4,
node_shape = "circle",
cluster_shape = "circle",
title = NULL,
subtitle = NULL,
title_size = 1.2,
subtitle_size = 0.9,
legend_position = "right",
legend_size = 0.7,
legend_pt_size = 1.2,
summary_labels = TRUE,
summary_label_size = 0.8,
summary_label_position = 3,
summary_label_color = "gray20",
summary_arrows = TRUE,
summary_arrow_size = 0.1,
node_donut = NULL,
node_donut_inner_ratio = 0.55,
summary_donut_inner_ratio = 0.6,
summary_donut_show_value = FALSE,
curved_edges = NULL,
summary_curve = NULL,
summary_pie = c("inits", "self"),
edge_color_by = c("auto", "cluster", "sign"),
edge_positive_color = "#2E7D32",
edge_negative_color = "#C62828",
between_arrows = FALSE,
edge_width_range = c(0.3, 1.3),
between_edge_width_range = c(0.5, 2),
summary_edge_width_range = c(0.5, 2),
edge_alpha = 0.35,
between_edge_alpha = 0.6,
summary_edge_alpha = 0.7,
inter_layer_alpha = 0.5,
edge_labels = FALSE,
edge_label_size = 0.5,
edge_label_color = "gray40",
edge_label_digits = 2,
summary_edge_labels = FALSE,
```

```
  summary_edge_label_size = 0.6,
  top_layer_scale = c(0.8, 0.25),
  inter_layer_gap = 0.6,
  node_radius_scale = 0.55,
  shell_alpha = 0.15,
  shell_border_width = 0.75,
  node_border_color = "gray30",
  node_border_width = 0.4,
  summary_border_color = "gray20",
  summary_border_width = 0.6,
  label_color = "gray20",
  label_position = 3,
  directed = NULL,
  ...
)
```

Arguments

x	A square weight matrix with row and column names matching the node names in cluster_list, a tna object (its $weights are used), a cograph_network (its weights and node table are used), a cluster_summary from csum, or an mcml or mcml_pc object from Nestimate. A cluster_summary, mcml or mcml_pc object is plotted as it is, and cluster_list, aggregation and nodes are ignored.
cluster_list	Assignment of nodes to clusters. A named list of character vectors gives the node names of each cluster, and the list names become the cluster labels, for example list(GroupA = c("A", "B"), GroupB = c("C", "D")). A cograph_communities object from detect_communities is also accepted. For a cograph_network, a string names a node column to group by, and NULL uses a node column named clusters, cluster, groups or group.
expand	Names of clusters whose member states are shown as separate nodes in the summary layer. "all" or TRUE expands every cluster, and NULL (default) shows one summary node per cluster. The bottom layer always shows the clusters. The summary layer is then recomputed from x with each expanded cluster split into its states. For a cluster_summary, mcml or mcml_pc input it is computed with Nestimate::macro_network(), and a cograph_expand_unavailable error is raised when that function is not available.
mode	"weights" (default) or "tna". With "tna", edge_labels and summary_edge_labels default to TRUE unless they are supplied. The plotted weights are the same in both modes.
theme	Visual preset, one of "classic" (default, pie-chart nodes and straight summary edges), "rich" (donut nodes on both layers, curved summary edges and self-loops) or "light" (as "rich" with no shell outline and a lighter shell fill). node_donut and curved_edges override the preset when supplied.
layer_spacing	Vertical position of the summary layer, which sets the height of the figure. NULL (default) places it automatically above the bottom layer, at a distance set by inter_layer_gap. "fill" increases the gap between the layers so that the figure uses the full height of the device, and never makes it smaller than the

automatic gap. A single positive number gives the height of the center of the summary layer above the center of the bottom layer, in the units of `spacing`, and overrides `inter_layer_gap`. A number that places the summary layer inside the bottom layer raises a `cograph_layers_overlap` warning, and any other value raises a `cograph_bad_layer_spacing` error.

`spacing` Distance from the center to each cluster in the bottom layer. Default 3.

`shape_size` Radius of each cluster shell in the bottom layer. Default 1.2.

`summary_size` Size of the summary nodes. The radius of a summary node is `0.0875 * summary_size`. Default 4.

`skew_angle` Perspective tilt in degrees, from 0 to 90. At 0 the bottom layer is seen from directly above and at 90 it collapses to a line. Default 60.

`aggregation` Method for aggregating node-level edge weights into cluster-level weights: `"sum"` (default), `"mean"` or `"max"`.

`minimum` Edge weight threshold. Edges whose absolute weight does not exceed this value are not plotted. Default 0.

`colors` Character vector of cluster colors, recycled to the number of clusters. `NULL` (default) uses the Okabe-Ito palette.

`legend` Logical. Add a legend of cluster colors. Default `TRUE`.

`show_labels` Logical. Show node labels in the bottom layer. Default `TRUE`.

`nodes` Node metadata data frame for display labels. Its `labels` column, or else its `label` column, is used as the label text of the nodes in row order. It replaces the node table of a `cograph_network` and is ignored when `x` is a `cluster_summary`.

`label_size` Text size (`cex`) of bottom-layer node labels. `NULL` (default) uses 0.6.

`label_abbrev` Label abbreviation passed to `abbrev_label`: `NULL` (default) for full labels, an integer for the maximum number of characters, or `"auto"` for a length chosen from the number of nodes.

`node_size` Size of the detail nodes. The radius of a detail node is `0.035 * node_size`. Default 2.4.

`node_shape` Shape of the detail nodes, a single value or one value per node. `"circle"` (default) plots a pie chart of the node's self-transition share. Other node shapes, such as `"square"`, `"diamond"` or `"triangle"`, are plotted as solid shapes in the cluster color.

`cluster_shape` Not used. It is kept for compatibility with earlier versions.

`title` Plot title. Default `NULL`.

`subtitle` Subtitle shown below the figure. Default `NULL`.

`title_size` Text size (`cex.main`) of the title. Default 1.2.

`subtitle_size` Text size (`cex.sub`) of the subtitle. Default 0.9.

`legend_position` Legend position: `"right"` (default), `"left"`, `"top"`, `"bottom"` or `"none"`.

`legend_size` Text size (`cex`) of legend labels. Default 0.7.

`legend_pt_size` Point size (`pt.cex`) of legend symbols. Default 1.2.

`summary_labels` Logical. Show cluster names next to the summary nodes. Default `TRUE`.

summary_label_size
: Text size of summary labels. Default 0.8.

summary_label_position
: Position of summary labels relative to their nodes: 1 = below, 2 = left, 3 = above, 4 = right. When it is not supplied, each label is placed on the side of its node that faces away from the center of the summary layer.

summary_label_color
: Color of summary labels. Default `"gray20"`.

summary_arrows
: Logical. Add arrowheads to summary edges. Default `TRUE`. Arrowheads are removed when `directed = FALSE`.

summary_arrow_size
: Size of the arrowheads on summary edges. Default 0.10.

node_donut
: Logical or `NULL`. `TRUE` or `FALSE` turns donut nodes on or off. `NULL` (default) follows `theme`.

node_donut_inner_ratio
: Hole size, from 0 to 1, of the detail-node donut. Default 0.55.

summary_donut_inner_ratio
: Hole size, from 0 to 1, of the summary donut. Default 0.6.

summary_donut_show_value
: Logical. Print the fill proportion in the center of each summary donut. Default `FALSE`.

curved_edges
: Logical or `NULL`. `TRUE` or `FALSE` turns curved summary edges on or off. `NULL` (default) follows `theme`.

summary_curve
: Numeric or `NULL`. Curvature of curved summary edges. `NULL` (default) uses 0.25 for directed and 0 for undirected networks.

summary_pie
: What the colored slice of a summary node shows. `"inits"` (default) shows the cluster's share of the initial state distribution, so the slices of all clusters sum to 1. `"self"` shows the cluster's self-retention, the diagonal weight divided by the row sum of the summary matrix.

edge_color_by
: Edge coloring on all layers. `"auto"` (default) colors edges by cluster when all weights are non-negative and by sign when any weight is negative. `"cluster"` always uses the color of the source cluster. `"sign"` always uses `edge_positive_color` and `edge_negative_color`. The threshold `minimum` and the edge widths use absolute weights, so negative edges are plotted.

edge_positive_color
: Color of positive edges under sign coloring. Default `"#2E7D32"` (green).

edge_negative_color
: Color of negative edges under sign coloring. Default `"#C62828"` (red).

between_arrows
: Logical. Add arrowheads to between-cluster edges in the bottom layer. Default `FALSE`.

edge_width_range
: Numeric vector `c(min, max)` of line widths for within-cluster edges. Widths grow linearly with absolute weight, from `min` at zero to `max` at the largest absolute weight. Default `c(0.3, 1.3)`.

`between_edge_width_range`
: Numeric vector `c(min, max)` of line widths for between-cluster edges. Default `c(0.5, 2.0)`.

`summary_edge_width_range`
: Numeric vector `c(min, max)` of line widths for summary edges. Default `c(0.5, 2.0)`.

`edge_alpha`
: Opacity, from 0 to 1, of within-cluster edges. Default 0.35.

`between_edge_alpha`
: Opacity, from 0 to 1, of between-cluster edges. Default 0.6.

`summary_edge_alpha`
: Opacity, from 0 to 1, of summary edges. Default 0.7.

`inter_layer_alpha`
: Opacity, from 0 to 1, of the dashed inter-layer lines. Default 0.5.

`edge_labels`
: Logical. Show weight labels on within-cluster edges. Default `FALSE`, or `TRUE` when `mode = "tna"`.

`edge_label_size`
: Text size of within-cluster edge labels. Default 0.5.

`edge_label_color`
: Color of within-cluster edge labels. Default `"gray40"`.

`edge_label_digits`
: Number of decimal places of edge labels on both layers. Default 2.

`summary_edge_labels`
: Logical. Show weight labels on summary edges. Default `FALSE`, or `TRUE` when `mode = "tna"`.

`summary_edge_label_size`
: Text size of summary edge labels. Default 0.6.

`top_layer_scale`
: Numeric vector `c(x_scale, y_scale)` giving the horizontal and vertical radii of the oval of summary nodes as multiples of `spacing`. Default `c(0.8, 0.25)`.

`inter_layer_gap`
: Vertical distance from the upper edge of the bottom layer to the center of the summary layer, as a multiple of `spacing`. Default 0.6.

`node_radius_scale`
: Radius of the circle of nodes inside each cluster shell, as a fraction of `shape_size`. Default 0.55.

`shell_alpha`
: Fill opacity, from 0 to 1, of the cluster shells. Default 0.15, or 0.10 with `theme = "light"`.

`shell_border_width`
: Line width of the cluster shell borders. Default 0.75, or 0 with `theme = "light"`.

`node_border_color`
: Border color of the detail nodes. Default `"gray30"`.

`node_border_width`
: Border width of the detail nodes. Default 0.4.

`summary_border_color`
: Border color of the summary nodes. Default `"gray20"`.

summary_border_width
: Border width of the summary nodes. Default 0.6.

label_color
: Text color of detail node labels. Default "gray20".

label_position
: Not used. Detail labels are placed to the left or right of each node according to its position in the shell.

directed
: Logical or NULL. NULL (default) uses the $meta$directed flag of a cluster_summary or mcml input and the $directed field of other objects, and treats a plain matrix as undirected when it is symmetric. With TRUE, every non-zero weight is plotted as a directed edge with an arrowhead. With FALSE, arrowheads are removed from all layers, overriding summary_arrows and between_arrows, each pair is plotted once from the upper triangle with its label at the midpoint, and a warning is raised when the aggregated weights are not symmetric.

...
: Not used.

Details

For a multi-cluster plot without the summary layer, see plot_mtna. For stacked multilevel or multiplex layers, see plot_mlna.

A weight matrix, tna object or cograph_network is passed together with cluster_list, and the aggregated weights are computed with csum. A cluster_summary computed beforehand with csum can be passed as x instead, which avoids repeating the aggregation when the same clustering is plotted several times.

For a directed network the aggregated weights are computed with type = "tna", so each row of the summary matrix sums to 1. For an undirected network they are computed with type = "cooccurrence". The mode argument changes only the default of the edge labels.

Bottom-layer clusters are arranged on a circle of radius spacing, flattened by the perspective skew_angle. Nodes inside each cluster sit on a smaller circle of radius shape_size * node_radius_scale. The summary nodes are placed on an oval above the bottom layer whose radii are set by top_layer_scale.

Value

Invisibly, the cluster_summary object used for plotting. It can be passed back to plot_mcml(), printed, or converted with as_tna.

Edge Types

The plot contains four kinds of edges, each with its own visual parameters.

Within-cluster (bottom) Edges between nodes of the same cluster, set by edge_width_range, edge_alpha, edge_labels, edge_label_size, edge_label_color and edge_label_digits.

Between-cluster (bottom) Edges between cluster shells, set by between_edge_width_range, between_edge_alpha and between_arrows.

Summary (top) Edges between summary nodes, set by summary_edge_width_range, summary_edge_alpha, summary_edge_labels, summary_edge_label_size, summary_arrows and summary_arrow_size.

Inter-layer (dashed) Lines from each detail node to its summary node, set by inter_layer_alpha.

See Also

csum for the aggregated cluster data, plot_mtna for a multi-cluster plot without a summary layer, plot_mlna for stacked multilevel or multiplex layers, detect_communities for algorithmic cluster detection

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
plot_mcml(regulation_net, clusters)
```

---

plot_mixed_network *Plot Mixed Network*

Description

Plots one network that combines the edges of a symmetric matrix, shown as straight undirected edges, with the edges of an asymmetric matrix, shown as curved directed edges.

Usage

```
plot_mixed_network(
  sym_matrix,
  asym_matrix,
  layout = "oval",
  sym_color = "ivory4",
  asym_color = COGRAPH_SCALE$tna_edge_color,
  curvature = 0.3,
  edge_width = NULL,
  node_size = 7,
  title = NULL,
  threshold = 0,
  edge_labels = TRUE,
  arrow_size = 0.61,
  edge_label_size = 0.6,
  edge_label_position = 0.7,
  initial = NULL,
  ...
)
```

Arguments

sym_matrix	A symmetric matrix of undirected relationships. Each non-zero pair is plotted once as a straight edge without arrows.
asym_matrix	An asymmetric matrix of directed relationships, with the same dimensions as sym_matrix. Its edges are plotted as curved arrows.

layout	Layout algorithm or coordinate matrix. Default "oval".
sym_color	Color for undirected edges. Default "ivory4".
asym_color	Color for directed edges. Either a single color, or two colors for reciprocal pairs: the first for the edge from the lower-indexed node and the second for the reverse edge. Non-reciprocal edges use the first color. Default "#003355" (dark blue, the TNA edge color).
curvature	Curvature magnitude for directed edges. Default 0.3.
edge_width	Edge width(s). If NULL (default), widths scale with edge weight as in TNA plots. A numeric value overrides the scaling.
node_size	Node size. Default 7.
title	Plot title. Default NULL.
threshold	Minimum absolute edge weight to display. Values with abs(value) < threshold are set to zero, and zero-weight edges are not plotted. Default 0.
edge_labels	Show edge weight labels. Default TRUE.
arrow_size	Arrow head size for directed edges. Default 0.61 (TNA style).
edge_label_size	Size of edge labels. Default 0.6.
edge_label_position	Position of edge labels along edge (0-1). Default 0.7.
initial	Optional numeric vector of initial state probabilities. A named vector is matched to the node names, with missing states set to 0; an unnamed vector is used in node order. Nodes are then plotted as donuts filled in proportion to the initial probability. A warning is issued when the values do not sum to 1 (tolerance 0.01). Default NULL.
...	Additional arguments passed to splot.

Value

Invisibly, a list with three elements: edges, a data frame with one row per plotted edge and columns from, to (node indices), weight, type ("undirected" or "directed") and color; and sym_matrix and asym_matrix, the input matrices after thresholding.

Examples

```
plot_mixed_network(symmetrize(regulation_net, keep_format = TRUE), regulation_net)
```

plot_ml_heatmap *Multilayer Network Heatmap*

Description

Visualizes multiple network layers as heatmaps on tilted 3D-perspective planes, in the style of plot_mlna.

Usage

```
plot_ml_heatmap(
  x,
  layer_list = NULL,
  colors = "viridis",
  layer_spacing = NULL,
  skew = 0.4,
  compress = 0.6,
  show_connections = FALSE,
  connection_color = "#E63946",
  connection_style = "dashed",
  show_borders = TRUE,
  border_color = "black",
  border_width = 1,
  cell_border_color = "white",
  cell_border_width = 0.2,
  show_labels = TRUE,
  show_node_labels = TRUE,
  node_label_size = 3,
  label_size = 5,
  show_legend = TRUE,
  legend_title = "Weight",
  title = NULL,
  limits = NULL,
  na_color = "grey90",
  threshold = 0
)
```

Arguments

x	A list of matrices (one per layer), a group_tna object, a cograph_network, or a single matrix with layer_list specified.
layer_list	Named list of node vectors, one per layer. For a matrix x each layer is the submatrix of its nodes. For a cograph_network it can also be the name of a node column, and when NULL a node column named layers, layer, level or levels is used.
colors	Color palette: "viridis", "heat", "blues", "reds", "inferno", "plasma", or a vector of colors. Any other single name gives the viridis colors. Default "viridis".

`layer_spacing` Vertical spacing between layers, in data units. `NULL` (the default) uses 1.1 times the plane height, which is the number of rows of a layer times `compress`, with a minimum of 1, so the planes do not overlap. A positive number sets the spacing directly.

`skew` Horizontal skew for perspective effect (0-1). Default 0.4.

`compress` Vertical compression for perspective (0-1). Default 0.6.

`show_connections` Show inter-layer connection lines? Default FALSE.

`connection_color` Color for inter-layer connections. Default "#E63946".

`connection_style` Line style: "dashed", "solid", "dotted". Default "dashed".

`show_borders` Show layer outline borders? Default TRUE.

`border_color` Color for layer borders. Default "black".

`border_width` Width of layer borders. Default 1.

`cell_border_color` Color for cell borders. Default "white".

`cell_border_width` Width of cell borders. Default 0.2.

`show_labels` Show layer name labels? Default TRUE.

`show_node_labels` Show the row and column names? Default TRUE. When all layers have the same row and column names, the names are shown once, along the left and lower edges of the front plane. When the layers hold different nodes, each plane shows its own row names along its left edge and the column names below the front plane are those of the front (last) layer.

`node_label_size` Size of the row and column names. Default 3.

`label_size` Size of layer labels. Default 5.

`show_legend` Show color legend? Default TRUE.

`legend_title` Title for legend. Default "Weight".

`title` Plot title. Default NULL.

`limits` Color scale limits c(min, max). NULL for auto.

`na_color` Color for NA values. Default "grey90".

`threshold` Minimum absolute value to display. Cells with `abs(value) < threshold` are set to NA and shown in `na_color`. Default 0.

Value

A ggplot2 object.

Examples

```
clusters <- list(Plan = c("Explore", "Plan", "Monitor", "Adapt", "Reflect"),
                 Act = c("Discuss", "Synthesize", "Evaluate", "Create", "Share"))
plot_ml_heatmap(regulation_net, layer_list = clusters)
```

---

plot_mlna *Multilevel Network Visualization*

Description

Plots a multilevel network as a stack of layers in a pseudo-3D perspective view. Each layer is enclosed in a parallelogram shell. Edges within a layer are plotted as solid curved arrows, and edges between adjacent layers are plotted as straight lines in the style set by `between_style`.

Usage

```
plot_mlna(
  model,
  layer_list = NULL,
  community = NULL,
  layout = "horizontal",
  layer_spacing = 4,
  layer_width = 8,
  layer_depth = 4,
  skew_angle = 25,
  node_spacing = 0.7,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  within_edges = TRUE,
  between_edges = TRUE,
  between_style = 2,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.15,
  node_size = 3,
  minimum = 0,
  scale = 1,
  show_labels = TRUE,
  nodes = NULL,
  label_abbrev = NULL,
  ...
)

mlna(
```

```
  model,
  layer_list = NULL,
  community = NULL,
  layout = "horizontal",
  layer_spacing = 4,
  layer_width = 8,
  layer_depth = 4,
  skew_angle = 25,
  node_spacing = 0.7,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  within_edges = TRUE,
  between_edges = TRUE,
  between_style = 2,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.15,
  node_size = 3,
  minimum = 0,
  scale = 1,
  show_labels = TRUE,
  nodes = NULL,
  label_abbrev = NULL,
  ...
)
```

Arguments

`model`	A tna object, weight matrix, or cograph_network.
`layer_list`	Layer assignment of the nodes. One of • a named list of character vectors with the node names of each layer; at least two non-overlapping layers are required; • a single string naming a column of the node data (e.g., "layer"); • NULL, in which case the first node-data column found among "layer", "layers", "level", "levels", "groups", "group", "clusters" and "cluster" is used.
`community`	Community detection method used to form the layers. When given, it overrides `layer_list`. Passed to `detect_communities`, which accepts "louvain", "walktrap", "fast_greedy", "label_prop", "infomap" and "leiden". An error is raised when fewer than two communities are found.
`layout`	Node layout within layers. "horizontal" (default) places nodes on a horizontal line, "circle" arranges them in an ellipse, and "spring" uses force-directed placement based on within-layer edges.
`layer_spacing`	Vertical distance between layer centers. Default 4.
`layer_width`	Horizontal width of each layer shell. Default 8.

layer_depth	Depth of each layer shell in the perspective view. Default 4.
skew_angle	Angle of perspective skew in degrees. Default 25.
node_spacing	Proportion of the layer width (0-1) over which nodes are spread. Higher values place nodes closer to the layer edges. Default 0.7.
colors	Vector of fill colors, one per layer. NULL (default) uses a built-in palette.
shapes	Vector of node shapes, one per layer. Supported values are "circle", "square", "diamond" and "triangle"; other values are plotted as circles. NULL (default) assigns "circle", "square", "diamond" and "triangle" to the first four layers.
edge_colors	Vector of colors for between-layer edges, indexed by the source layer. NULL (default) uses a built-in palette. Within-layer edges use a darkened version of the layer color.
within_edges	Logical. Plot edges within layers. Default TRUE.
between_edges	Logical. Plot edges between adjacent layers. Default TRUE.
between_style	Line type of between-layer edges. Default 2 (dashed). Use 1 for solid, 3 for dotted.
show_border	Logical. Plot the parallelogram shells around layers. Default TRUE.
legend	Logical. Show the layer legend. Default TRUE.
legend_position	Position of the legend. Default "topright".
curvature	Curvature of within-layer edges. Default 0.15.
node_size	Size of nodes. Default 3.
minimum	Edge weight threshold. Edges whose weight does not exceed this value are not plotted. Default 0.
scale	Scaling factor for high-resolution output (e.g., scale = 4 for 300 dpi). Node sizes, line widths and text sizes are divided by `sqrt(scale)`. Default 1.
show_labels	Logical. Show node labels. Default TRUE.
nodes	Node metadata. NULL (default) uses the node data of a cograph_network. A data frame replaces it and must have one row per node in the node order of `model`. Display text is taken from its `labels` column if present, otherwise from its `label` column.
label_abbrev	Label abbreviation: NULL (none), integer (max chars), or "auto" (adaptive based on node count).
...	Additional parameters (currently unused).

Value

Invisibly returns NULL.

See plot_mlna.

Examples

```
clusters <- list(Plan = c("Explore", "Plan", "Monitor", "Adapt", "Reflect"),
                 Act = c("Discuss", "Synthesize", "Evaluate", "Create", "Share"))
plot_mlna(regulation_net, layer_list = clusters)
```

plot_mtna *Multi-Cluster TNA Network Plot*

Description

Plots a network whose nodes are grouped into clusters. Each cluster is plotted as a shell shape containing its nodes. By default, edges between clusters are aggregated into summary edges and edges within clusters are plotted individually.

Usage

```
plot_mtna(
  x,
  cluster_list = NULL,
  community = NULL,
  layout = "circle",
  spacing = 4,
  shape_size = 1.8,
  node_spacing = 0.5,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  bundle_edges = TRUE,
  bundle_strength = 0.8,
  summary_edges = TRUE,
  aggregation = c("sum", "mean", "max", "min", "median", "density"),
  within_edges = TRUE,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.3,
  node_size = 3,
  layout_margin = 0.15,
  scale = 1,
  show_labels = FALSE,
  nodes = NULL,
  label_size = NULL,
  label_abbrev = NULL,
  cluster_shape = NULL,
  ...
)

mtna(
  x,
  cluster_list = NULL,
  community = NULL,
  layout = "circle",
```

```
  spacing = 4,
  shape_size = 1.8,
  node_spacing = 0.5,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  bundle_edges = TRUE,
  bundle_strength = 0.8,
  summary_edges = TRUE,
  aggregation = c("sum", "mean", "max", "min", "median", "density"),
  within_edges = TRUE,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.3,
  node_size = 3,
  layout_margin = 0.15,
  scale = 1,
  show_labels = FALSE,
  nodes = NULL,
  label_size = NULL,
  label_abbrev = NULL,
  cluster_shape = NULL,
  ...
)
```

Arguments

x	A tna object, weight matrix, or cograph_network.
cluster_list	Cluster assignment of the nodes. One of • a named list of character vectors with the node names of each cluster; at least two non-overlapping clusters are required; • a single string naming a column of the node data (e.g., "groups"); • NULL, in which case the first node-data column found among "clusters", "cluster", "groups", "group", "community" and "module" is used.
community	Community detection method used to form the clusters. When given, it overrides cluster_list. See detect_communities for available methods.
layout	How to arrange the clusters: "circle" (default), "grid", "horizontal", "vertical".
spacing	Distance between cluster centers. Default 4.
shape_size	Size of each cluster shape (shell radius). Default 1.8.
node_spacing	Radius for node placement within shapes in summary mode, as a proportion (0-1) of shape_size. Default 0.5. When summary_edges = FALSE, nodes are placed at radius shape_size.
colors	Vector of colors for each cluster. NULL (default) uses a built-in palette.
shapes	Vector of shapes for each cluster. Defaults cycle through "circle", "square", "diamond", "triangle", "pentagon", "hexagon", "star", and "cross". In summary

mode, shells other than circle, square, diamond and triangle are plotted as circles.

`edge_colors` Vector of edge colors by source cluster. NULL (default) uses a built-in palette.

`bundle_edges` Logical. Order the nodes around each shell by the direction of the clusters they connect to, so that edges toward the same cluster leave from neighbouring nodes. Used when `summary_edges = FALSE`. Default TRUE.

`bundle_strength`
Currently unused.

`summary_edges` Logical. Show aggregated summary edges between clusters instead of individual node edges. Default TRUE.

`aggregation` Method for aggregating edge weights between clusters: "sum" (total flow), "mean" (average strength), "max" (strongest link), "min" (weakest link), "median", or "density" (normalized by possible edges). Default "sum". Only used when summary_edges = TRUE.

`within_edges` Logical. When summary_edges is TRUE, also show individual edges within each cluster. Default TRUE.

`show_border` Logical. When `summary_edges = FALSE`, plot a dashed circle around each cluster. Default TRUE.

`legend` Logical. Whether to show legend. Default TRUE.

`legend_position`
Position for legend. Default "topright".

`curvature` Edge curvature. Default 0.3.

`node_size` Size of nodes inside shapes (summary mode). Default 3.

`layout_margin` Margin around the layout as fraction of range (summary mode). Default 0.15.

`scale` Scaling factor for high-resolution output. Values greater than 1 reduce node, edge, label, and legend sizes by `sqrt(scale)` while leaving cluster spacing and shape_size unchanged. Default 1.

`show_labels` Logical. Show node labels inside clusters (summary mode). Default FALSE.

`nodes` Node metadata. NULL (default) uses the node data of a cograph_network. A data frame replaces it and must have one row per node in the node order of `x`. In summary mode, display text is taken from its `labels` column if present, otherwise from its `label` column.

`label_size` Label text size (summary mode). Default NULL (auto-scaled).

`label_abbrev` Label abbreviation in summary mode: NULL (none), integer (max chars), or "auto" (adaptive based on node count).

`cluster_shape` Accepted for compatibility; currently unused. Use `shapes` to control cluster shell shapes.

`...` When `summary_edges = FALSE`, additional parameters passed to `plot_tna()`. In summary mode, only `edge.lwd`, `edge.labels`, `edge.label.cex` and `minimum` are read.

Value

Invisibly returns a `cluster_summary` object when `summary_edges = TRUE`, and otherwise the `plot_tna()` result (a `cograph_network` object).

See `plot_mtna`.

See Also

`csum`, `plot_mcml`

Examples

```
clusters <- list(Plan = c("Explore", "Plan", "Monitor", "Adapt", "Reflect"),
                 Act = c("Discuss", "Synthesize", "Evaluate", "Create", "Share"))
plot_mtna(regulation_net, cluster_list = clusters)
```

---

plot_network_evolution

Plot Network Evolution (Small Multiples)

Description

Plots a network at different time points side by side. The input is an edge list data frame with a time column, a `cograph_network` whose stored edge data contain such a column, or a list of networks. All panels share one node layout. At least two periods are required.

Usage

```
plot_network_evolution(
  x,
  time = NULL,
  slices = NULL,
  cumulative = FALSE,
  labels = NULL,
  layout = "spring",
  ncol = NULL,
  node_size = 5,
  seed = 42,
  combined = TRUE,
  ...
)
```

Arguments

`x`	An edge list data frame with columns `from`, `to`, optionally `weight`, and a time column; a `cograph_network` with stored edge data; or a list of network objects (matrices, igraph, etc.).

time	Character. Name of the time column in x. Required for data frame input and ignored if x is a list.
slices	Integer or NULL. Number of equal-width bins of the numeric time column. Default NULL uses the unique time values. Bins work with cumulative = TRUE as well.
cumulative	Logical. If TRUE, each panel shows all edges up to that time point (growing network). If FALSE (default), each panel shows only edges from that period.
labels	Character vector of panel labels, one per period. The default NULL uses the time values, or "T1", "T2", ... for list input.
layout	Character, or a matrix or data frame of coordinates. Any character value computes one Fruchterman-Reingold layout from the union of all edges and uses it for every panel. A matrix or data frame gives the x and y coordinates of the nodes in its first two columns, one row per node; rows named after the nodes are matched by name, otherwise they are taken in node order. Default "spring".
ncol	Integer. Number of grid columns. The default NULL uses min(number of periods, 4).
node_size	Numeric. Node size passed to splot. Default 5.
seed	Integer or NULL. Random seed for the shared layout. The caller's random number state is restored on exit. NULL sets no seed. Default 42.
combined	Logical. When TRUE (default), the period panels are arranged in an internal grid via graphics::par(mfrow = ...). When FALSE, the panels are plotted into a layout the caller has already configured, for example with panel_layout().
...	Additional arguments passed to splot.

Value

Invisibly, a list with one element per period. For data frame input each element is the edge-list data frame of that period (all earlier periods included when cumulative = TRUE). For list input it is the input list.

Examples

```
set.seed(1)
edges <- data.frame(
  from = sample(LETTERS[1:5], 30, replace = TRUE),
  to   = sample(LETTERS[1:5], 30, replace = TRUE),
  week = sample(1:4, 30, replace = TRUE))
cograph::plot_network_evolution(edges, time = "week")
```

plot_robustness *Plot Network Robustness*

Description

Plots the fraction of nodes remaining in the largest connected component against the fraction of nodes or edges removed, with one line per attack strategy, in base graphics.

Usage

```
plot_robustness(
  ...,
  x = NULL,
  measures = c("betweenness", "degree", "random"),
  colors = NULL,
  title = "Network Robustness: sequential removal of nodes",
  xlab = "Fraction of removed nodes",
  ylab = "Fraction of remaining nodes",
  lwd = 1.5,
  legend_pos = "topright",
  n_iter = 1000,
  seed = NULL,
  type = "vertex"
)
```

Arguments

...	One or more robustness results from robustness. When these are supplied, x is ignored.
x	Network on which robustness is computed for each of measures. Used when ... is empty.
measures	Character vector of attack strategies to compare when x is supplied. Default c("betweenness", "degree", "random").
colors	Vector of colors named by measure ("betweenness", "degree", "random"). Default NULL uses red for betweenness, green for degree and blue for random. Unmatched measures are gray.
title	Plot title. Default "Network Robustness: sequential removal of nodes".
xlab	X-axis label. Default "Fraction of removed nodes".
ylab	Y-axis label. Default "Fraction of remaining nodes".
lwd	Line width. Default 1.5.
legend_pos	Legend position. Default "topright".
n_iter	Number of iterations for random removal when x is supplied. Default 1000.
seed	Random seed. Default NULL.
type	Removal type, "vertex" or "edge", when x is supplied. Default "vertex". With "edge", measures must not include "degree".

Value

Invisibly, a `cograph_robustness` data frame that stacks the plotted results (columns as in `robustness`).

Examples

```
plot_robustness(x = regulation_net, n_iter = 20, seed = 1)
```

---

plot_simplicial *Simplicial Complex Visualization*

Description

Visualizes higher-order pathways as smooth blobs overlaid on a network layout. Source nodes are filled with `node_color` (blue by default) and target nodes with `target_color` (orange by default).

Usage

```
plot_simplicial(
  x = NULL,
  pathways = NULL,
  method = "hon",
  max_pathways = 10L,
  pathway_index = NULL,
  anomaly = c("all", "over", "under"),
  layout = "circle",
  labels = NULL,
  node_color = "#4A7FB5",
  target_color = "#E8734A",
  ring_color = "#F5A623",
  node_size = 22,
  label_size = 5,
  label_color = "#e8e8e8",
  target_label_color = NULL,
  label_halo = TRUE,
  label_halo_color = NULL,
  label_halo_width = 0.035,
  label_halo_alpha = 0.6,
  blob_alpha = 0.25,
  blob_colors = NULL,
  blob_linetype = NULL,
  blob_linewidth = 0.7,
  blob_line_alpha = 0.8,
  shadow = TRUE,
  title = NULL,
  dismantled = FALSE,
  ncol = NULL,
```

```
  ordered = NULL,
  direction = NULL,
  direction_cues = c("shade", "ring", "arrows"),
  node_radius = NULL,
  legend = NULL,
  ...
)
```

Arguments

x	A network object: tna, netobject, matrix, igraph, cograph_network, net_hon, net_hypa, net_association_rules, net_link_prediction or simplicial_complex. A data frame with a path column is used as pathways, and the states are then taken from the path strings.
pathways	Character vector of pathway strings, a list of character vectors, a net_hon, net_hypa, net_association_rules, net_link_prediction or simplicial_complex object, or a data frame with a path column, such as the output of Nestimate::mogen_transitions(). Accepted string forms are "A B -> C", "A -> B -> C", "A, B, C", "A - B - C" and "A B C", and the last state is the target. The rows of a data frame with a count column are sorted by count in decreasing order before max_pathways is applied. When NULL and x is a model with sequence data, pathways are built with method.
method	Pathway source when building from a tna or netobject: "hon" (default) for a higher-order network, "hypa" for paths that are anomalous under a hypergeometric null model, or "rules" for association-rule itemsets from Nestimate::association_rules(), which are plotted as sets.
max_pathways	Maximum number of pathways to display. HON pathways are ranked by count and HYPA pathways by anomaly ratio. NULL shows all. Default 10.
pathway_index	Optional positive integer vector selecting ranked pathways before max_pathways is applied. For example, 2 plots the second-ranked pathway and 2:4 plots the pathways ranked second through fourth.
anomaly	HYPA anomaly type to display, one of "all" (default), "over" or "under". It applies to a net_hypa input and to method = "hypa". For any other input an explicitly supplied value is ignored with a warning.
layout	"circle" (default) or a coordinate matrix with one row per state.
labels	Display labels. NULL uses state names.
node_color	Source node fill color.
target_color	Target node fill color.
ring_color	Donut ring color.
node_size	Node point size.
label_size	Label text size.
label_color	Label text color for source and target nodes. Default "#e8e8e8", a very light grey.
target_label_color	Target-node label color. NULL (default) uses label_color.

label_halo Logical. Place a contrasting halo behind each label so that it stays readable on node, blob and background fills. Default TRUE.

label_halo_color
Halo color. NULL (default) chooses black or white from the luminance of label_color.

label_halo_width
Halo thickness in plot units. Default 0.035. A value of 0 removes the halo.

label_halo_alpha
Halo opacity, from 0 to 1. Default 0.6.

blob_alpha Blob fill transparency.

blob_colors Blob fill colors (recycled).

blob_linetype Blob border line styles (recycled).

blob_linewidth Blob border line width.

blob_line_alpha
Blob border line transparency.

shadow Add soft drop shadows?

title Plot title of the combined overlay.

dismantled If TRUE, one panel per pathway arranged in a grid.

ncol Number of columns in the grid when dismantled = TRUE. NULL (default) uses the ceiling of the square root of the number of pathways.

ordered Logical. TRUE treats each pathway as a path whose last state is the target. FALSE treats it as a set of equal members, so no node gets target_color, no direction cue is shown and the panel title lists the members. NULL (default) treats net_association_rules, simplicial_complex and method = "rules" pathways as sets and all other inputs as paths.

direction Logical. Show the order of traversal in each per-pathway panel by a light-to-dark shading of the node cores along the path, a ring whose ring_color is strongest on the side facing the next state, and an arrowhead pointing to the successor. NULL (default) enables it when dismantled = TRUE. direction = TRUE with dismantled = FALSE raises a cograph_direction_needs_panels error, because a state shared by several pathways in the overlay has no single successor. Direction is turned off for sets and when target_color equals node_color.

direction_cues Which direction cues to show, any of "shade", "ring" and "arrows". Default all three.

node_radius Node core radius in data units, used in panels that show direction. NULL (default) scales it to the panel extent.

legend Add a legend strip beneath a dismantled grid. NULL (default) adds it when direction is on.

... Additional arguments passed to Nestimate::build_hon(), Nestimate::build_hypa() or Nestimate::association_rules() when pathways are built from a model.

Details

When x is a tna or netobject model with sequence data and pathways is NULL, the pathways are built from the sequences with the **Nestimate** package. Pathways passed as net_hon or net_hypa objects have their numeric state identifiers translated to labels when x is a tna or netobject.

Value

Invisibly, a ggplot object for the combined overlay. With dismantled = TRUE the arranged grid is returned instead, as a gtable when **gridExtra** is available and otherwise as a list of the per-pathway ggplot objects. NULL is returned, with a message, when no pathways could be extracted. The figure is printed as a side effect.

Examples

```
plot_simplicial(regulation_net, c("Plan Monitor -> Adapt", "Explore Reflect -> Plan"))
```

---

plot_temporal *Temporal Network Prism (3D Glass Box)*

Description

Plots a network at different time points as vertical planes inside a 3D oblique-projection box, with time running from left to right. Each network plane extends into the depth of the box, and all planes share one node layout. At least two time points are required.

Usage

```
plot_temporal(
  x,
  time = NULL,
  slices = NULL,
  cumulative = FALSE,
  labels = NULL,
  layout = "spring",
  node_size = 2.5,
  node_color = "steelblue",
  color_by = c("layer", "node"),
  node_shape = 21,
  node_border = "gray30",
  edge_color = "#E41A1C",
  edge_width = 1.5,
  edge_alpha = 0.35,
  plane_color = "gray92",
  plane_alpha = 0.2,
  plane_border = "gray60",
  plane_lty = 2,
  box = TRUE,
  box_color = "gray40",
  connections = FALSE,
  connection_color = "gray50",
  connection_alpha = 0.15,
  minimum = 0,
```

```
  show_labels = FALSE,
  label_size = 0.4,
  title = NULL,
  angle = c(1, 0.7),
  seed = 42,
  ...
)
```

Arguments

x	An edge list data frame with columns `from`, `to`, optionally `weight`, and a time column; a `cograph_network` whose stored edge data contain the time column; or a list of network objects.
time	Character. Name of the time column. Required for data frame input. It also labels the time axis.
slices	Integer or NULL. Number of equal-width bins of the numeric time column. Empty bins are kept as empty planes. Default NULL uses the unique time values.
cumulative	Logical. If TRUE, each plane contains all edges up to its time point. Default FALSE.
labels	Character vector of layer labels, one per plane. The default NULL uses the time values, or `"T1"`, `"T2"`, ... for list input.
layout	Character or matrix. A character value computes one Fruchterman-Reingold layout from the summed network. A two-column matrix supplies shared coordinates, one row per node. Default `"spring"`.
node_size	Numeric. Point size (`cex`). Default 2.5.
node_color	Character or vector. Node fill color. A single color applies to every node. An unnamed vector is recycled across layers and colors each plane as a whole, unless `color_by = "node"`. A named vector is matched to node names and gives each node the same color on every plane. A named vector that lacks a node raises an error of class `cograph_node_color_incomplete`. Default `"steelblue"`.
color_by	One of `"layer"` (default) or `"node"`. It sets whether an unnamed `node_color` vector is recycled over layers or over nodes. A named `node_color` always colors by node.
node_shape	Integer. Point shape (`pch`). Default 21 (filled circle).
node_border	Character. Node border color. Default `"gray30"`.
edge_color	Character or vector. Edge color (single or per-layer). Default `"#E41A1C"`.
edge_width	Numeric. Maximum added edge width. An edge has width `0.3 + edge_width * abs(w) / max(abs(w))`. Default 1.5.
edge_alpha	Numeric. Edge transparency (0-1). Default 0.35.
plane_color	Character or vector. Plane fill color (single or per-layer). Default `"gray92"`.
plane_alpha	Numeric. Plane fill transparency (0-1). Default 0.2.
plane_border	Character. Plane border color. Default `"gray60"`.
plane_lty	Integer. Plane border line type. Default 2 (dashed).

box	Logical. Whether the 3D bounding box is plotted. Default TRUE.
box_color	Character. Box edge color. Default "gray40".
connections	Logical. Whether lines connect each node to itself on the next plane. Default FALSE.
connection_color	Character. Color of the connecting lines. Default "gray50".
connection_alpha	Numeric. Transparency of the connecting lines. Default 0.15.
minimum	Numeric. Only edges with weight greater than minimum are plotted. Default 0.
show_labels	Logical. Show node labels. Default FALSE.
label_size	Numeric. Label text size. Default 0.4.
title	Character or NULL. Plot title. Default NULL.
angle	Numeric vector of length 2: c(dz_x, dz_y) controlling the oblique projection shear. Default c(1.0, 0.7).
seed	Integer or NULL. Random seed for the shared layout. The caller's random number state is restored on exit. NULL sets no seed. Default 42.
...	Currently unused.

Value

Invisibly, a list of weight matrices, one per plane, with one row and one column per node.

See Also

plot_network_evolution, plot_mlna

Examples

```
set.seed(1)
edges <- data.frame(
  from = sample(LETTERS[1:5], 30, replace = TRUE),
  to   = sample(LETTERS[1:5], 30, replace = TRUE),
  week = sample(1:3, 30, replace = TRUE))
cograph::plot_temporal(edges, time = "week")
```

---

plot_tna *TNA-Style Network Plot (qgraph Compatible)*

Description

Plots a network with splot() and TNA styling, using qgraph argument names such as vsize, edge.color and pie. tplot() is an alias.

Usage

```
plot_tna(
  x,
  color = NULL,
  labels = NULL,
  layout = "oval",
  theme = "colorblind",
  mar = c(0.1, 0.1, 0.1, 0.1),
  cut = NULL,
  edge.label.position = 0.7,
  edge.label.cex = 0.6,
  edge.color = COGRAPH_SCALE$tna_edge_color,
  vsize = 7,
  pie = NULL,
  pieColor = NULL,
  lty = NULL,
  directed = NULL,
  minimum = NULL,
  posCol = NULL,
  negCol = NULL,
  arrowAngle = NULL,
  title = NULL,
  ...
)

tplot(
  x,
  color = NULL,
  labels = NULL,
  layout = "oval",
  theme = "colorblind",
  mar = c(0.1, 0.1, 0.1, 0.1),
  cut = NULL,
  edge.label.position = 0.7,
  edge.label.cex = 0.6,
  edge.color = COGRAPH_SCALE$tna_edge_color,
  vsize = 7,
  pie = NULL,
  pieColor = NULL,
  lty = NULL,
  directed = NULL,
  minimum = NULL,
  posCol = NULL,
  negCol = NULL,
  arrowAngle = NULL,
  title = NULL,
  ...
)
```

Arguments

x	A weight matrix (adjacency matrix) or tna object
color	Node fill colors
labels	Node labels
layout	Layout: "oval" (default), "circle", "spring", or a coordinate matrix.
theme	Plot theme. Default "colorblind".
mar	Plot margins (numeric vector of length 4)
cut	Edge emphasis threshold, passed to edge_cutoff of splot().
edge.label.position	Position of edge labels along edge (0-1)
edge.label.cex	Edge label size multiplier
edge.color	Edge colors
vsize	Node size
pie	Donut fill values in 0-1 (for example, initial probabilities).
pieColor	Donut fill colors.
lty	Line type for edges: 1 solid, 2 dashed, 3 dotted, 4 dotdash, 5 longdash, 6 twodash, or a line-type name.
directed	Logical, is the graph directed? NULL (default) treats a symmetric weight matrix as undirected and any other input as directed.
minimum	Minimum absolute edge weight to display, passed to threshold of splot().
posCol	Color for positive edges
negCol	Color for negative edges
arrowAngle	Arrow head angle in radians. Default NULL, which leaves splot()'s own arrow_angle default of pi/6 (30 degrees) in place.
title	Plot title
...	Additional arguments passed to splot(). They take precedence over the translated qgraph arguments.

Value

The cograph_network object returned by splot(), invisibly.

Examples

```
plot_tna(regulation_net)
```

plot_trajectories *Plot Individual Trajectories*

Description

Creates an alluvial-style diagram in which each individual's trajectory is plotted as a separate line. It calls `plot_transitions()` with `track_individuals = TRUE`.

Usage

```
plot_trajectories(
  x,
  from_title = NULL,
  title = NULL,
  from_colors = NULL,
  flow_color_by = "first",
  node_width = 0.08,
  node_border = NA,
  node_spacing = 0.02,
  label_size = 3.5,
  label_position = c("beside", "inside", "above", "below", "outside"),
  mid_label_position = NULL,
  label_halo = TRUE,
  label_color = "black",
  label_fontface = "plain",
  label_nudge = 0.02,
  title_size = 5,
  title_color = "black",
  title_fontface = "bold",
  curve_strength = 0.6,
  line_alpha = 0.3,
  line_width = 0.5,
  jitter_amount = 0.8,
  show_totals = FALSE,
  total_size = 4,
  total_color = "white",
  total_fontface = "bold",
  show_values = FALSE,
  value_position = c("center", "origin", "destination"),
  value_size = 3,
  value_color = "black",
  value_halo = NULL,
  value_fontface = "bold",
  value_nudge = 0.03,
  value_min = 0,
  value_digits = 2,
  column_gap = 1,
```

```
  proportional_nodes = TRUE,
  node_label_format = NULL,
  bundle_size = NULL,
  bundle_legend = TRUE,
  bundle_legend_size = 3,
  bundle_legend_color = "grey50",
  bundle_legend_fontface = "italic",
  bundle_legend_position = c("bottom", "top")
)
```

Arguments

x	Data frame with one column per time point and one row per individual trajectory, or a tna object with sequence data. In a data frame, a missing value (NA) marks a time point at which the individual was not observed: no line enters or leaves that column for the individual, and the node sizes count the observed states only. Colors by "first" and "last" use the first and last observed states.
from_title	Column titles. Default NULL, which uses the column names of x. Pass a character vector to override them.
title	Optional plot title.
from_colors	Colors for the left-side nodes. In multi-step and individual-tracking plots, the colors of all states. Default NULL uses the built-in palette.
flow_color_by	Color trajectory lines by state. Supports "source", "destination", "first", "last", or NULL. Default "first".
node_width	Width of node rectangles (0-1 scale). Default 0.08.
node_border	Border color for nodes. Default NA (no border).
node_spacing	Vertical spacing between nodes (0-1 scale). Default 0.02.
label_size	Size of node labels. Default 3.5.
label_position	Position of node labels: "beside" (default), "inside", "above", "below", "outside". In multi-step and individual-tracking plots, "beside" and "outside" label the first and last columns only. See mid_label_position for middle columns.
mid_label_position	Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as label_position. Default NULL uses label_position.
label_halo	Logical: add a white halo around labels and column titles? Default TRUE.
label_color	Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots. Two-column aggregate plots use black external labels and white inside labels.
label_fontface	Font face of state name labels ("plain", "bold", "italic", "bold.italic"). Default "plain". Applied to multi-step and individual-tracking plots.
label_nudge	Distance between node edge and label (in plot units). Default 0.02. Used by multi-step and individual-tracking plots.
title_size	Size of column titles. Default 5.

`title_color` Color of column title text. Default "black". Applied to multi-step and individual-tracking plots.

`title_fontface` Font face of column titles. Default "bold". Applied to multi-step and individual-tracking plots.

`curve_strength` Controls bezier curve shape (0-1). Default 0.6.

`line_alpha` Alpha for individual tracking lines. Default 0.3. When bundling is active, values up to 0.3 are raised to 0.9 and larger values are increased by 0.3, capped at 1.

`line_width` Width of individual tracking lines. Default 0.5. When bundling is active, widths range from `line_width` to twice that value according to the number of cases per line.

`jitter_amount` Currently unused. Lines are spaced evenly within each node. Default 0.8.

`show_totals` Logical: show total counts on nodes? Default FALSE.

`total_size` Size of total labels. Default 4.

`total_color` Color of total labels. Default "white".

`total_fontface` Font face of total labels. Default "bold". Applied to multi-step and individual-tracking plots.

`show_values` Logical: show transition counts on flows? Default FALSE.

`value_position` Position of trajectory value labels: `"center"`, `"origin"`, or `"destination"`. Default `"center"`.

`value_size` Size of value labels on flows. Default 3.

`value_color` Color of value labels. Default "black".

`value_halo` Logical: add halo around flow value labels? Default NULL uses `label_halo`. Applied to multi-step and individual-tracking plots.

`value_fontface` Font face of flow value labels. Default "bold". Applied to multi-step and individual-tracking plots.

`value_nudge` Distance of value labels from node edge when using "origin" or "destination" positions. Default 0.03.

`value_min` Minimum count to show a flow value label in multi-step and individual-tracking plots. Default 0 (show all). Two-column aggregate plots show every nonzero value label when `show_values = TRUE`.

`value_digits` Number of decimal places for flow value labels and node totals. Default 2.

`column_gap` Horizontal spread of columns (0-1) for multi-step and individual-tracking plots. Default 1 uses the full width. Smaller values (e.g., 0.6) bring the columns closer together.

`proportional_nodes`
Logical: size nodes proportionally to counts in individual-tracking plots? When FALSE, all states in a column have equal height. Default TRUE.

`node_label_format`
Format string for node labels with `{state}` and `{count}` placeholders in individual-tracking plots, for example `"{state} (n={count})"`. Default NULL (plain state name).

bundle_size
: Controls line bundling for large datasets in individual-tracking plots. Default NULL (no bundling). A value of 1 or more sets the number of cases each line represents. A value in (0, 1) sets the fraction of the original number of lines to keep (e.g., 0.15 keeps about 15 percent). Paths with fewer than half the cases of one line are dropped.

bundle_legend
: Logical or character: show an annotation when bundling is active? Default TRUE shows "Each line ~ N cases". A string is used as custom text, with `{n}` as the placeholder for the number of cases.

bundle_legend_size
: Size of the bundle legend text. Default 3.

bundle_legend_color
: Color of the bundle legend text. Default "grey50".

bundle_legend_fontface
: Font face of the bundle legend text. Default "italic".

bundle_legend_position
: Position of the bundle legend: "bottom" (default) or "top".

Value

A `ggplot` object.

See Also

`plot_transitions`, `plot_alluvial`

Examples

```
df <- data.frame(
  Baseline = c("Light", "Light", "Intense", "Resource"),
  Week4    = c("Light", "Intense", "Intense", "Light"),
  Week8    = c("Resource", "Intense", "Light", "Light"))
plot_trajectories(df, flow_color_by = "first")
```

---

plot_transitions *Plot Transitions Between States*

Description

Creates an alluvial (Sankey) diagram of how items flow from one set of categories to another, for example cluster membership changes or state changes between time points. Aggregated flows are plotted as ribbons whose width is proportional to the transition count. With `track_individuals = TRUE`, each row of a data frame is plotted as a separate line.

Usage

```
plot_transitions(
  x,
  from_title = "From",
  to_title = "To",
  title = NULL,
  from_colors = NULL,
  to_colors = NULL,
  flow_fill = "#888888",
  flow_alpha = 0.4,
  flow_color_by = NULL,
  flow_border = NA,
  flow_border_width = 0.5,
  node_width = 0.08,
  node_border = NA,
  node_spacing = 0.02,
  label_size = 3.5,
  label_position = c("beside", "inside", "above", "below", "outside"),
  mid_label_position = NULL,
  label_halo = TRUE,
  label_color = "black",
  label_fontface = "plain",
  label_nudge = 0.02,
  title_size = 5,
  title_color = "black",
  title_fontface = "bold",
  curve_strength = 0.6,
  show_values = FALSE,
  value_position = c("center", "origin", "destination", "outside_origin",
    "outside_destination"),
  value_size = 3,
  value_color = "black",
  value_halo = NULL,
  value_fontface = "bold",
  value_nudge = 0.03,
  value_min = 0,
  show_totals = FALSE,
  total_size = 4,
  total_color = "white",
  total_fontface = "bold",
  conserve_flow = TRUE,
  min_flow = 0,
  threshold = 0,
  value_digits = 2,
  column_gap = 1,
  track_individuals = FALSE,
  line_alpha = 0.3,
  line_width = 0.5,
```

```
  jitter_amount = 0.8,
  proportional_nodes = TRUE,
  node_label_format = NULL,
  bundle_size = NULL,
  bundle_legend = TRUE,
  bundle_legend_size = 3,
  bundle_legend_color = "grey50",
  bundle_legend_fontface = "italic",
  bundle_legend_position = c("bottom", "top")
)
```

Arguments

x	Input data in one of these formats: • A transition matrix (rows = from, columns = to, values = counts). • A vector of "before" states, with the vector of "after" states passed as the second argument (`from_title`). Both vectors must have the same length, greater than 2, and their contingency table is computed. • A data frame with two columns of raw observations, whose contingency table is computed. • A data frame with three or more columns of raw observations, one column per time point, plotted as a multi-step diagram. • A data frame with columns `from`, `to` and `count`. • A list of matrices for multi-step transitions. • A `tna` object. Its sequence data are used as a data frame of time points (rows with missing values are dropped), or its weight matrix when no sequence data are stored.
from_title	Title for the left column. Default "From". For multi-step and individual-tracking plots, a vector with one title per column. Data frame input then uses the column names by default, and a vector shorter than the number of columns is replaced by "T1", "T2", ...
to_title	Title for the right column. Default "To". Ignored for multi-step plots.
title	Optional plot title.
from_colors	Colors for the left-side nodes. In multi-step and individual-tracking plots, the colors of all states. Default NULL uses the built-in palette.
to_colors	Colors for the right-side nodes in two-column plots. Default NULL uses the built-in palette.
flow_fill	Fill color for flows. Default "#888888" (grey). In multi-step plots it is replaced by the state colors when `flow_color_by` is set. Individual-tracking lines do not use it.
flow_alpha	Alpha transparency for flows. Default 0.4.
flow_color_by	Color flows by state. Multi-step aggregate plots accept `"source"` or `"destination"`. Individual-tracking plots also accept `"first"` and `"last"`. Default NULL uses `flow_fill`. Two-column aggregate plots ignore this argument.
flow_border	Border color for flows. Default NA (no border).

`flow_border_width`
: Line width for flow borders. Default 0.5.

`node_width`
: Width of node rectangles (0-1 scale). Default 0.08.

`node_border`
: Border color for nodes. Default NA (no border).

`node_spacing`
: Vertical spacing between nodes (0-1 scale). Default 0.02.

`label_size`
: Size of node labels. Default 3.5.

`label_position`
: Position of node labels: "beside" (default), "inside", "above", "below", "outside". In multi-step and individual-tracking plots, "beside" and "outside" label the first and last columns only. See `mid_label_position` for middle columns.

`mid_label_position`
: Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as `label_position`. Default NULL uses `label_position`.

`label_halo`
: Logical: add a white halo around labels and column titles? Default TRUE.

`label_color`
: Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots. Two-column aggregate plots use black external labels and white inside labels.

`label_fontface`
: Font face of state name labels ("plain", "bold", "italic", "bold.italic"). Default "plain". Applied to multi-step and individual-tracking plots.

`label_nudge`
: Distance between node edge and label (in plot units). Default 0.02. Used by multi-step and individual-tracking plots.

`title_size`
: Size of column titles. Default 5.

`title_color`
: Color of column title text. Default "black". Applied to multi-step and individual-tracking plots.

`title_fontface`
: Font face of column titles. Default "bold". Applied to multi-step and individual-tracking plots.

`curve_strength`
: Controls bezier curve shape (0-1). Default 0.6.

`show_values`
: Logical: show transition counts on flows? Default FALSE.

`value_position`
: Position of flow values: "center", "origin", "destination", "outside_origin", "outside_destination". Default "center". Individual-tracking plots use "center", "origin" and "destination".

`value_size`
: Size of value labels on flows. Default 3.

`value_color`
: Color of value labels. Default "black".

`value_halo`
: Logical: add halo around flow value labels? Default NULL uses `label_halo`. Applied to multi-step and individual-tracking plots.

`value_fontface`
: Font face of flow value labels. Default "bold". Applied to multi-step and individual-tracking plots.

`value_nudge`
: Distance of value labels from node edge when using "origin" or "destination" positions. Default 0.03.

`value_min`
: Minimum count to show a flow value label in multi-step and individual-tracking plots. Default 0 (show all). Two-column aggregate plots show every nonzero value label when `show_values = TRUE`.

`show_totals`
: Logical: show total counts on nodes? Default FALSE.

`total_size` Size of total labels. Default 4.

`total_color` Color of total labels. Default "white".

`total_fontface` Font face of total labels. Default "bold". Applied to multi-step and individual-tracking plots.

`conserve_flow` Logical. When TRUE (default), node heights on both sides of a two-column plot are proportions of the same total flow. When FALSE, each side is scaled to its own total. Ignored for multi-step and individual-tracking plots.

`min_flow` Minimum flow value to display in aggregate plots. Default 0 (show all).

`threshold` Minimum flow value to display in aggregate plots. Flows below `max(threshold, min_flow)` are removed. Default 0.

`value_digits` Number of decimal places for flow value labels and node totals. Default 2.

`column_gap` Horizontal spread of columns (0-1) for multi-step and individual-tracking plots. Default 1 uses the full width. Smaller values (e.g., 0.6) bring the columns closer together.

`track_individuals`
Logical: plot individual lines instead of aggregated flows? Default FALSE. When TRUE and `x` is a data frame of raw observations, each row becomes a separate line.

`line_alpha` Alpha for individual tracking lines. Default 0.3. When bundling is active, values up to 0.3 are raised to 0.9 and larger values are increased by 0.3, capped at 1.

`line_width` Width of individual tracking lines. Default 0.5. When bundling is active, widths range from `line_width` to twice that value according to the number of cases per line.

`jitter_amount` Currently unused. Lines are spaced evenly within each node. Default 0.8.

`proportional_nodes`
Logical: size nodes proportionally to counts in individual-tracking plots? When FALSE, all states in a column have equal height. Default TRUE.

`node_label_format`
Format string for node labels with `{state}` and `{count}` placeholders in individual-tracking plots, for example `"{state} (n={count})"`. Default NULL (plain state name).

`bundle_size` Controls line bundling for large datasets in individual-tracking plots. Default NULL (no bundling). A value of 1 or more sets the number of cases each line represents. A value in (0, 1) sets the fraction of the original number of lines to keep (e.g., 0.15 keeps about 15 percent). Paths with fewer than half the cases of one line are dropped.

`bundle_legend` Logical or character: show an annotation when bundling is active? Default TRUE shows "Each line ~ N cases". A string is used as custom text, with `{n}` as the placeholder for the number of cases.

`bundle_legend_size`
Size of the bundle legend text. Default 3.

`bundle_legend_color`
Color of the bundle legend text. Default "grey50".

bundle_legend_fontface
: Font face of the bundle legend text. Default "italic".

bundle_legend_position
: Position of the bundle legend: "bottom" (default) or "top".

Value

A ggplot object.

Examples

```
plot_transitions(regulation_net)
```

---

project_bipartite *Project Bipartite Network to One-Mode*

Description

Projects a two-mode (bipartite/incidence) network into a one-mode adjacency matrix. Row-mode projection yields a matrix of shared-column connections among row nodes; column-mode projection does the converse.

Usage

```
project_bipartite(x, mode = "rows", method = "sum", ...)
```

Arguments

x
: An incidence matrix (rows = type 1 nodes, columns = type 2 nodes) where non-zero entries indicate connections. Can also be a data.frame with columns `type1`, `type2`, and optionally `weight`.

mode
: Character. `"rows"` (default) projects onto row nodes (result: n_rows x n_rows). `"columns"` projects onto column nodes (result: n_cols x n_cols).

method
: Character. Projection method:

  `"sum"` Weighted projection: `A %*% t(A)` (rows) or `t(A) %*% A` (columns). Edge weight equals sum of shared connection-weight products.

  `"binary"` Co-occurrence count: binarize A first, then compute overlap. Edge weight equals number of shared connections.

  `"jaccard"` Jaccard similarity: shared / (total_i + total_j - shared) for each pair.

  `"cosine"` Cosine similarity: dot product of row (or column) vectors divided by the product of their norms.

  `"newman"` Newman's weighted projection (Newman 2001): each shared affiliation contributes `1 / (d_k - 1)` where `d_k` is the degree of the shared node. Gives more weight to connections through exclusive affiliations.

...
: Additional arguments (currently unused).

Details

Only `method = "sum"` and `method = "cosine"` use the incidence values themselves. `"binary"`, `"jaccard"` and `"newman"` first binarize the incidence matrix (`x > 0`), so any weights are discarded for those three.

For the Newman projection, affiliations shared with only one node of the focal type (`d_k = 1`) are skipped, since `1 / (d_k - 1)` is undefined. This follows the convention in Newman (2001).

Value

A square adjacency matrix, one row and column per node of the projected mode: `n_rows` x `n_rows` named by `rownames(x)` for `mode = "rows"`, `n_cols` x `n_cols` named by `colnames(x)` for `mode = "columns"`. The diagonal is set to 0 (no self-loops).

References

Newman, M. E. J. (2001). Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical Review E*, 64(1), 016132.

See Also

`is_bipartite`, `plot_heatmap`

Examples

```
cograph::project_bipartite(regulation_net, mode = "rows", method = "jaccard")
```

---

reaching_global *Global Reaching Centrality (Mones, Vicsek & Vicsek 2012)*

Description

A graph-level hierarchy measure computed from per-node local reaching centralities:

$$GRC(G) = \frac{1}{N-1} \sum_{v} \left(\max_{u} LRC(u) - LRC(v) \right)$$

Usage

```
reaching_global(x, mode = "all", ...)
```

Arguments

`x`	Network input (matrix, edge-list data frame, igraph, network, cograph_network, tna object).
`mode`	For directed networks: `"all"` (default), `"in"`, or `"out"`.
`...`	Additional arguments passed to `centrality_reaching_local`.

Details

Values close to 0 indicate a flat network in which all nodes reach equal proportions of the graph. Larger values indicate a more hierarchical structure. The result matches `networkx.global_reaching_centrality`.

Value

A single numeric value. On an unweighted graph it lies in $[0, 1]$. On a weighted graph the local reaching centralities scale with the edge weights, so the value is unbounded. A graph with at most one node returns 0.

References

Mones, E., Vicsek, L., & Vicsek, T. (2012). Hierarchy measure for complex networks. *PLoS ONE*, 7(3), e33799.

See Also

`centrality_reaching_local`, `summarize_network`.

Examples

```
reaching_global(regulation_net, mode = "out")
```

---

regulation_net *Learning Regulation Transition Network*

Description

A synthetic weighted transition network among ten learning regulation states, used in the package examples and the introduction vignette. Each cell holds the weight of the transition from the row state to the column state.

Usage

```
regulation_net
```

Format

A 10 x 10 numeric matrix with row and column names `Explore`, `Plan`, `Monitor`, `Adapt`, `Reflect`, `Discuss`, `Synthesize`, `Evaluate`, `Create` and `Share`. Thirty of the 90 off-diagonal cells carry weights between 0.05 and 0.49. The remaining cells, including the diagonal, are zero.

Details

The network is synthetic and represents no observed data. It was generated with `set.seed(42)`. Thirty off-diagonal cells were drawn at random and given weights drawn uniformly between 0.05 and 0.5, rounded to two decimals. Rows are not normalized.

Value

A 10 x 10 numeric matrix of transition weights with state names as row and column names.

Source

Synthetic, generated for the package examples.

Examples

```
regulation_net
splot(regulation_net, tna_styling = TRUE)
```

`remove_edges` *Remove Edges from a Network*

Description

Deletes the edges between given pairs of nodes. In an undirected network the order of the two endpoints does not matter.

Usage

```
remove_edges(
  x,
  from,
  to,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`from`	Source nodes, by label or index.
`to`	Target nodes, by label or index. The same length as `from`.
`keep_isolates`	Logical. Keep nodes that end up with no edges? Default TRUE, matching `filter_edges`.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` without those edges, or the input format when `keep_format = TRUE`. Named pairs that carry no edge are reported in a `cograph_no_such_edge` warning.

See Also

add_edges, filter_edges, remove_isolates

Examples

```
remove_edges(regulation_net, from = "Plan", to = "Monitor")
```

---

remove_isolates *Remove Isolated Nodes*

Description

Removes every node with no edges. Edge filters such as filter_edges keep all nodes, and this function removes the isolates that such a filter leaves behind.

Usage

```
remove_isolates(x, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input: cograph_network, matrix, igraph, network, tna, or an edge-list data frame.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network with the isolated nodes removed (or the input format when keep_format = TRUE). The remaining nodes keep their order, and edge indices are remapped to the new node numbering.

See Also

filter_edges, split_components, filter_nodes

Examples

```
remove_isolates(threshold_edges(regulation_net, minimum = 0.3))
```

remove_nodes *Remove Nodes from a Network*

Description

Deletes nodes and every edge incident to them. A node that is not in the network raises a `cograph_bad_selection` error.

Usage

```
remove_nodes(x, nodes, keep_format = FALSE, directed = NULL)
```

Arguments

`x`	Network input.
`nodes`	Node labels or indices to remove.
`keep_format`	Logical. Return the input format when TRUE.
`directed`	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` without those nodes and without any edge that touched them, or the input format when `keep_format = TRUE`.

See Also

`add_nodes`, `filter_nodes`, `remove_isolates`

Examples

```
remove_nodes(regulation_net, nodes = "Share")
```

rename_nodes *Rename Nodes*

Description

Rename Nodes

Usage

```
rename_nodes(x, from, to = NULL, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
from	Character vector of current labels, or a named character vector mapping old label to new (in which case to is not used).
to	Character vector of new labels, the same length as from.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network with the renamed nodes, or the input format when keep_format = TRUE. Labels not named in from are unchanged. A cograph_bad_selection error is raised when from names a node that is not in the network or when the renaming would give two nodes the same label.

See Also

reorder_nodes, set_nodes

Examples

```
rename_nodes(regulation_net, from = "Plan", to = "Planning")
```

---

render-ggplot *ggplot2 Conversion*

Description

sn_ggplot() converts a network to a ggplot object.

Value

sn_ggplot() returns a ggplot object representing the network.

Examples

```
sn_ggplot(regulation_net)
```

render-grid *Grid Rendering*

Description

The grid rendering engine. soplot plots a network with grid graphics, and sn_ggplot converts a network into a ggplot2 object.

Value

See individual functions: soplot returns a cograph_network object invisibly; sn_ggplot returns a ggplot2 object.

Examples

```
soplot(regulation_net)
```

reorder_nodes *Reorder the Nodes of a Network*

Description

Changes the order in which the nodes are stored, which is the order in which plotting functions place them. The edges and their weights are unchanged.

Usage

```
reorder_nodes(x, order, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
order	Node labels or indices, in the wanted order, or one of "label", "degree" or "strength". "label" sorts alphabetically, and the two measures sort in decreasing order. A vector that does not name every node exactly once raises a cograph_bad_selection error.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network with the nodes in the requested order and edge indices remapped, or the input format when keep_format = TRUE.

See Also

rename_nodes, select_nodes

Examples

```
reorder_nodes(regulation_net, order = "degree")
```

---

reverse_edges *Reverse Edge Direction*

Description

Swaps the endpoints of every edge, which transposes the weight matrix. In a transition network the reversed arcs show where each transition came from. Additional edge columns are kept.

Usage

```
reverse_edges(x, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A cograph_network with every edge reversed, or the input format when keep_format = TRUE. An undirected network is returned unchanged, with a cograph_no_effect warning.

See Also

to_directed, to_undirected

Examples

```
reverse_edges(regulation_net)
```

`rich_club` *Rich Club Coefficient*

Description

Computes the rich club curve across all prominence thresholds. The unweighted coefficient (Colizza et al. 2006) measures the density of ties among prominent nodes. The weighted coefficient (Opsahl et al. 2008) measures whether prominent nodes concentrate the strongest ties of the network among themselves. A directed network is converted to an undirected one before the computation, with the weights of reciprocal edges summed, and self-loops are removed.

Usage

```
rich_club(
  x,
  rich = c("k", "s"),
  weighted = TRUE,
  normalized = TRUE,
  n_random = 100,
  directed = NULL,
  seed = NULL,
  digits = NULL,
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`rich`	Character. Prominence definition: `"k"` (degree, default) or `"s"` (strength, the weighted degree).
`weighted`	Logical. If TRUE (default), compute the weighted rich club coefficient. If FALSE, compute the unweighted version (density among rich nodes).
`normalized`	Logical. If TRUE (default), normalize against degree-preserving random graphs and report the null distribution. The null graphs are generated with `igraph::sample_degseq()`, which fixes the degree sequence. For a weighted rich club the observed edge weights are also reshuffled across the null edges, following Opsahl et al. (2008). A null graph that `sample_degseq()` cannot draw is left out with a warning of class `"cograph_null_draw_failed"` that reports how many draws failed.
`n_random`	Integer. Number of random graphs for normalization. Default 100.
`directed`	Logical or NULL. Default NULL (auto-detect).
`seed`	Integer or NULL. Random seed for the null graphs. When a seed is given, the caller's random state is restored on exit. Default NULL.
`digits`	Integer or NULL. Round all numeric columns, including `threshold`. Default NULL.
`...`	Passed to `to_igraph`, which takes no further arguments, so any argument supplied here raises an error.

Details

The unweighted coefficient is $\phi(k) = 2E_{>k}/(N_{>k}(N_{>k}-1))$, where $N_{>k}$ is the number of nodes with prominence above k and $E_{>k}$ the number of edges among them.

The weighted coefficient is $\phi^w(k) = W_{>k}/\sum_{l=1}^{E_{>k}} w_l^{ranked}$, where $W_{>k}$ is the total weight of the edges among the rich nodes and w_l^{ranked} is the l-th largest edge weight in the network.

The normalized coefficient is $\phi_{norm} = \phi_{obs}/\bar{\phi}_{rand}$. A value above 1 indicates rich club ordering beyond what the degree sequence alone explains.

Value

A data frame with class `"cograph_rich_club"`, one row per prominence threshold at which at least two nodes are rich, and columns:

threshold The prominence cut-off. Nodes with prominence strictly greater than this value form the club. Thresholds range over the observed prominence values excluding the maximum.

n_rich Number of club members at that threshold.

phi Observed rich club coefficient.

phi_norm, phi_rand, ci_lo, ci_hi Present only when `normalized = TRUE`. They hold the observed coefficient divided by the null mean, the null mean itself, and the 2.5\ quantiles of the null distribution.

The data frame has zero rows for graphs that are too small, complete, or regular for any threshold to yield a club. The arguments `rich`, `weighted`, `normalized` and the original input (`"network"`) are stored as attributes.

Printing and plotting

Printing the result shows the prominence definition, the weighting and normalization settings, the number of thresholds with `phi_norm > 1` and the first ten rows of the table. `plot()` on the result is documented in `plot-results`.

References

Opsahl, T., Colizza, V., Panzarasa, P. & Ramasco, J.J. (2008). Prominence and control: The weighted rich-club effect. *Physical Review Letters*, 101, 168702.

Colizza, V., Flammini, A., Serrano, M.A. & Vespignani, A. (2006). Detecting rich-club ordering in complex networks. *Nature Physics*, 2, 110-115.

See Also

`rich_club_local`, `robustness`, `centrality`

Examples

```
rich_club(regulation_net, n_random = 20, seed = 1)
```

`rich_club_local` *Local Rich Club Score*

Description

Computes, for each node, whether its strongest ties go to prominent nodes. The score is the mean weight of the node's ties to prominent neighbors divided by the mean weight of all its ties. A score above 1 means that the node's ties to prominent nodes are stronger than its average tie. A directed network is converted to an undirected one before the computation, with the weights of reciprocal edges summed, and self-loops are removed.

Usage

```
rich_club_local(
  x,
  prominence = NULL,
  rich = c("k", "s"),
  directed = NULL,
  digits = NULL,
  sort_by = "score",
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`prominence`	Which nodes are prominent. Either a logical or 0/1 vector with one element per node (TRUE or 1 marks a prominent node), or a single number used as a threshold, in which case nodes with degree or strength strictly greater than it are prominent. If NULL (default), nodes with degree or strength strictly above the median are prominent.
`rich`	Character. `"k"` (degree, default) or `"s"` (strength). Used when `prominence` is NULL or a threshold.
`directed`	Logical or NULL. Default NULL (auto-detect).
`digits`	Integer or NULL. Round scores. Default NULL.
`sort_by`	Character or NULL. Column to sort by in descending order, `"score"` (default) or `"node"`. Any other value, and NULL, keep the node order.
`...`	Passed to `to_igraph`, which takes no further arguments, so any argument supplied here raises an error.

Details

For each node i, $r_i = \bar{w}_{i,rich} / \bar{w}_i$, where both means are taken over the ties of i with positive weight.

Value

A plain data frame with one row per node and columns `node` (node label) and `score`. A node with no neighbors or no prominent neighbor scores 1.

References

Opsahl, T., Colizza, V., Panzarasa, P. & Ramasco, J.J. (2008). Prominence and control: The weighted rich-club effect. *Physical Review Letters*, 101, 168702.

See Also

`rich_club`, `centrality`

Examples

```
cograph::rich_club_local(regulation_net)
```

`robustness` *Network Robustness Analysis*

Description

Performs a targeted attack or random failure analysis on a network and computes the size of the largest (weakly) connected component after each vertex or edge removal.

In a targeted attack, vertices are ranked by degree or betweenness centrality (edges by edge betweenness) and removed from highest to lowest. In a random failure analysis, vertices or edges are removed in random order.

Usage

```
robustness(
  x,
  type = c("vertex", "edge"),
  measure = c("betweenness", "degree", "random"),
  strategy = c("sequential", "static"),
  n_iter = 1000,
  mode = "all",
  seed = NULL,
  ...
)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
type	Character string; either "vertex" or "edge" removals. Default: "vertex"
measure	Character string; rank by "betweenness", "degree", or remove in "random" order. Default: "betweenness". "degree" is not available for edge removals and raises an error.
strategy	Character string. "sequential" (default) recomputes the centrality after each removal. "static" computes the centrality once on the original network and removes elements in that fixed order, as in brainGraph. The argument applies to targeted attacks only.
n_iter	Integer; number of random removal sequences averaged when measure = "random". Default: 1000.
mode	Degree mode for directed networks: "all", "in", or "out". Default "all". Used only when measure = "degree".
seed	Random seed for reproducibility. Default NULL.
...	Passed to to_igraph, whose only other argument is directed; anything else raises an "unused argument" error.

Details

A betweenness attack removes the vertices or edges that lie on the most shortest paths, which bridge different regions of the network. A degree attack removes the most connected vertices first. Random failure removes vertices or edges in random order and averages the component sizes over n_iter sequences. Betweenness is computed with igraph, which treats edge weights as distances.

The sequential strategy updates the ranking after every removal, so the attack follows the vertices that become new bridges or hubs. The static strategy ranks once on the original network, as in Albert et al. (2000).

Scale-free networks are typically robust to random failures and vulnerable to targeted attacks. Random networks degrade more uniformly.

Value

A data frame (class "cograph_robustness") with one row per removal step, from zero removed through all removed (n + 1 rows, where n is the number of vertices or edges), and columns:

removed_pct Fraction of vertices/edges removed (0 to 1)

comp_size Size of largest component after removal (averaged over n_iter runs when measure = "random")

comp_pct Ratio of component size to original maximum

measure The measure argument: "betweenness", "degree", or "random"

type A label for the analysis, one of "Targeted vertex attack", "Targeted edge attack", "Random vertex removal" or "Random edge removal"

The last row always has comp_size = 0. The original number of vertices or edges ("n_original") and the original largest-component size ("orig_max") are stored as attributes.

References

Albert, R., Jeong, H., & Barabasi, A.L. (2000). Error and attack tolerance of complex networks. *Nature*, 406, 378-381. doi:10.1038/35019019

See Also

plot_robustness, robustness_auc

Examples

```
robustness(regulation_net, measure = "betweenness")
```

---

robustness_auc *Calculate Area Under Robustness Curve (AUC)*

Description

Computes the area under the robustness curve (comp_pct against removed_pct) by trapezoidal integration. A higher AUC indicates a more robust network. The maximum AUC is 1.

Usage

```
robustness_auc(x)
```

Arguments

x	A robustness result from robustness, or any data frame with columns removed_pct and comp_pct. Other input raises an error.

Value

A single numeric AUC value between 0 and 1.

Examples

```
robustness_auc(robustness(regulation_net, measure = "degree"))
```

robustness_summary *Summary of Robustness Analysis*

Description

Summarizes robustness results across attack strategies.

Usage

```
robustness_summary(..., x = NULL, measures = NULL, n_iter = 1000)
```

Arguments

...	Robustness results from robustness. When these are supplied, x is ignored.
x	Network on which vertex robustness is computed for each of measures. Used when ... is empty.
measures	Measures to compute when x is supplied. Default NULL uses c("betweenness", "degree", "random").
n_iter	Iterations for random removal. Default 1000.

Value

A data frame with one row per supplied (or computed) robustness result and columns measure, auc (area under the robustness curve), critical_50 (fraction removed when the largest component first falls below 50\ same at 10\ crossed. All numeric columns are rounded to 4 decimal places. Supplying neither results nor x raises an error.

Examples

```
robustness_summary(x = regulation_net, measures = c("degree", "random"), n_iter = 10)
```

select_bridges *Select Bridge Edges*

Description

Select edges whose removal would disconnect the graph.

Usage

```
select_bridges(
  x,
  ...,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
...	Additional filter expressions.
keep_isolates	Keep nodes that end up with no edges? Default TRUE.
keep_format	Keep input format? Default FALSE.
directed	Auto-detect if NULL.

Value

A cograph_network with bridge edges only.

See Also

select_edges, select_nodes

Examples

```
strong <- filter_edges(regulation_net, weight > 0.3, keep_isolates = FALSE)
select_bridges(strong, keep_isolates = FALSE)
```

---

select_component *Select Connected Component*

Description

Select nodes belonging to a specific connected component.

Usage

```
select_component(
  x,
  which = "largest",
  ...,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
which	Component selection:
	"largest" (default) The largest connected component
	Integer Component by ID
	Character Component containing the named node

...	Additional filter expressions to apply after component selection.
keep_edges	How to handle edges. Default "internal".
keep_format	Logical. Keep input format? Default FALSE.
directed	Logical or NULL. Auto-detect if NULL.

Value

A cograph_network with nodes in the selected component.

See Also

select_nodes, select_neighbors

Examples

```
select_component(regulation_net, which = "largest")
```

---

select_edges *Select Edges*

Description

Selects edges by filter expressions, by node sets, by rank on a metric, or by structural properties such as bridges, communities and reciprocity.

Usage

```
select_edges(
  x,
  ...,
  top = NULL,
  by = "weight",
  involving = NULL,
  between = NULL,
  bridges_only = FALSE,
  mutual_only = FALSE,
  community = "louvain",
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL,
  .keep_isolates = NULL
)
```

Arguments

x	Network input: cograph_network, matrix, igraph, network, or tna object.
...	Filter expressions using edge columns or computed metrics. Available variables: **Edge columns** from, to, weight, plus any custom **Computed metrics** abs_weight, from_degree, to_degree, from_strength, to_strength, edge_betweenness, weight_rank **Predicates** is_bridge, is_mutual (alias is_reciprocal), is_loop, is_multiple, same_community **Endpoint labels** from_label, to_label, from_community, to_community
top	Integer. Select top N edges by a metric.
by	Character. Metric for top selection. Default "weight". Options: "weight", "abs_weight", "edge_betweenness", "from_degree", "to_degree", "from_strength", "to_strength", "weight_rank".
involving	Character or integer. Select edges involving these nodes (by name or index). An edge is selected if either endpoint matches.
between	List of two character/integer vectors. Select edges between two node sets. Example: between = list(c("A", "B"), c("C", "D")).
bridges_only	Logical. Select only bridge edges (edges whose removal disconnects the graph). Default FALSE.
mutual_only	Logical. For directed networks, select only mutual (reciprocated) edges. Default FALSE.
community	Character. Community detection method for same_community variable. One of "louvain", "walktrap", "fast_greedy", "label_prop", "infomap", "leiden". Default "louvain".
keep_isolates	Logical. Keep nodes that end up with no edges? Default TRUE, matching igraph::delete_edges() and tidygraph: filtering edges does not remove nodes. Set FALSE to drop them, or call remove_isolates() afterwards.
keep_format	Logical. If TRUE, matrix, igraph, statnet network and tna inputs are returned in that format. Default FALSE returns cograph_network.
directed	Logical or NULL. If NULL (default), auto-detect.
.keep_isolates	Deprecated. Use keep_isolates.

Details

Selection criteria are combined with AND logic, so an edge is selected only when it satisfies all of them. The top ranking is applied to the edges that pass involving, between, bridges_only and mutual_only, and the filter expressions in ... are applied afterwards. For example, select_edges(x, top = 10, involving = "A") selects the 10 strongest edges among those involving node A.

Only the edge metrics referenced in expressions or required by a selection criterion are computed.

Value

A cograph_network object with selected edges. If keep_format = TRUE, matrix, igraph, statnet network and tna inputs are converted back to that type. Nodes left without edges are kept and reported in a cograph_isolates_created warning, unless keep_isolates = FALSE.

See Also

filter_edges, select_nodes, select_bridges, select_top_edges

Examples

```
select_edges(regulation_net, top = 5, keep_isolates = FALSE)
```

---

select_edges_between *Select Edges Between Node Sets*

Description

Select edges connecting two specified node sets.

Usage

```
select_edges_between(
  x,
  set1,
  set2,
  ...,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
set1	Character or integer. First node set (names or indices).
set2	Character or integer. Second node set (names or indices).
...	Additional filter expressions.
keep_isolates	Keep nodes that end up with no edges? Default TRUE.
keep_format	Keep input format? Default FALSE.
directed	Auto-detect if NULL.

Value

A cograph_network with edges between the two node sets.

See Also

select_edges, select_edges_involving

Examples

```
select_edges_between(regulation_net, set1 = c("Plan", "Monitor"),
                     set2 = c("Adapt", "Reflect"), keep_isolates = FALSE)
```

`select_edges_involving` *Select Edges Involving Nodes*

Description

Select edges where at least one endpoint is in the specified node set.

Usage

```
select_edges_involving(
  x,
  nodes,
  ...,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`nodes`	Character or integer. Node names or indices.
`...`	Additional filter expressions.
`keep_isolates`	Keep nodes that end up with no edges? Default TRUE.
`keep_format`	Keep input format? Default FALSE.
`directed`	Auto-detect if NULL.

Value

A cograph_network with edges involving the specified nodes.

See Also

`select_edges`, `select_edges_between`

Examples

```
select_edges_involving(regulation_net, nodes = "Plan", keep_isolates = FALSE)
```

`select_k_core` *Select the k-Core of a Network*

Description

The k-core is the maximal subgraph in which every node has degree at least `k`, found by repeatedly removing nodes of degree below `k`. Degree is the total degree, which is in-degree plus out-degree in a directed network. A self-loop adds 2 to the degree of its node.

Usage

```
select_k_core(x, k, keep_format = FALSE, directed = NULL)
```

Arguments

`x`	Network input.
`k`	A single non-negative whole number. The core number.
`keep_format`	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a `cograph_network` with a `cograph_no_format_roundtrip` warning. Default FALSE returns a `cograph_network`.
`directed`	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A `cograph_network` holding the k-core, or the input format when `keep_format = TRUE`. When no node reaches coreness `k`, the result is an empty network and a warning is raised.

References

Seidman, S. B. (1983). Network structure and minimum degree. *Social Networks*, 5(3), 269–287.

See Also

`select_nodes`, `centrality`

Examples

```
select_k_core(regulation_net, k = 2)
```

`select_neighbors` *Select Node Neighbors (Ego Network)*

Description

Select nodes within a specified distance from focal nodes.

Usage

```
select_neighbors(
  x,
  of,
  order = 1L,
  ...,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`of`	Character or integer. Focal node(s) by name or index.
`order`	Integer. Neighborhood order (1 = direct neighbors). Default 1.
`...`	Additional filter expressions to apply after neighborhood selection.
`keep_edges`	How to handle edges. Default "internal".
`keep_format`	Logical. Keep input format? Default FALSE.
`directed`	Logical or NULL. Auto-detect if NULL.

Value

A cograph_network with nodes in the neighborhood.

See Also

`select_nodes`, `select_component`

Examples

```
select_neighbors(regulation_net, of = "Plan")
```

`select_nodes` *Select Nodes*

Description

Selects nodes by name, index, rank on a centrality measure, neighborhood, connected component, or filter expressions on node columns, centrality measures and structural variables.

Usage

```
select_nodes(
  x,
  ...,
  name = NULL,
  index = NULL,
  top = NULL,
  by = "degree",
  neighbors_of = NULL,
  order = 1L,
  component = NULL,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL,
  .keep_edges = NULL
)
```

Arguments

`x`	Network input: cograph_network, matrix, igraph, network, or tna object.
`...`	Filter expressions using node columns, centrality measures, or global context variables. Centrality measures are computed lazily (only those actually referenced). Available variables: **Node columns** All columns of the node table, such as `id`, `label`, `name`, `x`, `y` and any custom columns. **Centrality measures** `degree`, `indegree`, `outdegree`, `strength`, `instrength`, `outstrength`, `betweenness`, `closeness`, `eigenvector`, `pagerank`, `hub`, `authority`, `coreness`. Any other measure `centrality()` computes can be named too; see `list_centralities()`. **Global context** `component`, `component_size`, `is_largest_component`, `neighborhood_size`, `k_core`, `is_articulation`, `is_bridge_endpoint` **Predicates** `is_isolated`, `is_source`, `is_sink`, `is_leaf`, `is_cut`, `local_transitivity`, `local_triangles`
`name`	Character vector. Select nodes by name/label.
`index`	Integer vector. Select nodes by index (1-based).
`top`	Integer. Select top N nodes by centrality measure.

by	Character. Centrality measure for top selection. Default "degree".
neighbors_of	Character or integer. Select neighbors of these nodes (by name or index).
order	Integer. Neighborhood order (1 = direct neighbors, 2 = neighbors of neighbors, etc.). Default 1.
component	Selection mode for connected components: "largest" Select nodes in the largest connected component **Integer** Select nodes in component with this ID **Character** Select component containing node with this name
keep_edges	How to handle edges. One of: "internal" (default) Keep only edges between remaining nodes "none" Remove all edges
keep_format	Logical. If TRUE, matrix, igraph, statnet network and tna inputs are returned in that format. Default FALSE returns cograph_network.
directed	Logical or NULL. If NULL (default), auto-detect.
.keep_edges	Deprecated. Use keep_edges.

Details

Selection criteria are combined with AND logic, so a node is selected only when it satisfies all of them. The top ranking is applied to the nodes that pass name, index, component and neighbors_of, and the filter expressions in ... are applied afterwards. For example, select_nodes(x, top = 10, component = "largest") selects the 10 highest-degree nodes within the largest component.

Only the centrality measures referenced in expressions or in by are computed.

For networks with negative edge weights, betweenness, closeness and pagerank are undefined and return NA, with a cograph_negative_weights warning.

Value

A cograph_network object with selected nodes. If keep_format = TRUE, matrix, igraph, statnet network and tna inputs are converted back to that type.

See Also

filter_nodes, select_neighbors, select_component, select_top

Examples

```
select_nodes(regulation_net, top = 3, by = "pagerank")
```

select_top *Select Top N Nodes by Centrality*

Description

Select the top N nodes ranked by a centrality measure.

Usage

```
select_top(
  x,
  n,
  by = "degree",
  ...,
  keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
n	Integer. Number of top nodes to select.
by	Character. Centrality measure for ranking: "degree", "indegree", "outdegree", "strength", "instrength", "outstrength", "betweenness", "closeness", "eigenvector", "pagerank", "hub", "authority", "coreness", or the name of any other measure centrality() computes (see list_centralities()). An unknown name raises a cograph_bad_selection error. Default "degree".
...	Additional filter expressions, applied after the top n nodes are selected.
keep_edges	How to handle edges. Default "internal".
keep_format	Logical. Keep input format? Default FALSE.
directed	Logical or NULL. Auto-detect if NULL.

Value

A cograph_network with the top N nodes.

See Also

select_nodes, select_component

Examples

```
select_top(regulation_net, n = 3, by = "pagerank")
```

`select_top_edges` *Select Top N Edges*

Description

Select the top N edges ranked by weight or another metric.

Usage

```
select_top_edges(
  x,
  n,
  by = "weight",
  ...,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`n`	Integer. Number of top edges to select.
`by`	Character. Metric for ranking. One of: `"weight"` (default), `"abs_weight"`, `"edge_betweenness"`, `"from_degree"`, `"to_degree"`, `"from_strength"`, `"to_strength"`, `"weight_rank"`. Any other name raises a `cograph_bad_selection` error.
`...`	Additional filter expressions, applied after the top `n` edges are selected.
`keep_isolates`	Keep nodes that end up with no edges? Default TRUE.
`keep_format`	Keep input format? Default FALSE.
`directed`	Auto-detect if NULL.

Value

A cograph_network with the top N edges.

See Also

`select_edges`, `select_top`

Examples

```
select_top_edges(regulation_net, n = 5, keep_isolates = FALSE)
```

shapes *Node Shapes*

Description

Node shapes are stored in a registry and selected by name through the node_shape argument of splot and soplot or the shape argument of sn_nodes. register_shape() adds a shape given by a drawing function, and get_shape() returns the drawing function of a registered shape. list_shapes() returns the names of all registered shapes. A shape added with register_shape() is a grid shape and is used by soplot() only. splot() plots its built-in shapes and SVG shapes. register_svg_shape() adds a shape defined by an SVG file or an inline SVG string, list_svg_shapes() returns the names of the registered SVG shapes, and unregister_svg_shape() removes one. SVG shapes are used by both splot() and soplot(). Registering an existing name replaces that shape for the rest of the session.

Usage

```
register_shape(name, draw_fn)

get_shape(name)

list_shapes()

register_svg_shape(name, svg_source)

list_svg_shapes()

unregister_svg_shape(name)
```

Arguments

name	Character. The name of the shape.
draw_fn	A function that renders the shape. It receives the arguments x, y, size, fill, border_color, border_width, alpha and ... and returns a grid grob.
svg_source	Character. A path to an SVG file or an inline SVG string.

Value

register_shape() and register_svg_shape() return NULL invisibly. get_shape() returns the drawing function, or NULL if no shape has that name. list_shapes() and list_svg_shapes() return a character vector of shape names. unregister_svg_shape() returns TRUE invisibly if the shape was removed and FALSE if it was not found.

Examples

```
list_shapes()
splot(regulation_net, node_shape = "diamond")
```

shortest_paths *Compute Shortest Path Distances*

Description

Computes shortest path distances between nodes in a network. Supports all-pairs, single-source, and point-to-point queries.

Usage

```
shortest_paths(x, from = NULL, to = NULL, weights = NULL, directed = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
from	Character or numeric node identifier(s) for the source. If NULL (default), compute distances from all nodes.
to	Character or numeric node identifier(s) for the target. If NULL (default), compute distances to all nodes.
weights	Edge weight handling: NULL (default) auto-detects from edge attributes, NA forces unweighted distances, or a numeric vector of custom weights.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
...	Not used. Any argument supplied here raises an "unused argument" error.

Details

Distances are computed with `igraph::distances()`. For weighted networks, edge weights are used as distances by default. With `weights = NA`, every edge has unit distance.

igraph also exports a `shortest_paths()` with a different signature and return value; when both packages are attached, qualify the call as `cograph::shortest_paths()`.

Value

Depends on the query:

- If both `from` and `to` are NULL: a full distance matrix (all pairs)
- If `from` is a single node and `to` is NULL: a named numeric vector of distances from that node to all others
- If `from` is multiple nodes and `to` is NULL: a matrix with rows for each source
- If both `from` and `to` are single nodes: a single numeric value
- Otherwise: a matrix of distances between the specified node sets

See Also

k_shortest_paths, network_summary

Examples

```
cograph::shortest_paths(regulation_net, from = "Plan")
```

`simmelian_strength` *Simmelian Strength (Triangle Count per Edge)*

Description

Convenience wrapper around `edge_centrality` that returns only the triangle count per edge, sorted descending.

Usage

```
simmelian_strength(x, top = NULL, directed = NULL, digits = NULL, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`top`	Integer or NULL. Return only the top N edges. Default NULL.
`directed`	Logical or NULL. Default NULL (auto-detect).
`digits`	Integer or NULL. Round numeric columns. Default NULL.
`...`	Additional arguments passed to `edge_centrality`.

Value

A data frame sorted by `triangles` (descending) with columns: `from`, `to`, `weight`, `triangles`.

See Also

`edge_centrality`, `neighborhood_overlap`

Examples

```
cograph::simmelian_strength(regulation_net, top = 5)
```

simplify *Simplify a Network*

Description

Removes self-loops and (where representable) merges duplicate (multi-)edges, similar to `igraph::simplify()`.

Usage

```
simplify(x, remove_loops, remove_multiple, edge_attr_comb, ...)

## S3 method for class 'matrix'
simplify(
  x,
  remove_loops = TRUE,
  remove_multiple = TRUE,
  edge_attr_comb = "mean",
  ...
)

## S3 method for class 'cograph_network'
simplify(
  x,
  remove_loops = TRUE,
  remove_multiple = TRUE,
  edge_attr_comb = "mean",
  ...
)

## S3 method for class 'igraph'
simplify(
  x,
  remove_loops = TRUE,
  remove_multiple = TRUE,
  edge_attr_comb = "mean",
  ...
)

## S3 method for class 'tna'
simplify(
  x,
  remove_loops = TRUE,
  remove_multiple = TRUE,
  edge_attr_comb = "mean",
  ...
)
```

```
## Default S3 method:
simplify(
  x,
  remove_loops = TRUE,
  remove_multiple = TRUE,
  edge_attr_comb = "mean",
  ...
)
```

Arguments

x	Network input (matrix, cograph_network, igraph, tna object).
remove_loops	Logical. Remove self-loops (diagonal entries)? Default TRUE.
remove_multiple	Logical. Merge duplicate edges? Default TRUE. Ignored for matrix and tna inputs (see Details).
edge_attr_comb	How to combine weights of duplicate edges: "sum", "mean" (default), "max", "min", "first", or a custom function. Ignored for matrix and tna inputs.
...	Additional arguments (currently unused).

Details

The extent of simplification depends on the input representation:

- matrix and tna: edges are stored as an n x n weight matrix. Each cell (i, j) is unique by construction, so duplicate-edge merging has no effect and remove_multiple and edge_attr_comb are ignored. Only self-loops (the diagonal) are removed. Duplicate aggregation requires a cograph_network or igraph input.
- cograph_network: duplicate edges in the edge list are merged, and their weights are combined with edge_attr_comb.
- igraph: delegates to igraph::simplify(). The weight attribute is combined with edge_attr_comb and other edge attributes are dropped.

Value

The simplified network, in the same format and class as the input (matrix in / matrix out, cograph_network in / cograph_network out, and so on). The default method raises an error for any other class.

See Also

filter_edges for conditional edge removal, centrality which has its own simplify parameter

Examples

```
# igraph also exports simplify(); qualify the call when both are loaded.
cograph::simplify(cograph(student_interactions), edge_attr_comb = "sum")
```

sn_edges *Set Edge Aesthetics*

Description

Stores edge aesthetics in a network object for plotting with soplot. splot does not read these settings and takes its edge arguments directly.

Usage

```
sn_edges(
  network,
  width = NULL,
  edge_size = NULL,
  esize = NULL,
  edge_width_range = NULL,
  edge_scale_mode = NULL,
  edge_cutoff = NULL,
  cut = NULL,
  color = NULL,
  edge_positive_color = NULL,
  positive_color = NULL,
  edge_negative_color = NULL,
  negative_color = NULL,
  alpha = NULL,
  style = NULL,
  curvature = NULL,
  arrow_size = NULL,
  show_arrows = NULL,
  maximum = NULL,
  width_scale = NULL,
  labels = NULL,
  label_size = NULL,
  label_color = NULL,
  label_position = NULL,
  label_offset = NULL,
  label_bg = NULL,
  label_bg_padding = NULL,
  label_fontface = NULL,
  label_border = NULL,
  label_border_color = NULL,
  label_underline = NULL,
  label_shadow = NULL,
  label_shadow_color = NULL,
  label_shadow_offset = NULL,
  label_shadow_alpha = NULL,
  bidirectional = NULL,
```

```
  loop_rotation = NULL,
  curve_shape = NULL,
  curve_pivot = NULL,
  curves = NULL,
  ci = NULL,
  ci_scale = NULL,
  ci_alpha = NULL,
  ci_color = NULL,
  ci_style = NULL,
  ci_arrows = NULL,
  ci_lower = NULL,
  ci_upper = NULL,
  label_style = NULL,
  label_template = NULL,
  label_digits = NULL,
  label_ci_format = NULL,
  label_p = NULL,
  label_p_digits = NULL,
  label_p_prefix = NULL,
  label_stars = NULL
)
```

Arguments

network	A CographNetwork, cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
width	Edge width. Can be a single value, vector (per-edge), or "weight".
edge_size	Maximum edge size for renderer weight scaling. NULL (default) uses the renderer's edge-width range. Larger values give thicker edges overall.
esize	Deprecated. Use edge_size instead.
edge_width_range	Output width range as c(min, max) for weight-based scaling. If NULL (default), the plotting renderer's default range is used.
edge_scale_mode	Scaling mode for edge weights: "linear" (default), "log" (for wide weight ranges), "sqrt" (moderate compression), or "rank" (equal visual spacing).
edge_cutoff	Optional cutoff for edge emphasis. NULL (default) or 0 disables cutoff handling. The value is stored in the network; the soplot() renderer keeps width scaling continuous and does not change the plot for it.
cut	Deprecated. Use edge_cutoff instead.
color	Edge color. Can be a single color, vector, or "weight" for automatic coloring based on edge weights.
edge_positive_color	Color for positive edge weights.
positive_color	Deprecated. Use edge_positive_color instead.

edge_negative_color
: Color for negative edge weights.

negative_color
: Deprecated. Use edge_negative_color instead.

alpha
: Edge transparency (0-1).

style
: Line style: "solid", "dashed", "dotted", "longdash", "twodash".

curvature
: Edge curvature amount (0 = straight).

arrow_size
: Size of arrow heads for directed networks.

show_arrows
: Logical. Show arrows? Default TRUE for directed networks.

maximum
: Maximum edge weight for scaling width. Weights above this are capped. Similar to qgraph's maximum parameter.

width_scale
: Scale factor for edge widths. Values > 1 make edges thicker, values < 1 make them thinner. Applied after all other width calculations.

labels
: Edge labels. Can be TRUE (show weights), a vector, or column name.

label_size
: Edge label text size.

label_color
: Edge label text color.

label_position
: Position along edge (0 = source, 0.5 = middle, 1 = target).

label_offset
: Perpendicular offset from edge line.

label_bg
: Background color for edge labels (default "white"). Set to NA for transparent.

label_bg_padding
: Padding around label text as proportion of text size (default 0.3).

label_fontface
: Font face: "plain", "bold", "italic", "bold.italic" (default "plain").

label_border
: Border style: NULL (none), "rect", "rounded", "circle" (default NULL).

label_border_color
: Border color for label border (default "gray50").

label_underline
: Logical. Underline the label text? (default FALSE).

label_shadow
: Logical. Enable drop shadow for labels? (default FALSE).

label_shadow_color
: Color for label shadow (default "gray40").

label_shadow_offset
: Offset distance for shadow in points (default 0.5).

label_shadow_alpha
: Transparency for shadow (0-1, default 0.5).

bidirectional
: Logical. Show arrows at both ends of edges?

loop_rotation
: Angle in radians for self-loop direction (default: pi/2 = top).

curve_shape
: Spline tension for curved edges (-1 to 1, default: 0).

curve_pivot
: Pivot position along edge for curve control point (0-1, default: 0.5).

curves
: Curve mode: FALSE (straight edges), "mutual" (only curve reciprocal pairs), or "force" (curve all edges). If NULL, the plotting renderer's default is used.

ci
: Numeric vector of CI widths (0-1 scale). Larger values indicate more uncertainty.

ci_scale	Width multiplier for CI underlay thickness. Default 2.
ci_alpha	Transparency for CI underlay (0-1). Default 0.15.
ci_color	CI underlay color. NA (default) uses main edge color.
ci_style	Line type for CI underlay: 1=solid, 2=dashed, 3=dotted. Default 2.
ci_arrows	Logical: show arrows on CI underlay? Default FALSE.
ci_lower	Numeric vector of lower CI bounds for labels.
ci_upper	Numeric vector of upper CI bounds for labels.
label_style	Preset style: "none", "estimate", "full", "range", "stars".
label_template	Template with placeholders: {est}, {range}, {low}, {up}, {p}, {stars}.
label_digits	Decimal places for estimates in template. Default 2.
label_ci_format	CI format: "bracket" for [low, up] or "dash" for low-up.
label_p	Numeric vector of p-values for edges.
label_p_digits	Decimal places for p-values. Default 3.
label_p_prefix	Prefix for p-values. Default "p=".
label_stars	Stars for labels: character vector, TRUE (compute from p), or numeric (treated as p-values).

Details

Vectorization:

Most aesthetic parameters can be specified as:

- A single value is applied to all edges.
- A vector gives per-edge values and is recycled to the edge count.
- "weight" maps width and color from the edge weights.
- A single string naming an edge column takes the values of that column.

Weight-Based Styling:

When color = "weight", edges are colored by sign:

- Positive weights use edge_positive_color (default: green)
- Negative weights use edge_negative_color (default: red)

When width = "weight", edge widths scale linearly with absolute weight values to the range 0.5 to 3, capped at maximum if set. Explicit widths take precedence over edge_size, edge_width_range and edge_scale_mode.

Edge Label Templates:

For statistical output (e.g., regression coefficients with CIs), use templates:

- label_template = "\{est\}": Show estimate only
- label_template = "\{est\} [\{low\}, \{up\}]": Estimate with CI
- label_template = "\{est\}\{stars\}": Estimate with significance

Preset styles via label_style:

- "estimate": Weight/estimate only

- "full": Estimate + CI in brackets
- "range": CI range only
- "stars": Significance stars

CI Underlays:

Uncertainty is visualized as a wider, semi-transparent edge plotted behind the main edge.

- ci: Vector of CI widths (0-1 scale)
- ci_scale: Width multiplier (default 2)
- ci_alpha: Transparency (default 0.15)

Value

The input as a cograph_network object with the edge aesthetics merged into its stored settings, ready to pipe into further sn_* functions or soplot().

See Also

sn_nodes for node customization, cograph for network creation, soplot for plotting, sn_layout for layout algorithms, sn_theme for visual themes

Examples

```
cograph(regulation_net) |>
  sn_edges(width = "weight", color = "weight") |>
  soplot()
```

---

sn_ggplot *Convert Network to ggplot2*

Description

Builds a ggplot object of a network that can be modified and combined with other ggplot2 layers. Nodes are points with labels and edges are straight segments, with arrows when the network is directed. Edges with positive weights are green and edges with negative weights are red. The plot uses fixed default aesthetics and does not apply styling set with the sn_* functions or a theme.

Usage

```
sn_ggplot(network, title = NULL)
```

Arguments

network	A cograph_network object, matrix, data frame edge list, igraph, statnet network, qgraph or tna object. A network without layout coordinates receives a spring layout computed with seed 42.
title	Optional plot title.

Value

A ggplot object.

Examples

```
sn_ggplot(regulation_net, title = "Regulation")
```

`sn_layout` *Apply Layout to Network*

Description

Apply a layout algorithm to compute node positions.

Usage

```
sn_layout(network, layout, seed = 42, ...)
```

Arguments

`network`	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
`layout`	Layout algorithm name (see Details), a two-letter or full igraph layout name, an igraph layout function, a `CographLayout` object, or a coordinate matrix / data frame with one row per node and `x`, `y` in its first two columns. Anything else is an error.
`seed`	Random seed for deterministic layouts. Default 42. Set NULL for random.
`...`	Additional arguments passed to the layout function.

Details

Built-in Layouts:

`"spring"` Force-directed layout (Fruchterman-Reingold).

`"oval"`/`"ellipse"` Nodes arranged around an ellipse.

`"circle"` Nodes arranged in a circle.

`"groups"` Circular layout with grouped nodes clustered together.

`"grid"` Nodes in a regular grid.

`"random"` Random positions.

`"star"` Central node with others arranged around it.

`"bipartite"` Two-column layout for bipartite networks.

`"gephi"`/`"gephi_fr"` Gephi-style force-directed layout.

igraph Layouts:

Two-letter codes for igraph layouts: "kk" (Kamada-Kawai), "fr" (Fruchterman-Reingold), "drl", "mds", "ni" (nicely), "tr" (tree), "ci" (circle), etc.

igraph layout functions and full igraph layout names such as `"layout_with_kk"` are also accepted.

Value

The cograph_network with node coordinates in the x and y columns of its node table.

See Also

cograph for network creation, sn_nodes for node customization, sn_edges for edge customization, sn_theme for visual themes, splot and soplot for plotting

Examples

```
cograph(regulation_net) |> sn_layout("circle") |> splot()
```

---

sn_nodes *Set Node Aesthetics*

Description

Sets the visual properties of the nodes in a network plot. Every argument except network defaults to NULL, which leaves the current setting unchanged; the defaults stated below are the values used at render time.

Usage

```
sn_nodes(
  network,
  size = NULL,
  shape = NULL,
  node_svg = NULL,
  svg_preserve_aspect = NULL,
  fill = NULL,
  border_color = NULL,
  border_width = NULL,
  alpha = NULL,
  label_size = NULL,
  label_color = NULL,
  label_position = NULL,
  show_labels = NULL,
  pie_values = NULL,
  pie_colors = NULL,
  pie_border_width = NULL,
  donut_fill = NULL,
  donut_values = NULL,
  donut_color = NULL,
  donut_colors = NULL,
  donut_border_width = NULL,
  donut_inner_ratio = NULL,
  donut_bg_color = NULL,
```

```
  donut_shape = NULL,
  donut_show_value = NULL,
  donut_value_size = NULL,
  donut_value_color = NULL,
  donut_value_fontface = NULL,
  donut_value_fontfamily = NULL,
  donut_value_digits = NULL,
  donut_value_prefix = NULL,
  donut_value_suffix = NULL,
  donut_value_format = NULL,
  donut2_values = NULL,
  donut2_colors = NULL,
  donut2_inner_ratio = NULL,
  label_fontface = NULL,
  label_fontfamily = NULL,
  label_hjust = NULL,
  label_vjust = NULL,
  label_angle = NULL,
  node_names = NULL
)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
size	Node size. Can be a single value, vector (per-node), or column name.
shape	Node shape. One of the shapes listed by `list_shapes()` (for example "circle", "square", "triangle", "diamond", "pentagon", "hexagon", "ellipse", "heart", "star", "pie", "donut", "cross", "rectangle"), or a custom SVG shape registered with `register_svg_shape()`.
node_svg	Custom SVG for the node shape: a path to an SVG file or an inline SVG string. It is registered as a temporary shape and replaces `shape`.
svg_preserve_aspect	Logical: maintain the SVG aspect ratio? The value is stored with the network, but the renderer always preserves the aspect ratio, so it has no effect at present.
fill	Node fill color. Can be a single color, vector, or column name.
border_color	Node border color.
border_width	Node border width.
alpha	Node opacity in [0, 1]. Values outside this range raise an error.
label_size	Label text size.
label_color	Label text color.
label_position	Label position: "center" (render default), "above", "below", "left" or "right".
show_labels	Logical. Show node labels? Default TRUE.
pie_values	For pie shape: list or matrix of values for pie segments. Each element corresponds to a node and contains values for its segments.

pie_colors For pie shape: colors for pie segments.

pie_border_width
Border width for pie chart nodes.

donut_fill For donut shape: numeric value (0-1) specifying fill proportion. A value of 0.5 fills half of the ring and 1 fills the whole ring. Can be a single value (all nodes) or vector (per-node values).

donut_values Deprecated. Use donut_fill. Ignored when donut_fill is supplied.

donut_color For donut shape: fill color(s) for the donut ring. Single color sets fill for all nodes. Two colors set fill and background for all nodes. More than 2 colors set per-node fill colors (recycled to n_nodes). The render default is a "maroon" fill on a "gray90" background.

donut_colors Deprecated. Use donut_color. Ignored when donut_color is supplied.

donut_border_width
Border width for donut chart nodes.

donut_inner_ratio
For donut shape: inner radius ratio (0-1). Default 0.5.

donut_bg_color For donut shape: background color for unfilled portion.

donut_shape For donut: base shape for ring ("circle", "square", "hexagon", "triangle", "diamond", "pentagon"). Default NULL, which inherits the ring shape from the node's own shape (hexagon nodes get hexagon donuts); set it explicitly to override that for every node.

donut_show_value
For donut shape: show value in center? Default FALSE.

donut_value_size
For donut shape: font size for center value.

donut_value_color
For donut shape: color for center value text.

donut_value_fontface
For donut shape: font face for center value ("plain", "bold", "italic", "bold.italic"). Default "bold".

donut_value_fontfamily
For donut shape: font family for center value ("sans", "serif", "mono"). Default "sans".

donut_value_digits
For donut shape: decimal places for value display. Default 2.

donut_value_prefix
For donut shape: text before value (e.g., "$"). Default "".

donut_value_suffix
For donut shape: text after value (e.g., "%"). Default "".

donut_value_format
For donut shape: a function that formats the center value. It replaces donut_value_digits. A non-function raises an error.

donut2_values For double donut: list of values for inner donut ring.

donut2_colors For double donut: colors for inner donut ring segments.

`donut2_inner_ratio`
: For double donut: inner radius ratio for inner donut ring. Default 0.4.

`label_fontface`
: Font face for node labels: "plain", "bold", "italic", "bold.italic". Default "plain".

`label_fontfamily`
: Font family for node labels: "sans", "serif", "mono", or system font. Default "sans".

`label_hjust`
: Horizontal justification for node labels (0=left, 0.5=center, 1=right). Default 0.5.

`label_vjust`
: Vertical justification for node labels (0=bottom, 0.5=center, 1=top). Default 0.5.

`label_angle`
: Text rotation angle in degrees for node labels. Default 0.

`node_names`
: Alternative names for legend (separate from display labels).

Details

Vectorization:

The arguments `size`, `shape`, `fill`, `border_color`, `border_width`, `alpha`, `label_size`, `label_color`, `label_position` and `node_names` accept a single value, which is applied to all nodes, or a per-node vector. A vector of another length is recycled to the number of nodes without a warning. A single string that matches a column of the node data frame is replaced by that column.

Donut Charts:

A donut node shows one proportion in [0, 1] per node. `donut_fill` sets the proportion, `donut_color` the ring color, `donut_shape` the base shape of the ring, and `donut_show_value = TRUE` prints the value in the center.

Value

The input as a `cograph_network` object, with the supplied settings merged into its node aesthetics. It can be piped to further customization or plotting functions.

See Also

`sn_edges` for edge customization, `cograph` for network creation, `splot` and `soplot` for plotting, `sn_layout` for layout algorithms, `sn_theme` for visual themes

Examples

```
cograph(regulation_net) |>
  sn_nodes(fill = "steelblue", shape = "square") |>
  splot()
```

sn_palette *Apply Color Palette to Network*

Description

Apply a color palette for node and/or edge coloring.

Usage

```
sn_palette(network, palette, target = "nodes", by = NULL)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
palette	Palette name (see list_palettes) or a function that takes n and returns n colors.
target	What to apply the palette to: "nodes" (default), "edges", or "both". For edges, the first two palette colors become the colors of positive and negative edges.
by	Name of a node-table column whose values are mapped to palette colors. When by is NULL or not a column of the node table, every node gets the first palette color.

Details

Available Palettes:

Use `list_palettes()` to see all available palettes. Common options:

`"viridis"` Perceptually uniform, colorblind-friendly.

`"colorblind"` Optimized for color vision deficiency.

`"pastel"` Soft, muted colors.

`"blues"` Blue sequential palette.

`"reds"` Red sequential palette.

`"diverging"` Blue-white-red diverging palette.

Value

The cograph_network with the node fill colors and edge colors stored in its node_aes and edge_aes elements.

See Also

cograph for network creation, sn_theme for visual themes, sn_nodes for node customization, list_palettes to see available palettes, splot and soplot for plotting

Examples

```
cograph(regulation_net) |> sn_palette("viridis") |> splot()
```

sn_save *Save Network Visualization*

Description

Save a Cograph network visualization to a file.

Usage

```
sn_save(network, filename, width = 7, height = 7, dpi = 300, title = NULL, ...)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
filename	Output filename. Format is detected from the extension; one of .pdf, .png, .svg, .jpeg/.jpg, .tiff, .eps/.ps.
width	Width in inches (default 7).
height	Height in inches (default 7).
dpi	Resolution for raster formats (default 300).
title	Optional plot title.
...	Additional arguments passed to the graphics device.

Value

The output filename, invisibly.

Examples

```
sn_save(cograph(regulation_net),
  filename = file.path(tempdir(), "network.pdf"))
```

sn_save_ggplot *Save as ggplot2*

Description

Save network as a ggplot2 object to file using ggsave.

Usage

```
sn_save_ggplot(
  network,
  filename,
  width = 7,
  height = 7,
  dpi = 300,
  title = NULL,
  ...
)
```

Arguments

network	A cograph_network object.
filename	Output filename. Format is detected from the extension by ggplot2::ggsave().
width	Width in inches (default 7).
height	Height in inches (default 7).
dpi	Resolution for raster formats (default 300).
title	Optional plot title.
...	Additional arguments passed to ggsave.

Value

The output filename, invisibly.

Examples

```
sn_save_ggplot(cograph(regulation_net),
  filename = file.path(tempdir(), "network.pdf"))
```

---

sn_theme *Apply Theme to Network*

Description

Apply a visual theme to the network.

Usage

```
sn_theme(network, theme, ...)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
theme	Theme name (string) or CographTheme object.
...	Theme parameters to override, such as background, node_fill or edge_color. An unknown parameter name raises an error.

Details

Available Themes:

"classic" White background, blue nodes and gray edges.

"dark" Dark background with bright nodes, for presentations.

"minimal" Subtle styling with thin edges and muted colors.

"colorblind" Optimized for color vision deficiency.

"gray"/"grey" Black and white theme suitable for print.

"viridis" Perceptually uniform colors.

"nature" Nature-inspired colors.

Use list_themes() to see all available themes.

Value

The cograph_network with the theme stored in its theme element.

See Also

cograph for network creation, sn_palette for color palettes, sn_nodes for node customization, sn_edges for edge customization, list_themes to see available themes, splot and soplot for plotting

Examples

```
cograph(regulation_net) |> sn_theme("dark") |> splot()
```

---

soplot *Plot Cograph Network*

Description

Plots a network with grid graphics. Node and edge aesthetics can be passed as arguments or set beforehand with sn_nodes and sn_edges.

Usage

```
soplot(
  network,
  title = NULL,
  title_size = 14,
  margins = c(0.05, 0.05, 0.1, 0.05),
  layout_margin = 0.15,
  newpage = TRUE,
  background = "white",
  layout = NULL,
  theme = NULL,
  seed = 42,
  labels = NULL,
  threshold = NULL,
  maximum = NULL,
  node_size = NULL,
  node_shape = NULL,
  node_fill = NULL,
  node_border_color = NULL,
  node_border_width = NULL,
  node_alpha = NULL,
  label_size = NULL,
  label_color = NULL,
  label_position = NULL,
  show_labels = NULL,
  pie_values = NULL,
  pie_colors = NULL,
  pie_border_width = NULL,
  donut_values = NULL,
  donut_border_width = NULL,
  donut_inner_ratio = NULL,
  donut_bg_color = NULL,
  donut_show_value = NULL,
  donut_value_size = NULL,
  donut_value_color = NULL,
  donut_fill = NULL,
  donut_color = NULL,
  donut_colors = NULL,
  donut_shape = "circle",
  donut_value_fontface = "bold",
  donut_value_fontfamily = "sans",
  donut_value_digits = 2,
  donut_value_prefix = "",
  donut_value_suffix = "",
  donut2_values = NULL,
  donut2_colors = NULL,
  donut2_inner_ratio = 0.4,
  edge_width = NULL,
```

```
  edge_size = NULL,
  esize = NULL,
  edge_width_range = NULL,
  edge_scale_mode = "linear",
  edge_cutoff = NULL,
  cut = NULL,
  edge_width_scale = NULL,
  edge_color = NULL,
  edge_alpha = NULL,
  edge_style = NULL,
  curvature = NULL,
  arrow_size = NULL,
  show_arrows = NULL,
  edge_positive_color = NULL,
  positive_color = NULL,
  edge_negative_color = NULL,
  negative_color = NULL,
  edge_duplicates = NULL,
  edge_labels = NULL,
  edge_label_size = NULL,
  edge_label_color = NULL,
  edge_label_position = NULL,
  edge_label_offset = NULL,
  edge_label_bg = NULL,
  edge_label_fontface = NULL,
  edge_label_border = NULL,
  edge_label_border_color = NULL,
  edge_label_underline = NULL,
  bidirectional = NULL,
  loop_rotation = NULL,
  curve_shape = NULL,
  curve_pivot = NULL,
  curves = NULL,
  node_names = NULL,
  legend = FALSE,
  legend_position = "topright",
  scaling = "default",
  weight_digits = 2
)

sn_render(
  network,
  title = NULL,
  title_size = 14,
  margins = c(0.05, 0.05, 0.1, 0.05),
  layout_margin = 0.15,
  newpage = TRUE,
  background = "white",
```

```
  layout = NULL,
  theme = NULL,
  seed = 42,
  labels = NULL,
  threshold = NULL,
  maximum = NULL,
  node_size = NULL,
  node_shape = NULL,
  node_fill = NULL,
  node_border_color = NULL,
  node_border_width = NULL,
  node_alpha = NULL,
  label_size = NULL,
  label_color = NULL,
  label_position = NULL,
  show_labels = NULL,
  pie_values = NULL,
  pie_colors = NULL,
  pie_border_width = NULL,
  donut_values = NULL,
  donut_border_width = NULL,
  donut_inner_ratio = NULL,
  donut_bg_color = NULL,
  donut_show_value = NULL,
  donut_value_size = NULL,
  donut_value_color = NULL,
  donut_fill = NULL,
  donut_color = NULL,
  donut_colors = NULL,
  donut_shape = "circle",
  donut_value_fontface = "bold",
  donut_value_fontfamily = "sans",
  donut_value_digits = 2,
  donut_value_prefix = "",
  donut_value_suffix = "",
  donut2_values = NULL,
  donut2_colors = NULL,
  donut2_inner_ratio = 0.4,
  edge_width = NULL,
  edge_size = NULL,
  esize = NULL,
  edge_width_range = NULL,
  edge_scale_mode = "linear",
  edge_cutoff = NULL,
  cut = NULL,
  edge_width_scale = NULL,
  edge_color = NULL,
  edge_alpha = NULL,
```

```
  edge_style = NULL,
  curvature = NULL,
  arrow_size = NULL,
  show_arrows = NULL,
  edge_positive_color = NULL,
  positive_color = NULL,
  edge_negative_color = NULL,
  negative_color = NULL,
  edge_duplicates = NULL,
  edge_labels = NULL,
  edge_label_size = NULL,
  edge_label_color = NULL,
  edge_label_position = NULL,
  edge_label_offset = NULL,
  edge_label_bg = NULL,
  edge_label_fontface = NULL,
  edge_label_border = NULL,
  edge_label_border_color = NULL,
  edge_label_underline = NULL,
  bidirectional = NULL,
  loop_rotation = NULL,
  curve_shape = NULL,
  curve_pivot = NULL,
  curves = NULL,
  node_names = NULL,
  legend = FALSE,
  legend_position = "topright",
  scaling = "default",
  weight_digits = 2
)
```

Arguments

network	A cograph_network object, matrix, data.frame, igraph or tna object. Other inputs are converted with as_cograph(); tna objects are converted with from_tna().
title	Optional plot title.
title_size	Title font size.
margins	Plot margins as c(bottom, left, top, right).
layout_margin	Margin around the network layout (proportion of viewport). Default 0.15.
newpage	Logical. Start a new graphics page? Default TRUE.
background	Background color for the plot. Default "white".
layout	Layout algorithm. Built-in: "circle", "spring", "groups", "grid", "random", "star", "bipartite". igraph (2-letter): "kk" (Kamada-Kawai), "fr" (Fruchterman-Reingold), "drl", "mds", "ni" (nicely), "tr" (tree), etc. Can also pass a coordinate matrix or igraph layout function directly. NULL (default) keeps the layout stored in a cograph_network and uses "oval" for inputs without stored coordinates.

theme	Theme name: "classic", "dark", "minimal", etc.
seed	Random seed for deterministic layouts. Default 42. Set NULL for random.
labels	Node labels. Can be a character vector to set custom labels.
threshold	Minimum absolute edge weight to display. Edges with abs(weight) < threshold are hidden. Similar to qgraph's threshold.
maximum	Maximum edge weight for width scaling. Weights above this are capped. Similar to qgraph's maximum parameter.
node_size	Node size.
node_shape	Node shape: "circle", "square", "triangle", "diamond", "ellipse", "heart", "star", "pie", "donut", "cross".
node_fill	Node fill color.
node_border_color	Node border color.
node_border_width	Node border width.
node_alpha	Node transparency (0-1).
label_size	Node label text size.
label_color	Node label text color.
label_position	Label position: "center", "above", "below", "left", "right".
show_labels	Logical. Show node labels?
pie_values	For pie/donut/donut_pie nodes: list or matrix of values for segments. For donut with single value (0-1), shows that proportion filled.
pie_colors	For pie/donut/donut_pie nodes: colors for pie segments.
pie_border_width	Border width for pie chart segments.
donut_values	For donut_pie nodes: vector of values (0-1) for outer ring proportion.
donut_border_width	Border width for donut ring.
donut_inner_ratio	For donut nodes: inner radius ratio (0-1). Default 0.5.
donut_bg_color	For donut nodes: background color for unfilled portion.
donut_show_value	For donut nodes: show value in center? Default FALSE.
donut_value_size	For donut nodes: font size for center value.
donut_value_color	For donut nodes: color for center value text.
donut_fill	Numeric value (0-1) for donut fill proportion. This is the simplified API for creating donut charts. Can be a single value or vector per node.
donut_color	Fill color(s) for the donut ring. Simplified API: single color for fill, or c(fill, background) for both.
donut_colors	Deprecated. Use donut_color instead.

donut_shape Base shape for donut: "circle", "square", "hexagon", "triangle", "diamond", "pentagon". The default "circle" takes the base shape from node_shape when that is one of these shapes.

donut_value_fontface
Font face for donut center value: "plain", "bold", "italic", "bold.italic". Default "bold".

donut_value_fontfamily
Font family for donut center value. Default "sans".

donut_value_digits
Decimal places for donut center value. Default 2.

donut_value_prefix
Text before donut center value (e.g., "$"). Default "".

donut_value_suffix
Text after donut center value (e.g., "%"). Default "".

donut2_values List of values for inner donut ring (for double donut).

donut2_colors List of color vectors for inner donut ring segments.

donut2_inner_ratio
Inner radius ratio for inner donut ring. Default 0.4.

edge_width Edge width. If NULL, scales by weight using edge_size and edge_width_range.

edge_size Maximum edge width for weight scaling. It replaces the upper bound of edge_width_range. NULL (default) uses edge_width_range unchanged.

esize Deprecated. Use edge_size instead.

edge_width_range
Output width range as c(min, max) for weight-based scaling. Default c(0.5, 4). Edges are scaled to fit within this range.

edge_scale_mode
Scaling mode for edge weights: "linear" (default), "log" (for wide weight ranges), "sqrt" (moderate compression), or "rank" (equal visual spacing).

edge_cutoff Accepted for compatibility with splot(). The grid renderer keeps width scaling continuous and does not use the value.

cut Deprecated. Use edge_cutoff instead.

edge_width_scale
Scale factor for edge widths. Values > 1 make edges thicker.

edge_color Edge color.

edge_alpha Edge transparency (0-1).

edge_style Line style: "solid", "dashed", "dotted", "longdash", "twodash".

curvature Edge curvature amount.

arrow_size Size of arrow heads.

show_arrows Logical. Show arrows?

edge_positive_color
Color for positive edge weights.

positive_color Deprecated. Use edge_positive_color instead.

`edge_negative_color`
: Color for negative edge weights.

`negative_color`
: Deprecated. Use `edge_negative_color` instead.

`edge_duplicates`
: How to handle duplicate edges in undirected networks. NULL (default) = stop with error listing duplicates. Options: "sum", "mean", "first", "max", "min", or a custom aggregation function.

`edge_labels`
: Edge labels. Can be TRUE to show weights, or a vector.

`edge_label_size`
: Edge label text size.

`edge_label_color`
: Edge label text color.

`edge_label_position`
: Position along edge (0 = source, 0.5 = middle, 1 = target).

`edge_label_offset`
: Perpendicular offset from edge line.

`edge_label_bg`
: Background color for edge labels (default "white"). Set to NA for transparent.

`edge_label_fontface`
: Font face: "plain", "bold", "italic", "bold.italic".

`edge_label_border`
: Border style: NULL, "rect", "rounded", "circle".

`edge_label_border_color`
: Border color for label border.

`edge_label_underline`
: Logical. Underline the label text?

`bidirectional`
: Logical. Show arrows at both ends of edges?

`loop_rotation`
: Angle in radians for self-loop direction (default: pi/2 = top).

`curve_shape`
: Spline tension for curved edges (-1 to 1, default: 0).

`curve_pivot`
: Pivot position along edge for curve control point (0-1, default: 0.5).

`curves`
: Curve mode. NULL (default) or "mutual" keeps single edges straight and curves reciprocal edges as two opposing arcs; FALSE plots all edges straight; "force" curves all edges. `TRUE` is rejected with an error.

`node_names`
: Alternative names for legend (separate from display labels).

`legend`
: Logical. Show legend?

`legend_position`
: Legend position: "topright", "topleft", "bottomright", "bottomleft".

`scaling`
: Scaling mode: "default" for qgraph-matched scaling where node_size=6 looks similar to qgraph vsize=6, or "legacy" for the earlier cograph scaling constants.

`weight_digits`
: Number of decimal places to which a matrix input is rounded before conversion, so that entries rounding to zero are not plotted as edges. Other inputs are not rounded. Default 2. Set NULL to disable rounding.

Details

soplot and splot:

`soplot()` uses grid graphics and `splot()` uses base R graphics. The two functions share argument names for the common aesthetics, and `splot()` has a larger set of arguments.

Edge Curve Behavior:

With the default `curves`, reciprocal edge pairs (A->B and B->A) curve in opposite directions and single edges remain straight. `curves = FALSE` plots all edges as straight lines, and `curves = "force"` curves every edge.

Weight Scaling Modes:

`edge_scale_mode` sets how edge weights map to widths. `"linear"` makes width proportional to weight, `"log"` compresses weights that span orders of magnitude, `"sqrt"` gives moderate compression, and `"rank"` spaces widths evenly by weight rank.

Donut Visualization:

Donuts show proportions (0-1) as filled rings around nodes. `donut_fill` sets the filled proportion per node, `donut_color` sets the fill color (or fill and background), `donut_shape` sets the base shape and `donut_show_value` prints the value in the center.

Value

The updated `cograph_network` object, invisibly. The function is called for the plot it produces.

The updated `cograph_network` object, invisibly. The function is called for the plot it produces.

See Also

`splot` for base R graphics rendering (alternative engine), `cograph` for creating network objects, `sn_nodes` for node customization, `sn_edges` for edge customization, `sn_layout` for layout algorithms, `sn_theme` for visual themes, `from_qgraph` and `from_tna` for converting external objects

Examples

```
soplot(regulation_net, layout = "circle")
```

---

spanning_tree *Minimum or Maximum Spanning Tree*

Description

Computes a spanning tree with Prim's algorithm on each connected component, so a disconnected network yields a spanning forest. A directed network is symmetrized first, each pair taking the larger of its two arc weights. Self-loops are ignored. Missing or infinite weights raise a `cograph_bad_selection` error.

Usage

```
spanning_tree(
  x,
  weights = c("weight", "none"),
  maximum = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
weights	"weight" (default) uses the edge weights as costs; "none" treats every edge as cost 1.
maximum	Logical. Find the maximum spanning tree instead of the minimum. Default FALSE. Set TRUE when the weights are similarities.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input. The tree itself is undirected.

Value

An undirected cograph_network holding the spanning tree (or forest), or the input format when keep_format = TRUE. Every node is kept.

References

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6), 1389–1401.

See Also

disparity_filter, threshold_edges

Examples

```
spanning_tree(regulation_net, maximum = TRUE)
```

`split_components` *Split a Network into Its Connected Components*

Description

Split a Network into Its Connected Components

Usage

```
split_components(x, min_size = 1L, keep_format = FALSE, directed = NULL)
```

Arguments

`x`	Network input.
`min_size`	Integer. Components with fewer nodes are dropped. Default 1 keeps every component, including isolated nodes.
`keep_format`	Logical. If TRUE, each component of a matrix, igraph, statnet network or tna input is returned in that format. An edge-list data frame or a qgraph object gives `cograph_network` components with a `cograph_no_format_roundtrip` warning. Default FALSE.
`directed`	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A list of `cograph_network` objects, one per component, ordered from largest to smallest and named `"component_1"`, `"component_2"`, and so on. Components are weakly connected, matching `igraph::components(mode = "weak")`. A network with no nodes gives an empty list and a warning.

See Also

`select_component`, `remove_isolates`

Examples

```
split_components(threshold_edges(regulation_net, minimum = 0.3))
```

splot.net_bootstrap *Base R Graphics Network Plotting*

Description

Plots a network with base R graphics (`polygon()`, `lines()`, `xspline()`). `splot()` uses the same snake_case argument names as the grid-based `soplot()`. Result objects of tna, Nestimate and cograph analyses are plotted by the methods described in Details.

Usage

```
splot.net_bootstrap(
  x,
  display = c("styled", "significant", "full"),
  show_ci = FALSE,
  show_stars = TRUE,
  inherit_style = TRUE,
  ...
)

splot.tna_communities(x, ...)

splot.cograph_communities(x, ...)

splot.net_mlvar(x, type = "temporal", combined = TRUE, ...)

splot.netobject(x, ...)

splot.boot_glasso(
  x,
  use_thresholded = TRUE,
  show_inclusion = TRUE,
  inclusion_threshold = NULL,
  edge_positive_color = "#2E7D32",
  edge_negative_color = "#C62828",
  ...
)

splot.wtna_mixed(x, type = c("overlay", "group"), ...)

splot.net_permutation(
  x,
  show_nonsig = FALSE,
  show_effect = FALSE,
  edge_positive_color = "#009900",
  edge_negative_color = "#C62828",
  edge_nonsig_color = "#888888",
```

```
  edge_nonsig_style = 2L,
  show_stars = TRUE,
  ...
)

splot(
  x,
  layout = "oval",
  directed = NULL,
  seed = 42,
  theme = NULL,
  node_size = NULL,
  node_size2 = NULL,
  scale_nodes_by = NULL,
  node_size_range = c(2, 8),
  scale_nodes_scale = 1,
  node_shape = "circle",
  node_svg = NULL,
  svg_preserve_aspect = TRUE,
  node_fill = NULL,
  node_border_color = NULL,
  node_border_width = 1,
  node_alpha = 1,
  labels = TRUE,
  label_abbrev = NULL,
  label_size = NULL,
  label_color = "black",
  label_position = "center",
  label_fontface = "plain",
  label_fontfamily = "sans",
  label_hjust = 0.5,
  label_vjust = 0.5,
  label_angle = 0,
  pie_values = NULL,
  pie_colors = NULL,
  pie_border_width = NULL,
  donut_fill = NULL,
  donut_values = NULL,
  donut_color = NULL,
  donut_colors = NULL,
  donut_border_color = NULL,
  donut_border_width = NULL,
  donut_inner_border_color = NULL,
  donut_inner_border_width = NULL,
  donut_outer_border_color = NULL,
  donut_line_type = "solid",
  donut_border_lty = NULL,
  donut_inner_ratio = 0.8,
```

```
  donut_bg_color = "gray90",
  donut_shape = "circle",
  donut_show_value = FALSE,
  donut_value_size = 0.8,
  donut_value_color = "black",
  donut_value_fontface = "bold",
  donut_value_fontfamily = "sans",
  donut_value_digits = 2,
  donut_value_prefix = "",
  donut_value_suffix = "",
  donut_empty = TRUE,
  donut2_values = NULL,
  donut2_colors = NULL,
  donut2_inner_ratio = 0.4,
  edge_color = NULL,
  edge_width = NULL,
  edge_size = NULL,
  esize = NULL,
  edge_width_range = c(0.1, 4),
  edge_scale_mode = "linear",
  edge_cutoff = NULL,
  cut = NULL,
  edge_alpha = 0.8,
  edge_labels = FALSE,
  edge_label_size = 0.8,
  edge_label_color = "gray30",
  edge_label_bg = NA,
  edge_label_position = 0.5,
  edge_label_offset = 0,
  edge_label_fontface = "plain",
  edge_label_shadow = FALSE,
  edge_label_shadow_color = "gray40",
  edge_label_shadow_offset = 0.5,
  edge_label_shadow_alpha = 0.5,
  edge_label_halo = TRUE,
  edge_style = 1,
  curvature = 0,
  curve_scale = TRUE,
  curve_shape = 0,
  curve_pivot = 0.5,
  curves = TRUE,
  arrow_size = 1,
  arrow_angle = pi/6,
  show_arrows = TRUE,
  bidirectional = FALSE,
  loop_rotation = NULL,
  show = NULL,
  edge_start_style = "solid",
```

```
edge_start_length = 0.15,
edge_start_dot_density = "12",
edge_ci = NULL,
edge_ci_scale = 2,
edge_ci_alpha = 0.15,
edge_ci_color = NA,
edge_ci_style = 2,
edge_ci_arrows = FALSE,
edge_priority = NULL,
edge_label_style = "none",
edge_label_template = NULL,
edge_label_digits = 2,
edge_label_oneline = TRUE,
edge_label_ci_format = "bracket",
edge_label_leading_zero = TRUE,
edge_ci_lower = NULL,
edge_ci_upper = NULL,
edge_label_p = NULL,
edge_label_p_diff = NULL,
edge_label_p_digits = 3,
edge_label_p_prefix = "p=",
edge_label_stars = NULL,
weight_digits = 2,
threshold = 0,
minimum = 0,
maximum = NULL,
edge_positive_color = "#2E7D32",
positive_color = NULL,
edge_negative_color = "#C62828",
negative_color = NULL,
edge_duplicates = NULL,
title = NULL,
title_size = 1.2,
margins = c(0.1, 0.1, 0.1, 0.1),
background = "white",
rescale = TRUE,
layout_scale = 1,
layout_margin = 0.15,
aspect = TRUE,
use_pch = FALSE,
usePCH = NULL,
scaling = "default",
align_panels = FALSE,
legend = FALSE,
legend_position = "topright",
legend_size = 0.8,
legend_edge_colors = TRUE,
legend_node_sizes = FALSE,
```

```
  groups = NULL,
  node_names = NULL,
  tna_styling = NULL,
  psych_styling = NULL,
  predictability = NULL,
  i = NULL,
  filetype = "default",
  filename = file.path(tempdir(), "splot"),
  width = 7,
  height = 7,
  res = 600,
  ...
)
```

Arguments

x	Network input. Can be: • A square numeric matrix (adjacency/weight matrix) • A data frame with edge list (from, to, optional weight columns) • An igraph object • A CographNetwork or cograph_network object • A tna object (from tna package) • A group_tna object (list of tna objects from tna package). Use parameter i to select a specific group, or omit to plot all groups. • A result object handled by one of the methods: net_bootstrap, net_permutation, boot_glasso, netobject, net_mlvar and wtna_mixed from Nestimate, tna_communities from tna, and cograph_communities.
display	Display mode of splot.net_bootstrap(): "styled" (default), "significant" or "full".
show_ci	Logical. Add the confidence bounds to the edge labels of a net_bootstrap plot. Default FALSE.
show_stars	Logical. Show significance stars in the edge labels of a net_bootstrap or net_permutation plot. Default TRUE.
inherit_style	Logical. Take labels, layout and colors from the original network of a net_bootstrap object. Default TRUE.
...	For splot(), additional arguments passed to the layout function. For the methods, additional arguments passed to splot(), or to plot_mixed_network() for splot.wtna_mixed() with type = "overlay". Arguments supplied here override the styling defaults of a method. For a multi-panel input, such as a group_tna, group_tna_bootstrap, group_tna_permutation or net_permutation_group object, combined = TRUE (default) arranges the panels with graphics::par(mfrow = ...), and combined = FALSE plots them into a layout set up beforehand, for example with panel_layout(). It has no effect for a single network.
type	For a net_mlvar object, the network to plot: "temporal" or "t" (default), "contemporaneous" or "c", "between" or "b", or "all" or "a" for a 1 x 3 panel, matched without regard to case. For a wtna_mixed object, "overlay"

(default) plots both networks on one canvas with plot_mixed_network, the co-occurrences as straight undirected edges and the transitions as curved directed edges, and "group" plots each network in its own panel.

combined
: Logical. With type = "all" for a net_mlvar object, TRUE (default) arranges the three panels in a 1 x 3 grid, and FALSE plots them into a layout set up beforehand, for example with panel_layout(). Ignored for a single network.

use_thresholded
: Logical. Plot $thresholded_pcor of a boot_glasso object, or $original_pcor when FALSE. Default TRUE.

show_inclusion
: Logical. Map the inclusion probability of each edge of a boot_glasso object to its opacity, from 0.2 to 1. Default TRUE.

inclusion_threshold
: Numeric. Minimum inclusion probability of a plotted boot_glasso edge. NULL (default) uses 1 - x$alpha, or 0.95 when x$alpha is absent.

edge_positive_color
: Color for positive weights. The default is "#2E7D32", and "#009900" for splot.net_permutation().

edge_negative_color
: Color for negative weights. Default "#C62828".

show_nonsig
: Logical. Show non-significant edges of a net_permutation plot. Default FALSE.

show_effect
: Logical. Show the effect size in parentheses in the edge labels of a net_permutation plot. Default FALSE.

edge_nonsig_color
: Color of non-significant edges. Default "#888888".

edge_nonsig_style
: Line type of non-significant edges. Default 2.

layout
: Layout algorithm: "oval" (default), "circle", "spring", "groups", "target" (qgraph-style focal-node BFS levels; node of interest via target), "saqr" (Start/End transition flow; start/ end/jitter), or a matrix of x,y coordinates, or an igraph layout function. Also supports igraph two-letter codes: "kk", "fr", "drl", "mds", "ni", etc.

directed
: Logical. Force directed interpretation. NULL for auto-detect.

seed
: Random seed for deterministic layouts. Default 42.

theme
: Theme name: "classic", "dark", "minimal", "colorblind", etc.

node_size
: Node size(s). Single value or vector. Default NULL, which resolves to 7 with default scaling.

node_size2
: Secondary node size for ellipse/rectangle height.

scale_nodes_by
: Scale node sizes by a centrality measure. Can be:

- A measure name: "degree", "strength", "betweenness", "closeness", "eigenvector", "pagerank", "authority", "hub", "harmonic", etc.
- A directional shorthand: "indegree", "outdegree", "instrength", "outstrength", "incloseness", "outcloseness", "inharmonic", "outharmonic", "ineccentricity", "outeccentricity".

- A list with measure and parameters: list("pagerank", damping = 0.9)

When used, node_size is ignored. Use node_size_range to control the min/max size. Default NULL (no centrality scaling).

`node_size_range`
: Size range for centrality-based scaling. Numeric vector c(min_size, max_size). Default c(2, 8).

`scale_nodes_scale`
: Dampening exponent for centrality-based sizing. Values < 1 compress differences (e.g., 0.5 applies square root), values > 1 exaggerate differences. Default 1 (linear).

`node_shape`
: Node shape(s): "circle", "square", "triangle", "diamond", "pentagon", "hexagon", "star", "heart", "ellipse", "cross", or any custom SVG shape registered with register_svg_shape().

`node_svg`
: Custom SVG for nodes: path to SVG file OR inline SVG string.

`svg_preserve_aspect`
: Logical: maintain SVG aspect ratio? Default TRUE.

`node_fill`
: Node fill color(s).

`node_border_color`
: Node border color(s).

`node_border_width`
: Node border width(s).

`node_alpha`
: Node transparency (0-1). Default 1.

`labels`
: Node labels: TRUE (use node names/indices), FALSE (none), or character vector.

`label_abbrev`
: Controls label abbreviation in the same way as `plot_mcml()`: `NULL` keeps full labels, an integer truncates labels to that maximum number of characters, and `"auto"` adapts the maximum length to the number of nodes.

`label_size`
: Label character expansion factor.

`label_color`
: Label text color.

`label_position`
: Label position: "center", "above", "below", "left", "right".

`label_fontface`
: Font face for labels: "plain", "bold", "italic", "bold.italic". Default "plain".

`label_fontfamily`
: Font family for labels: "sans", "serif", "mono". Default "sans".

`label_hjust`
: Horizontal justification (0=left, 0.5=center, 1=right). Default 0.5.

`label_vjust`
: Vertical justification (0=bottom, 0.5=center, 1=top). Default 0.5.

`label_angle`
: Text rotation angle in degrees. Default 0.

`pie_values`
: List of numeric vectors for pie chart nodes. Each element corresponds to a node and contains values for pie segments. If a simple numeric vector with values between 0 and 1 is provided (e.g., centrality scores), it is automatically converted to donut_fill for convenience.

`pie_colors`
: List of color vectors for pie segments.

`pie_border_width`
: Border width for pie slice dividers. NULL uses node_border_width.

`donut_fill` Numeric value (0-1) for donut fill proportion. This is the qgraph-style API: 0.1 = 10% filled, 0.5 = 50% filled, 1.0 = fully filled. Can be a single value (all nodes) or vector (per-node values).

`donut_values` Deprecated. Use donut_fill for simple fill proportion.

`donut_color` Fill color(s) for the donut ring. Single color sets fill for all nodes. Two colors set fill and background for all nodes. More than 2 colors set per-node fill colors (recycled to n_nodes). Default: "maroon" fill, "gray90" background when node_shape="donut".

`donut_colors` Deprecated. Use donut_color instead.

`donut_border_color`
Border color for donut rings. NULL uses node_border_color.

`donut_border_width`
Border width for donut rings. NULL uses node_border_width.

`donut_inner_border_color`
Color for the inner boundary (where the donut meets its hole). NULL (default) uses `donut_border_color`. Can be scalar or per-node vector.

`donut_inner_border_width`
Width for the inner boundary border. NULL (default) uses `donut_border_width`. Can be scalar or per-node vector.

`donut_outer_border_color`
Color for outer boundary border (enables double border). NULL (default) shows single border. Set to a color for double border effect. Can be scalar or per-node vector.

`donut_line_type`
Line type for donut borders: "solid", "dashed", "dotted", or numeric (1=solid, 2=dashed, 3=dotted). Can be scalar or per-node vector.

`donut_border_lty`
Deprecated. Use `donut_line_type` instead.

`donut_inner_ratio`
Inner radius ratio for donut (0-1). Default 0.8.

`donut_bg_color` Background color for unfilled donut portion.

`donut_shape` Base shape for donut: "circle", "square", "hexagon", "triangle", "diamond", "pentagon". Can be a single value or per-node vector. Default inherits from node_shape (e.g., hexagon nodes get hexagon donuts). Set explicitly to override (e.g., donut_shape = "hexagon" for hexagon donuts on all nodes regardless of node_shape).

`donut_show_value`
Logical: show value in donut center? Default FALSE.

`donut_value_size`
Font size for donut center value.

`donut_value_color`
Color for donut center value.

`donut_value_fontface`
Font face for donut center value: "plain", "bold", "italic", "bold.italic". Default "bold".

donut_value_fontfamily
: Font family for donut center value: "sans", "serif", "mono". Default "sans".

donut_value_digits
: Decimal places for donut center value. Default 2.

donut_value_prefix
: Text before donut center value (e.g., "$"). Default "".

donut_value_suffix
: Text after donut center value (e.g., "%"). Default "".

donut_empty
: Logical: render empty donut rings for NA values? Default TRUE.

donut2_values
: List of values for inner donut ring (for double donut).

donut2_colors
: List of color vectors for inner donut ring segments.

donut2_inner_ratio
: Inner radius ratio for inner donut ring. Default 0.4.

edge_color
: Edge color(s). If NULL, uses edge_positive_color/edge_negative_color based on weight.

edge_width
: Edge width(s). If NULL, scales by weight using edge_size and edge_width_range.

edge_size
: Maximum edge size for weight scaling. NULL (default) uses the upper bound of `edge_width_range`. Larger values = thicker edges overall.

esize
: Deprecated. Use `edge_size` instead.

edge_width_range
: Output width range as c(min, max) for weight-based scaling. Default c(0.1, 4). Edges are scaled to fit within this range unless `edge_size` supplies the maximum.

edge_scale_mode
: Scaling mode for edge weights: "linear" (default, qgraph-style), "log" (logarithmic for wide weight ranges), "sqrt" (moderate compression), or "rank" (equal visual spacing regardless of weight distribution).

edge_cutoff
: Optional cutoff for edge emphasis. NULL (default) or 0 disables cutoff fading. Positive values fade edges whose absolute weights are below the cutoff; width scaling remains continuous.

cut
: Deprecated. Use `edge_cutoff` instead.

edge_alpha
: Edge transparency (0-1). Default 0.8.

edge_labels
: Edge labels: TRUE (show weights), FALSE (none), or character vector.

edge_label_size
: Edge label size.

edge_label_color
: Edge label text color.

edge_label_bg
: Edge label background color.

edge_label_position
: Position along edge (0-1).

edge_label_offset
: Perpendicular offset for edge labels (0 = on line, positive = above).

`edge_label_fontface`
: Font face: "plain", "bold", "italic", "bold.italic".

`edge_label_shadow`
: Logical: enable drop shadow for edge labels? Default FALSE.

`edge_label_shadow_color`
: Color for edge label shadow. Default "gray40".

`edge_label_shadow_offset`
: Offset distance for shadow in points. Default 0.5.

`edge_label_shadow_alpha`
: Transparency for shadow (0-1). Default 0.5.

`edge_label_halo`
: Logical: enable white halo/outline around edge labels for readability over dark edges? Default TRUE. When TRUE, overrides shadow settings.

`edge_style`
: Line type(s): 1=solid, 2=dashed, 3=dotted, etc.

`curvature`
: Edge curvature. 0 for straight, positive/negative for curves.

`curve_scale`
: Not used.

`curve_shape`
: Spline tension (-1 to 1). Default 0.

`curve_pivot`
: Position along edge for curve control point (0-1).

`curves`
: Curve mode: TRUE (default) = single edges straight, reciprocal edges curve as ellipse (two opposing curves); FALSE = all straight; "force" = all curved.

`arrow_size`
: Arrow head size.

`arrow_angle`
: Arrow head angle in radians. Default pi/6 (30 degrees).

`show_arrows`
: Logical or vector: show arrows on directed edges?

`bidirectional`
: Logical or vector: show arrows at both ends?

`loop_rotation`
: Angle(s) in radians for self-loop direction.

`show`
: Dispatch-only placeholder used by method dispatch (e.g., `splot.tna_disparity`). Not intended for direct use.

`edge_start_style`
: Style for the start segment of edges: "solid" (default), "dashed", or "dotted". Use dashed/dotted to indicate edge direction (source node).

`edge_start_length`
: Fraction of edge length for the styled start segment (0-0.5). Default 0.15 (15% of edge). Only applies when edge_start_style is not "solid".

`edge_start_dot_density`
: Pattern for dotted start segments. A two-character string where the first digit is dot length and second is gap length (in line width units). Default "12" (1 unit dot, 2 units gap). Use "11" for tighter dots, "13" for more spacing. Only applies when edge_start_style = "dotted".

`edge_ci`
: Numeric vector of CI widths (0-1 scale). Larger values = more uncertainty.

`edge_ci_scale`
: Width multiplier for underlay thickness. Default 2.

`edge_ci_alpha`
: Transparency for underlay (0-1). Default 0.15.

`edge_ci_color`
: Underlay color. NA (default) uses main edge color.

edge_ci_style Line type for underlay: 1=solid, 2=dashed, 3=dotted. Default 2.

edge_ci_arrows Logical: show arrows on underlay? Default FALSE.

edge_priority Numeric vector of edge priorities. Higher values render on top. Useful for ensuring significant edges appear above non-significant ones.

edge_label_style
Preset style: "none", "estimate", "full", "range", "stars".

edge_label_template
Template with placeholders: {est}, {range}, {low}, {up}, {p}, {p_diff}, {stars}. Overrides edge_label_style if provided.

edge_label_digits
Decimal places for estimates. Default 2.

edge_label_oneline
Logical: single line format? Default TRUE.

edge_label_ci_format
CI format: "bracket" for [low, up] or "dash" for low-up.

edge_label_leading_zero
Logical: show leading zero for values < 1? Default TRUE. Set to FALSE to display ".5" instead of "0.5".

edge_ci_lower Numeric vector of lower CI bounds for labels.

edge_ci_upper Numeric vector of upper CI bounds for labels.

edge_label_p Numeric vector of p-values for edges.

edge_label_p_diff
Probability-of-difference values for the {p_diff} template placeholder: a per-edge numeric vector, or a full node-by-node matrix, which is indexed at each plotted edge and so stays aligned when minimum or threshold remove edges. A matrix with dimnames is aligned to the plot's node names, so it may be supplied in any node order.

edge_label_p_digits
Decimal places for p-values. Default 3.

edge_label_p_prefix
Prefix for p-values. Default "p=".

edge_label_stars
Stars for labels: character vector, TRUE (compute from p), or numeric (treated as p-values).

weight_digits Number of decimal places to which the weights of a matrix input are rounded before plotting, so edges that round to zero are removed. Default 2. NULL disables rounding.

threshold Minimum absolute weight to display.

minimum Alias for threshold (qgraph compatibility). Uses max of threshold and minimum.

maximum Maximum weight for scaling. NULL for auto.

positive_color Deprecated. Use edge_positive_color instead.

negative_color Deprecated. Use edge_negative_color instead.

edge_duplicates
: How to handle duplicate edges in undirected networks. NULL (default) = stop with error listing duplicates. Options: "sum", "mean", "first", "max", "min", or a custom aggregation function.

title
: Plot title.

title_size
: Title font size.

margins
: Margins as c(bottom, left, top, right).

background
: Background color.

rescale
: Logical: rescale layout to -1 to 1 range?

layout_scale
: Scale factor for layout. >1 expands (spreads nodes apart), <1 contracts (brings nodes closer). Use "auto" to automatically scale based on node count (compact for small networks, expanded for large). Default 1.

layout_margin
: Margin around the layout as fraction of range. Default 0.15. Set to 0 for no extra margin (tighter fit). Affects white space around nodes.

aspect
: Logical: maintain aspect ratio?

use_pch
: Logical: use points() for simple circles (faster). Default FALSE.

usePCH
: Deprecated. Use use_pch instead.

scaling
: Scaling mode: "default" for qgraph-matched scaling where node_size=6 looks similar to qgraph vsize=6, or "legacy" to preserve pre-v2.0 behavior.

align_panels
: Logical. If TRUE, forces a uniform symmetric plot box (c(-layout_scale, layout_scale) on each axis) so two networks plotted side-by-side in a par(mfrow) grid render at identical absolute scales, for example bootstrap panels or networks with different numbers of nodes. Default FALSE uses bounds computed from the layout, which fill the canvas more tightly. The fixed box is applied only when the layout is rescaled, so align_panels = TRUE has no effect with rescale = FALSE.

legend
: Logical: show legend?

legend_position
: Position: "topright", "topleft", "bottomright", "bottomleft".

legend_size
: Legend text size.

legend_edge_colors
: Logical: show positive/negative edge colors in legend?

legend_node_sizes
: Logical: show node size scale in legend?

groups
: Group assignments for node coloring/legend.

node_names
: Alternative names for legend (separate from labels).

tna_styling
: Logical or NULL. If TRUE, applies TNA visual defaults (oval layout, TNA color palette, edge labels as estimates, dotted edge starts, etc.) as a base layer. Any explicitly provided argument overrides the TNA default. If FALSE, no TNA styling is applied. If NULL (default), automatically set to TRUE when x is a tna object, FALSE otherwise. Can be used with any input type (matrix, igraph, cograph_network).

psych_styling	Logical or NULL. Undirected counterpart of tna_styling. If TRUE, applies psychometric-network defaults (spring layout, Okabe-Ito palette, no arrows, solid edge lines, and minimum = 0.01) as a base layer. If NULL (default), splot.netobject auto-enables it on correlation-family input (glasso, cor, pcor, ising) and on the undirected constituents of net_mlvar. Explicit user args always win.
predictability	Logical or NULL. Draws a per-node predictability ring (a donut fill) from a predictability column on the network's node table, the way qgraph/bootnet show node predictability. If TRUE, draws it when the column is present; if FALSE, never; if NULL (default), draws it when the object marks it as its default (network$meta$predictability_default, set e.g. by a psychnet glasso network). A caller's own pie_values / donut_fill takes precedence.
i	Group index or name when x is a group_tna object. If NULL (default), plots all groups in a grid. If specified (e.g., i = 1 or i = "Treatment"), plots only that group.
filetype	Output format: "default" (screen), "png", "pdf", "svg", "jpeg", "tiff".
filename	Output filename (without extension).
width	Output width in inches.
height	Output height in inches.
res	Resolution in DPI for raster outputs (PNG, JPEG, TIFF). Default 600.

Details

splot.net_bootstrap() plots a net_bootstrap object from the Nestimate package. It shows the original network, whose weights are taken from $original$weights, with the edges styled by bootstrap significance.

splot.tna_communities() plots the tna model stored in attr(x, "tna") by tna::communities(), with nodes colored by community. The walktrap assignment is used when present, and the first algorithm column of x$assignments otherwise.

splot.cograph_communities() plots the network stored in attr(x, "network") by detect_communities(), with nodes colored by community.

splot.net_mlvar() plots one or all of the temporal, contemporaneous and between-subjects networks of a net_mlvar object from Nestimate::build_mlvar().

splot.netobject() plots a netobject from Nestimate. Networks estimated by a transition-type method ("relative", "frequency", "attention", "co_occurrence", "wtna", "wtna_cooccurrence", "entropy") get tna_styling = TRUE, and other methods, such as correlation and partial correlation networks, get psych_styling = TRUE. When the method is not recorded, directed networks get the TNA styling. Arguments supplied by the caller override these defaults.

splot.boot_glasso() plots the partial-correlation network of a boot_glasso object from Nestimate, with the bootstrap inclusion probability of each edge mapped to its opacity.

splot.wtna_mixed() plots a wtna_mixed object from Nestimate::wtna(..., method = "both"), either as one overlaid network or as two panels.

splot.net_permutation() plots the edge differences of a net_permutation object from Nestimate, with significant differences highlighted. The direction of the network is taken from x$x$directed.

Edge Curve Behavior:

Edge curving is controlled by two interacting arguments:

`curves` Mode for automatic curving. `FALSE` = all straight, `TRUE` (default) = curve only reciprocal edge pairs as an ellipse, `"force"` = curve all edges inward toward network center.

`curvature` Manual curvature amount (0-1 typical). Sets the magnitude of curves. Default 0 uses automatic 0.175 for curved edges. Positive values curve edges; the direction is automatically determined.

For reciprocal edges (A->B and B->A both exist), the edges curve in opposite directions to form a visual ellipse, making bidirectional relationships clear.

Weight Scaling Modes (edge_scale_mode):

Controls how edge weights are mapped to visual widths:

`"linear"` **(default)** Width proportional to weight. Best when weights are similar in magnitude.

`"log"` Logarithmic scaling. Best when weights span multiple orders of magnitude (e.g., 0.01 to 100).

`"sqrt"` Square root scaling. Moderate compression, good for moderately skewed distributions.

`"rank"` Rank-based scaling. Ignores actual values; uses relative ordering. All edges get equal visual spacing regardless of weight distribution.

Donut vs Pie vs Double Donut:

Three ways to show additional data on nodes:

Donut (`donut_fill`**)** Single ring showing a proportion (0-1). Ideal for completion rates, probabilities, or any single metric per node. Use `donut_color` for fill color and `donut_bg_color` for unfilled portion.

Pie (`pie_values`**)** Multiple colored segments showing category breakdown. Ideal for composition data. Values are normalized to sum to 1. Use `pie_colors` for segment colors.

Double donut (`donut2_values`**)** Two concentric rings for comparing two metrics per node. Outer ring uses `donut_fill`/`donut_color`, inner ring uses `donut2_values`/`donut2_colors`.

CI Underlay System:

Confidence interval underlays draw a wider, semi-transparent edge behind the main edge to visualize uncertainty:

`edge_ci` Vector of CI widths (0-1 scale). Larger = more uncertainty.

`edge_ci_scale` Multiplier for underlay width relative to main edge. Default 2 means underlay is twice as wide as main edge at CI=1.

`edge_ci_alpha` Transparency of underlay (0-1). Default 0.15.

`edge_ci_style` Line type: 1=solid, 2=dashed (default), 3=dotted.

Edge Label Templates:

For statistical output, use templates to format complex labels:

`edge_label_template` Template string with placeholders: `{est}` for estimate/weight, `{low}`/`{up}` for CI bounds, `{range}` for formatted range, `{p}` for p-value, `{p_diff}` for the probability of the difference (Bayesian comparisons), `{stars}` for significance stars.

`edge_label_style` Preset styles: `"estimate"` (weight only), `"full"` (estimate + CI), `"range"` (CI only), `"stars"` (significance).

Producer-Supplied splot Metadata:

Packages that create cograph_network-compatible objects can attach a small plotting contract at x$meta$splot. This lets producer packages such as Nestimate, lagdynamics, or other modeling packages describe their preferred cograph rendering without adding a new cograph-side class branch for every object type.

The contract is optional. Objects without meta$splot follow the normal splot() path and all existing class-specific dispatch remains in place. When present, the supported fields are:

renderer Character scalar naming the cograph renderer to use. "network" (also "splot", "default", or "base") means the object follows the normal splot() path, including any class-specific method, with the metadata defaults applied. Other values are resolved through a cograph-maintained whitelist of existing renderers, for example "difference", "bootstrap", "permutation", "stability", "mlvar", "netobject", "netobject_group", "netobject_ml", "boot_glasso", and "wtna_mixed". Arbitrary function names are never evaluated.

weight Optional character scalar naming the default edge weight to render. If it names an edge column, that column is copied to edges$weight for the plot (the producer's edge set is kept, and the weights matrix is rebuilt to match). If it names a matrix stored on the object, that matrix becomes the rendered network: it is copied to weights and the drawn edge set is rebuilt from its nonzero cells (aligned to the object's node order via dimnames when present). This is useful when the analytical object stores several edge quantities (for example counts, probabilities, residuals, effects) but has one preferred plot view. When the name matches both an edge column and a stored matrix, the matrix form wins.

defaults Named list of splot() or renderer arguments. These are defaults only: any argument explicitly supplied by the user wins. Defaults can include regular splot() arguments such as layout, node_fill, edge_labels, weight_digits, or renderer-specific arguments such as display for bootstrap renderers.

The precedence rule is always:

```
user arguments > x$meta$splot$defaults > cograph defaults
```

Example producer-side metadata:

```
x$meta$splot <- list(
  renderer = "network",
  weight = "adj_res",
  defaults = list(
    node_fill = "white",
    edge_labels = TRUE,
    weight_digits = 1
  )
)
```

Value

splot() returns the plotted cograph_network object invisibly. splot.net_bootstrap(), splot.netobject(), splot.boot_glasso(), splot.tna_communities() and splot.cograph_communities() return the cograph_network built by splot(), and splot.net_permutation() returns it or NULL when there is no edge to plot. splot.net_mlvar() and splot.wtna_mixed() return x. All are called for the side effect of plotting.

See Also

soplot for grid graphics rendering (alternative engine), cograph for creating network objects, sn_nodes for node customization, sn_edges for edge customization, sn_layout for layout algorithms, sn_theme for visual themes, from_qgraph and from_tna for converting external objects

Examples

```
splot(regulation_net)
```

---

student_interactions *Student Interaction Edge List*

Description

An edge list of observed interactions between 34 students during collaborative learning sessions. Each row represents one observed interaction between two students. The same pair may appear multiple times, reflecting repeated interactions.

Usage

```
student_interactions
```

Format

A data frame with 389 rows and 2 columns:

from Character. Anonymized two-letter student code (e.g., "Ac", "Bd")

to Character. Anonymized two-letter student code (e.g., "Ce", "Df")

Details

The dataset includes self-loops (34 rows where `from == to`). These can be removed with `subset(student_interactions, from != to)`.

The 389 rows contain 226 distinct ordered pairs. Because interactions repeat, the edge list forms a multigraph when loaded into igraph with `igraph::graph_from_data_frame()`.

Value

A data frame with 389 rows and 2 columns:

from Character. Anonymized two-letter student code.

to Character. Anonymized two-letter student code.

Source

Anonymized collaborative learning interaction data.

Examples

```
as_cograph(student_interactions)
```

---

subgraphs *Extract Specific Motif Instances (Subgraphs)*

Description

Calls `motifs(x, named_nodes = TRUE, ...)`. The result has one row per node triple and MAN type. At individual level, `observed` counts the units in which the triple has that type, so one triple can occupy several rows. One MAN type can also appear in many rows, each with its own `z` and `p`. Per-triple significance is plotted by `plot(., type = "significance")` and `plot(., type = "triads")`. The per-type plots (`"types"`, `"patterns"`) omit significance for instance results.

Usage

```
subgraphs(...)
```

Arguments

`...`	Arguments forwarded to `motifs()`, which documents them (`x`, `actor`, `window`, `window_type`, `pattern`, `include`, `exclude`, `significance`, `n_perm`, `cores`, `min_count`, `edge_method`, `edge_threshold`, `min_transitions`, `top`, `seed`). `named_nodes` is fixed to `TRUE` and must not be supplied.

Details

The `"triads"` diagram shows a canonical representative of the MAN isomorphism class of each row. The labels name the participating nodes. Their positions in the diagram do not encode the observed source and sink roles of the nodes.

Value

A `cograph_motif_result` object with `named_nodes = TRUE`, described in `motifs()`, or NULL with a message when no triple passes the filters. Its results table has the columns `triad`, `node1`, `node2`, `node3`, `type` and `observed`, and with `significance = TRUE` also `expected`, `z`, `p` and `sig`. Its `type_summary` counts the node triples of each MAN type.

See Also

`motifs()`

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `motifs()`, `triad_census()`

Examples

```
subgraphs(regulation_net, significance = FALSE, min_count = 1)
```

summarize_clusters *Build MCML from Raw Transition Data*

Description

Builds a Multi-Cluster Multi-Level (MCML) model from raw transition data (edge lists or sequences) by recoding node labels to cluster labels and counting the observed transitions. The macro network is then the Markov chain over cluster states. Weight matrices are passed to csum, which aggregates them.

Usage

```
summarize_clusters(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "frequency", "cooccurrence", "semi_markov", "raw"),
  directed = TRUE,
  compute_within = TRUE
)
```

Arguments

x — Input data. Accepts multiple formats:

data.frame with from/to columns Edge list. Columns named from/source/src/v1/node1/i and to/target/tgt/v2/node2/j are auto-detected. Optional weight column (weight/w/value/strength).

data.frame without from/to columns Sequence data. Each row is a sequence, columns are time steps. Consecutive pairs (t, t+1) become transitions. A data frame with three or more columns that all have the default names V1, V2, V3, ... is read as sequence data, even though V1 and V2 are also from/to names.

tna object If x$data is non-NULL, uses sequence path on the raw data. Otherwise falls back to csum.

cograph_network If x$data is non-NULL, detects edge list vs sequence data. Otherwise falls back to csum.

group_tna Converted as by as_mcml.

mcml or cluster_summary Returned unchanged.

square numeric matrix Falls back to csum.

non-square or character matrix Treated as sequence data.

clusters — Cluster/group assignments. Accepts:

named list Direct mapping. List names = cluster names, values = character vectors of node labels. Example: list(A = c("N1","N2"), B = c("N3","N4"))

data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: data.frame(node = c("N1","N2","N3"), group = c("A","A","B"))

membership vector Character or numeric vector. Node names are extracted from the data. Example: `c("A","A","B","B")`

column name string For edge list data.frames, the name of a column containing cluster labels. The mapping is built from unique (node, group) pairs in both from and to columns.

NULL Auto-detect from `cograph_network$nodes` or `$node_groups` (same logic as `csum`).

`method` Aggregation method for combining edge weights: "sum", "mean", "median", "max", "min", "density", "geomean". Default "sum".

`type` Post-processing: "tna" (row-normalize), "frequency" or "raw" (no normalization), "cooccurrence" (symmetrize), or "semi_markov" (row-normalize, identical to "tna"). Default "tna".

`directed` Logical. Default `TRUE`. With `FALSE` the node-level weights are symmetrized as $(A + A^T)/2$ before aggregation: a matrix input is averaged with its transpose, and each observed transition counts half in each direction. The value is recorded in `meta$directed`.

`compute_within` Logical. Compute within-cluster matrices? Default TRUE.

Value

Usually an `mcml` object. Existing `mcml` or `cluster_summary` inputs are returned unchanged. Transition-data results include `meta$source = "transitions"` and are compatible with `plot_mcml`, `as_tna`, and `splot`.

See Also

`csum` for matrix-based aggregation, `as_tna` to convert to tna objects, `plot_mcml` for visualization

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
summarize_clusters(regulation_net, clusters = clusters, method = "mean")
```

---

summarize_network *Summarize Network by Clusters*

Description

Creates a summary network where each cluster becomes a single node. Edge weights are aggregated from the original network using the specified method, without normalization.

Usage

```
summarize_network(
  x,
  cluster_list = NULL,
  method = c("sum", "mean", "max", "min", "median", "density", "geomean"),
  directed = TRUE
)

cnet(
  x,
  cluster_list = NULL,
  method = c("sum", "mean", "max", "min", "median", "density", "geomean"),
  directed = TRUE
)
```

Arguments

x	A weight matrix, tna object, or cograph_network.
cluster_list	Cluster specification: • Named list of node vectors (e.g., list(A = c("n1", "n2"), B = c("n3", "n4"))) • A membership vector or a data frame, as in csum • A single string naming a column of the node table of a cograph_network (e.g., "clusters", "groups") • NULL (default) to use the first node column named "clusters", "cluster", "groups", "group", "community" or "module" of a cograph_network, with a message naming the column
method	Aggregation method for edge weights: "sum", "mean", "max", "min", "median", "density", "geomean". Default "sum".
directed	Logical. Whether the summary network is directed. Default TRUE.

Value

A cograph_network object with one node per cluster, labelled by cluster name. The edge weights are the aggregated between-cluster weights, and the diagonal holds the aggregated within-cluster weights. The node table has a size column with the number of original nodes in each cluster.

See summarize_network.

See Also

csum, plot_mcml

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
summarize_network(regulation_net, cluster_list = clusters)
```

supra_adjacency *Supra-Adjacency Matrix*

Description

Builds the supra-adjacency matrix of a multilayer network. The diagonal blocks hold the intra-layer adjacencies and the off-diagonal blocks the inter-layer coupling.

Usage

```
supra_adjacency(
  layers,
  omega = 1,
  coupling = c("diagonal", "full", "custom"),
  interlayer_matrices = NULL
)

supra(
  layers,
  omega = 1,
  coupling = c("diagonal", "full", "custom"),
  interlayer_matrices = NULL
)
```

Arguments

layers	List of adjacency matrices (same dimensions)
omega	Inter-layer coupling coefficient, a scalar or an L x L matrix. Default 1. For a matrix, entry `[a, b]` with `a < b` sets the coupling of layers a and b.
coupling	Coupling type. `"diagonal"` (default) couples each node to its own copy in the other layers with weight `omega`. `"full"` couples every node to every node of the other layers with weight `omega`. `"custom"` uses `interlayer_matrices`.
interlayer_matrices	For `coupling = "custom"`, a list of inter-layer matrices. Accepted shapes: • Named list with keys `"a_b"` (integer layer indices) or `"<layer_name_a>_<layer_name_b>"`; either order works. • Unnamed list of length `choose(L, 2)` giving every pair in upper-triangle row-major order: `(1,2), (1,3), ..., (1,L), (2,3), ..., (L-1,L)`. • Unnamed list of length `L-1` giving adjacent pairs only. Entry `i` is the coupling for `(i, i+1)`. The block of layers b and a is the transpose of the block of a and b. A pair with no matching entry receives the diagonal coupling `omega[a,b] * I` with a warning. A `NULL` value with `coupling = "custom"` is an error.

Value

A supra-adjacency matrix of dimension (N*L*) x (*N*L) with class `c("supra_adjacency", "matrix")`. Diagonal N x N blocks hold the intra-layer adjacencies and off-diagonal blocks the inter-layer coupling. The attributes `"n_nodes"`, `"n_layers"`, `"node_names"`, `"layer_names"`, `"omega"` and `"coupling"` record the construction and are read back by `supra_layer()` and `supra_interlayer()`.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
supra_adjacency(layers, omega = 0.5)
```

`supra_interlayer` *Extract Inter-Layer Block*

Description

Extract Inter-Layer Block

Usage

```
supra_interlayer(x, from, to)

extract_interlayer(x, from, to)
```

Arguments

`x`	Supra-adjacency matrix from `supra_adjacency`
`from`	Integer index of the source layer
`to`	Integer index of the target layer

Value

The N x N inter-layer block, with the supra-matrix labels (`"<layer>_<node>"`) as dimnames. An index outside `1:L` raises an error.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
supra <- supra_adjacency(layers, omega = 0.5)
supra_interlayer(supra, from = 1, to = 2)
```

supra_layer *Extract Layer from Supra-Adjacency Matrix*

Description

Extract Layer from Supra-Adjacency Matrix

Usage

```
supra_layer(x, layer)

extract_layer(x, layer)
```

Arguments

x	Supra-adjacency matrix from supra_adjacency
layer	Integer index of the layer to extract

Value

The N x N intra-layer adjacency matrix, with the node names as dimnames. An index outside 1:L raises an error.

Examples

```
layers <- list(forward = regulation_net, backward = t(regulation_net))
supra <- supra_adjacency(layers, omega = 0.5)
supra_layer(supra, layer = 2)
```

symmetrize *Symmetrize a Directed Network*

Description

Combines each pair of opposite arcs into one undirected edge. The result is an undirected network. Self-loops are kept unchanged.

Usage

```
symmetrize(
  x,
  method = c("max", "min", "mean", "sum", "mutual", "upper", "lower"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
method	How to combine w[i, j] and w[j, i]:
	"max" (default) the larger of the two; on a binary network this is sna's "weak" rule
	"min" the smaller of the two
	"mean" their average
	"sum" their total
	"mutual" keep only reciprocated pairs, taking the smaller weight; on a binary network this is sna's "strong" rule
	"upper" take the upper triangle and mirror it
	"lower" take the lower triangle and mirror it
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. Directedness to read the input with; the result is always undirected.

Details

"max", "min", "mean" and "sum" combine two values only where both arcs exist. An unreciprocated edge keeps its own weight. In a signed network an unreciprocated negative edge is therefore kept. "mutual" keeps only reciprocated edges.

Value

An undirected cograph_network, or the input format when keep_format = TRUE. The weight matrix satisfies isSymmetric(). Zero is how this representation stores "no edge", so any pair whose combined weight is exactly zero disappears: every unreciprocated arc under method = "mutual", and a cancelling pair under "sum". A cograph_edges_dropped warning says how many.

References

Butts, C. T. (2008). Social network analysis with sna. *Journal of Statistical Software*, 24(6), 1–51.

See Also

to_undirected, normalize_weights

Examples

```
symmetrize(regulation_net, method = "mean")
```

`themes` *Themes*

Description

A theme is a `CographTheme` object that sets the background, node, edge and label colors of a plot. Themes are stored in a registry and selected by name through the `theme` argument of `splot`, `soplot` and `sn_theme`. The functions `theme_cograph_*()` return the built-in themes:

`theme_cograph_classic()` Blue nodes and gray edges (`"classic"`).

`theme_cograph_colorblind()` Colors distinguishable under color vision deficiency (`"colorblind"`).

`theme_cograph_gray()` Black and white for print (`"gray"`, also `"grey"`).

`theme_cograph_dark()` Dark background for presentations (`"dark"`).

`theme_cograph_minimal()` Thin borders and few colors (`"minimal"`).

`theme_cograph_viridis()` The viridis palette (`"viridis"`).

`theme_cograph_nature()` Earth tones (`"nature"`).

`register_theme()` adds a theme under a new name, `get_theme()` returns a registered theme, and `list_themes()` returns the names of all registered themes.

Usage

```
register_theme(name, theme)

get_theme(name)

list_themes()

theme_cograph_classic()

theme_cograph_colorblind()

theme_cograph_gray()

theme_cograph_dark()

theme_cograph_minimal()

theme_cograph_viridis()

theme_cograph_nature()
```

Arguments

`name`	Character. The name of the theme.
`theme`	A `CographTheme` object, for example one created with `CographTheme$new()` or returned by a `theme_cograph_*()` function.

Value

The `theme_cograph_*()` functions return a `CographTheme` object. `register_theme()` returns `NULL` invisibly. `get_theme()` returns the theme, or `NULL` if no theme has that name. `list_themes()` returns a character vector of theme names.

Examples

```
splot(regulation_net, theme = "dark")
```

`threshold_edges` *Threshold Edges by Weight, Count, Proportion or Density*

Description

Keeps the edges that satisfy every criterion supplied. The operation corresponds to qgraph's `minimum` argument and to `tna::prune()`. The result is a network that can be analysed and plotted.

Usage

```
threshold_edges(
  x,
  minimum = NULL,
  maximum = NULL,
  proportion = NULL,
  density = NULL,
  top = NULL,
  absolute = TRUE,
  keep_isolates = TRUE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input: cograph_network, matrix, igraph, network, tna, or an edge-list data frame.
`minimum`	Numeric. Keep edges whose weight is at least this value.
`maximum`	Numeric. Keep edges whose weight is at most this value.
`proportion`	Numeric in (0, 1]. Keep this fraction of the edges, the strongest first.
`density`	Numeric in (0, 1]. Keep as many of the strongest edges as gives this density (edges as a fraction of the possible edges).
`top`	Non-negative integer. Keep this many edges, the strongest first. `top = 0` removes every edge.

absolute	Logical. Compare abs(weight) instead of the signed weight. Default TRUE, which suits correlation and partial-correlation networks. The minimum and maximum comparisons and the ranking used by proportion, density and top all follow this flag.
keep_isolates	Logical. Keep nodes that end up with no edges? Default TRUE. Set FALSE, or call remove_isolates(), to drop them.
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

Details

Several criteria are combined with AND. For example, threshold_edges(x, minimum = 0.2, top = 20) keeps the twenty strongest edges among those of weight at least 0.2. When proportion, density and top are combined, the smallest of the implied edge counts is used.

Ties at the cut point are all kept, so top = 10 can return more than ten edges when the tenth and eleventh weights are equal. The result therefore does not depend on the order in which the edges are stored.

Value

A cograph_network with the surviving edges, or the input format when keep_format = TRUE. Every node is kept unless keep_isolates = FALSE; nodes the threshold stranded are reported in a cograph_isolates_created warning. A non-finite minimum or maximum, a proportion or density outside (0, 1], and a negative or fractional top raise a cograph_bad_selection error.

References

Epskamp, S., Cramer, A. O. J., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). qgraph: Network visualizations of relationships in psychometric data. *Journal of Statistical Software*, 48(4), 1–18.

See Also

binarize, filter_edges, disparity_filter, remove_isolates

Examples

```
threshold_edges(regulation_net, minimum = 0.1)
```

---

to_data_frame *Export Network as Edge List Data Frame*

Description

Converts a network to an edge list data frame with columns for source, target, and weight.

Usage

```
to_data_frame(x, directed = NULL)

to_df(x, directed = NULL)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

Value

A base `data.frame` with one row per edge and exactly three columns:

- `from`: Source node name/label
- `to`: Target node name/label
- `weight`: Edge weight

Further edge columns, such as `session` or `time` from temporal edge lists, are dropped. `get_edges` returns the full edge table. An undirected network contributes one row per unordered pair.

See Also

`to_df`, `to_igraph`, `as_cograph`

Examples

```
to_data_frame(regulation_net)
```

---

to_directed *Convert an Undirected Network to Directed*

Description

Convert an Undirected Network to Directed

Usage

```
to_directed(
  x,
  mode = c("mutual", "arbitrary"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
mode	"mutual" (default) creates an arc in both directions for every undirected edge. "arbitrary" keeps one arc per edge, running from the lower node index to the higher. For a directed input, "mutual" gives both arcs of a pair the larger of the two weights, and "arbitrary" drops every arc from a higher to a lower index.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

A directed cograph_network, or the input format when keep_format = TRUE.

See Also

to_undirected, reverse_edges

Examples

```
to_directed(to_undirected(regulation_net))
```

---

to_igraph *Convert Network to igraph Object*

Description

Converts various network representations to an igraph object. Supports matrices, edge-list data frames, igraph objects, network objects, cograph_network, and tna objects.

Usage

```
to_igraph(x, directed = NULL)
```

Arguments

x	Network input. Can be: • A square numeric matrix (adjacency/weight matrix) • A data frame edge list with source and target columns • An igraph object (returned as-is or converted if directed differs) • A statnet network object • A cograph_network object • A tna object
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

Value

An igraph object.

See Also

to_data_frame, as_cograph

Examples

```
to_igraph(regulation_net)
```

`to_matrix` *Convert Network to Adjacency Matrix*

Description

Converts any supported network format to an adjacency matrix.

Usage

```
to_matrix(x, directed = NULL)
```

Arguments

`x`	Network input: matrix, cograph_network, igraph, network, tna, etc.
`directed`	Logical or NULL. If NULL (default), auto-detect from input.

Value

A square numeric adjacency matrix, preserving row/column names when available.

See Also

to_igraph, to_df, as_cograph, to_network

Examples

```
to_matrix(student_interactions)
```

to_network *Convert Network to statnet network Object*

Description

Converts any supported network format to a statnet network object.

Usage

```
to_network(x, directed = NULL)
```

Arguments

x	Network input: matrix, cograph_network, igraph, tna, etc.
directed	Logical or NULL. If NULL (default), auto-detect from input.

Value

A network object from the network package.

See Also

to_igraph, to_matrix, to_df, as_cograph

Examples

```
to_network(regulation_net)
```

to_undirected *Convert a Directed Network to Undirected*

Description

Collapses each pair of opposite arcs into one undirected edge, as `igraph::as_undirected()` and tidygraph's `to_undirected()` do.

Usage

```
to_undirected(
  x,
  method = c("max", "sum", "mean", "min", "mutual"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
method	How to combine w[i, j] and w[j, i]. One of "max" (default), "sum", "mean", "min" or "mutual". The first four combine the two weights when both arcs exist, and an arc without a reverse arc keeps its own weight. "mutual" keeps only reciprocated pairs, at the smaller of the two weights.
keep_format	Logical. If TRUE, a matrix, igraph, statnet network or tna input is returned in its own format. An edge-list data frame or a qgraph object is returned as a cograph_network with a cograph_no_format_roundtrip warning. Default FALSE returns a cograph_network.
directed	Logical or NULL. Directedness used to read the input. NULL (default) detects it from the input.

Value

An undirected cograph_network, or the input format when keep_format = TRUE. Self-loops keep their weight. A weight of zero means no edge, so a pair whose combined weight is exactly zero is dropped. Under method = "mutual" this applies to every unreciprocated arc, and under "sum" to a pair of opposite weights that cancel. Dropped edges raise a cograph_edges_dropped warning.

See Also

to_directed, symmetrize

Examples

```
to_undirected(regulation_net, method = "sum")
```

---

triad_census *Triad Census*

Description

Counts the 16 types of triads in a directed network using MAN notation. Edge weights are ignored.

Usage

```
triad_census(x)
```

Arguments

x	A matrix, igraph object, or cograph_network.

Details

The triad census is defined only for directed networks. Matrix input is read as directed. An undirected igraph or cograph_network input raises an error.

A MAN code gives the number of mutual (reciprocated) dyads, the number of asymmetric dyads and the number of null (absent) dyads of a triad, followed by a letter that separates types with the same counts. The 16 triad types, in the order of the result, are 003, 012, 102, 021D, 021U, 021C, 111D, 111U, 030T, 030C, 201, 120D, 120U, 120C, 210 and 300.

Value

A named numeric vector of length 16 giving the count of each MAN triad type, in the order listed under Details.

See Also

`motifs()` for the unified API, `motif_census()`

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `motifs()`, `subgraphs()`

Examples

```
cograph::triad_census(regulation_net)
```

---

trophic_incoherence *Trophic Incoherence Parameter*

Description

Computes the trophic incoherence parameter q, which measures how vertically ordered a directed network is (Johnson et al. 2014). For each edge (u, v), the trophic difference is $x_{uv} = s_v - s_u$ where s_i is the trophic level of node i. The trophic incoherence parameter is the (population) standard deviation of these differences:

$$q = \sqrt{\frac{1}{|E|} \sum_{(u,v) \in E} (x_{uv} - \bar{x})^2}$$

Usage

```
trophic_incoherence(x, cannibalism = TRUE)
```

Arguments

x	Directed network input.
cannibalism	Logical. If FALSE, self-loops are removed before computing trophic differences. Default TRUE.

Details

Values near 0 indicate a coherent network, in which every edge rises by about one level. High values indicate many level-skipping or downward edges. Johnson et al. (2014) reported that food webs with low q are more stable.

Trophic levels are computed on the binary adjacency matrix, so edge weights are ignored. The levels are defined only for a directed network in which every node can be reached from a basal node (a node with no incoming edges).

Value

Numeric scalar: the trophic incoherence parameter. `NA` when the network has no edges or when some node cannot be reached from a basal node. An undirected network returns `NA` with a warning.

References

Johnson, S., Dominguez-Garcia, V., Donetti, L., & Munoz, M. A. (2014). Trophic coherence determines food-web stability. *PNAS*, 111(50), 17923-17928.

See Also

`centrality` (the `trophic_level` measure) for the per-node levels used in the incoherence calculation.

Examples

```
strong <- filter_edges(regulation_net, weight > 0.3, keep_isolates = FALSE)
trophic_incoherence(strong)
```

`verify_with_igraph` *Verify Against igraph*

Description

Compares the macro weights of `csum` with the result of contracting the clusters in igraph (`igraph::contract()` followed by `igraph::simplify()`).

Usage

```
verify_with_igraph(x, clusters, method = "sum", type = "raw")

verify_igraph(x, clusters, method = "sum", type = "raw")
```

Arguments

`x`	Adjacency matrix
`clusters`	Cluster specification (see `csum`)
`method`	Aggregation method. Default "sum".
`type`	Normalization type passed to `csum`. Default "raw", the only type whose values igraph reproduces.

Value

A list with components `our_result` (cograph's macro weight matrix), `igraph_result` (igraph's `contract()` + `simplify()` matrix, diagonal set to zero), `matches` (logical, whether the off-diagonal cells agree to within 1e-10) and `difference` (the `all.equal()` report when they do not, otherwise NULL). Returns `NULL` with a message if igraph is not installed.

Examples

```
clusters <- list(C1 = c("Explore", "Reflect", "Discuss"),
                 C2 = c("Plan", "Create", "Share"),
                 C3 = c("Monitor", "Adapt", "Synthesize", "Evaluate"))
verify_with_igraph(regulation_net, clusters = clusters)
```

`vulnerability` *Node Vulnerability*

Description

Computes the vulnerability of each node, defined as the relative drop in global efficiency when that node is removed from the network.

Usage

```
vulnerability(
  x,
  directed = NULL,
  normalized = TRUE,
  weighted = FALSE,
  invert_weights = TRUE,
  alpha = 1,
  digits = NULL,
  ...
)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
`normalized`	Logical. If TRUE (default), return the proportional drop. If FALSE, return the raw efficiency difference.
`weighted`	Logical. If TRUE, edge weights are used when computing shortest paths (Dijkstra). If FALSE (default), all edges have unit length.
`invert_weights`	Logical. If TRUE (default) and weights are present, weights are converted to distances $1/w^{\alpha}$, so that a higher weight gives a shorter path, as in `centrality()`. If FALSE, the weights are used as distances. Ignored when `weighted = FALSE`.

`alpha`	Weight-to-distance exponent (default 1). Used only when `weighted = TRUE` and `invert_weights = TRUE`.
`digits`	Integer or NULL. Round scores to this many decimal places. Default NULL (no rounding).
`...`	Passed to `to_igraph`, which takes no further arguments, so any argument supplied here raises an error.

Details

$$V(i) = \frac{E_{global} - E_{global \setminus i}}{E_{global}}$$

where E_{global} is the global efficiency of the full network and $E_{global \setminus i}$ is the global efficiency after removing node i and all its edges.

Global efficiency is defined as:

$$E_{global} = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{1}{d(i,j)}$$

In a directed network the distances follow the edge directions. $E_{global \setminus i}$ is computed on the reduced graph with the original $n(n-1)$ denominator, so vulnerability is never negative (Latora & Marchiori 2007).

Nodes with high vulnerability are critical to the communication efficiency of the network, because their removal causes the greatest drop in global efficiency.

All-pairs shortest paths are computed once for the full graph and once per removed node. A warning is issued for networks with more than 500 nodes.

Value

A data frame of class `"cograph_vulnerability"` with one row per node and columns:

node Node labels.

vulnerability Vulnerability score.

Rows are sorted by decreasing vulnerability. The original input network (`"network"`) and the normalization mode (`"normalized"`) are stored as attributes. Scores are `NA` for a network with at most one node, and all zero when the full network already has zero global efficiency.

Printing and plotting

Printing the result shows the normalization mode and the node table. `plot()` on the result is documented in `plot-results`.

References

Latora, V. & Marchiori, M. (2007). A measure of centrality based on network efficiency. *New Journal of Physics*, 9(6), 188. doi:10.1088/13672630/9/6/188

See Also

network_global_efficiency, robustness, centrality

Examples

```
vulnerability(regulation_net)
```

Index

∗ **datasets**
  hai_datasets, 275
  regulation_net, 393
  student_interactions, 462
∗ **motifs**
  extract_motifs, 256
  extract_triads, 259
  get_edge_list, 268
  motif_census, 291
  motifs, 292
  subgraphs, 463
  triad_census, 478

abbrev_label, 9, *355*, *358*
add_edges, 10, *11*, *23*, *310*, *395*
add_nodes, *10*, 11, *310*, *396*
aggregate_layers, 12, *310*
aggregate_weights, 13, *242*
ai_coding *(*hai_datasets*)*, 275
ai_detailed *(*hai_datasets*)*, 275
as.data.frame, *14*, *271*, *294*
as.data.frame.cograph_motif_result *(*motifs*)*, 292
as.data.frame.cograph_network, *310*
as.data.frame.cograph_network *(*as_cograph*)*, 13
as_cograph, 13, *269*, *272*, *278*, *310*, *474*, *476, 477*
as_mcml, 15, *464*
as_tna, *16*, 17, *242*, *244*, *361*, *465*
assortativity, 18, *21*
assortativity_attribute, 20, *20*

barplot, *245*, *320*, *336*, *347*
binarize, 22, *238*, *309*, *312*, *473*
bind_networks, *10*, 23, *310*
bootstrap, *248*

centrality, *20*, 24, *41–203*, *240*, *251*, *254, 255*, *262*, *264*, *274*, *289, 290*, *297*, *331–335*, *337*, *342*, *401*, *403*, *412*, *414*, *416*, *422*, *480*, *483*
centrality_access_information, *27*, 41
centrality_adaptive_leaderrank, *32*, 42, *198*
centrality_alpha, 43, *91*, *126*
centrality_authority, 44, *91*, *182*
centrality_average_distance, 45, *47*, *199*
centrality_barycenter, 46, *46*, *60*
centrality_beta_measure, *32*, 47
centrality_betweenness, 48, *50, 51*, *65*, *73*, *94, 95*, *103*, *132*, *136*, *138*, *166*, *170*, *179*, *187*, *194*
centrality_bottleneck, 49
centrality_bridging, 50, *52*, *69*, *101*, *140*, *175*, *192*
centrality_bridging_capital, *32*, 51
centrality_bridging_coefficient, *147, 148*
centrality_bridging_coefficient *(*centrality_truss*)*, 190
centrality_brokerage_coordinator, *27*, *52*, *54–57*
centrality_brokerage_gatekeeper, 53, *53*, *57*
centrality_brokerage_itinerant, *53*, 54, *56*
centrality_brokerage_liaison, *53*, 55, *55*
centrality_brokerage_representative, *53, 54*, 56
centrality_cda, *31*, 57
centrality_centroid, 58, *61*
centrality_closeness, *42*, *46, 47*, 59, *59*, *82*, *88*, *106*, *111, 112*, *120*, *122, 123*, *135*, *171*, *180*, *199*
centrality_closeness_vitality, *59*, 61, *95*, *162*
centrality_clusterrank, 62, *190*
centrality_coleman_theil, 63

centrality_comm_centrality (centrality_community_based), 66
centrality_communicability, 64, *65*, *188*
centrality_communicability_betweenness, *64*, 65
centrality_community_based, *27*, 66, *68*, *149*
centrality_community_hub_bridge, *27*, 67, *67*
centrality_community_mediator (centrality_community_based), 66
centrality_constraint, *63*, 68, *89*
centrality_controlrank, *40*, 69
centrality_coreness, 70, *72*, *99*, *110*, *139*, *145*, *181*, *192*, *197*
centrality_cross_clique, 71, *151*
centrality_current_flow_betweenness, 72, *74*, *103*, *179*, *194*
centrality_current_flow_closeness, 73, *73*, *122*, *172*, *178*
centrality_dangalchev, 74, *76*, *176, 177*
centrality_decay, 75, *75*, *105, 106*, *177*
centrality_degree, *71*, 76, *130*, *133*, *139*, *148*, *159*, *183, 184*, *186*, *195*, *203*, *290*
centrality_degree_discount, *27*, *30*, 77
centrality_delta_betweenness (centrality_length_scaled_betweenness), 131
centrality_delta_closeness (centrality_length_scaled_betweenness), 131
centrality_diffusion, 78, *80*, *95, 96*
centrality_diffusion_centrality, *31*, *52*, 79, *79*, *87*
centrality_dil, *39*, 80
centrality_distance_entropy, *27*, 81, *92*, *94*
centrality_diversity, 82, *92*
centrality_dkgm, *32*, *38*, 83, *113*
centrality_dmnc, 84, *129*, *154*
centrality_dynamical_importance, *38*, 85, *178*, *203*
centrality_dynamics_sensitive, *31*, *80*, 87
centrality_eccentricity, 88
centrality_effective_size, *63*, *69*, 89, *115*
centrality_ego_betweenness, *147*
centrality_ego_betweenness (centrality_length_scaled_betweenness), 131
centrality_ehcc, *38*, 90, *113*
centrality_eigenvector, *44, 45*, *86*, 91, *126*, *160*, *163*, *167*, *185*, *188*
centrality_enrenew (centrality_wvoterank), 200
centrality_entropy, *83*, 92, *127*
centrality_entropy_variation, *27*, 93
centrality_epc, *151*
centrality_epc (centrality_local_efficiency), 143
centrality_exogenous, *31*, 94
centrality_expected, *62*, *79*, 95, *97*
centrality_expected_force, 96, *155*
centrality_expected_influence_1, 97, *99*
centrality_expected_influence_2, 98, *98*
centrality_extended_coreness, 99, *100*, *158*
centrality_extended_gravity, *99*, 100, *110*
centrality_extended_local_bridging, 101, *147*
centrality_extended_mixed_gravity, 102, *153*
centrality_flow_betweenness, 103
centrality_flow_coefficient (centrality_heatmap), 113
centrality_fragmentation (centrality_local_efficiency), 143
centrality_fuzzy_local_dimension (centrality_local_dimension_fixed), 142
centrality_gateway, *53*, 104, *165*, *200*
centrality_generalized_closeness, *76*, 105
centrality_geodesic_kpath, *128*, *145*
centrality_geodesic_kpath (centrality_weighted_kshell), 196
centrality_gilschmidt, 106
centrality_global_structure, 107, *117*,

121
centrality_godfather *(*centrality_truss*)*, 190
centrality_graph_regularization, *32*, 108
centrality_gravity, *27*, *100*, *102*, 109, *152*, *153*
centrality_harary, 110
centrality_harmonic, *60*, *106*, 111, *111*, *120*, *128*, *132*, *135*, *174*
centrality_hcc, *33*, *38*, *81*, *90*, 112, *134*
centrality_heatmap, *28*, 113
centrality_hide_information *(*centrality_access_information*)*, 41
centrality_hub *(*centrality_authority*)*, 44
centrality_hubbell, *27*, 115, *126*
centrality_hybrid_global_structure, *107*, 116, *121*
centrality_iec, *39*, 117
centrality_iira, *33*, *38*, 118, *125*
centrality_improved_closeness, 119
centrality_improved_global_structure, *107*, *117*, 121
centrality_incloseness *(*centrality_closeness*)*, 59
centrality_indegree, *48*
centrality_indegree *(*centrality_degree*)*, 76
centrality_ineccentricity *(*centrality_eccentricity*)*, 88
centrality_information, *27*, *74*, *108*, 122
centrality_inharmonic *(*centrality_harmonic*)*, 111
centrality_instrength *(*centrality_strength*)*, 185
centrality_integration, 123, *171*
centrality_ira, *32*, *33*, *38*, *118*, *119*, 124
centrality_katz, *27*, *44*, *116*, 125, *167*
centrality_ked, *38*, 126, *134*, *137*
centrality_kpath *(*centrality_local_efficiency*)*, 143
centrality_kreach, 127, *195*
centrality_lac, 128, *154*, *189*
centrality_laplacian, *70*, *108*, 129, *183*
centrality_leaderrank, 130, *163*, *185*, *198*
centrality_length_scaled_betweenness, *28*, 131
centrality_leverage, 132
centrality_lhc, *33*, *39*, *81*, 133
centrality_lin, 134
centrality_linerank, *32*, 135
centrality_lnc, *38*, *127*, 137
centrality_load, *49*, 138, *166*, *187*
centrality_lobby, *43*, 139
centrality_local_bridging, *51*, 140, *147*
centrality_local_dimension, *27*, *82*, 141, *143*, *146*
centrality_local_dimension_fixed, *27*, *141*, 142, *146*
centrality_local_efficiency, *28*, 143
centrality_local_entropy *(*centrality_heatmap*)*, 113
centrality_local_information_dimension, *27*, *141*, *143*, 145
centrality_local_volume_dimension *(*centrality_local_dimension_fixed*)*, 142
centrality_localized_bridging, *101*, *140*, 146
centrality_malatya, 147
centrality_map_equation, *32*, 148
centrality_markov, *118*, 149, *172*
centrality_mcc, *38*, *85*, 150
centrality_mcgm, *32*, *84*, 151
centrality_mdd *(*centrality_truss*)*, 190
centrality_mixed_gravity, *84*, *102*, 152, *152*
centrality_mnc, *85*, *129*, 153
centrality_modified_expected_force, *97*, 154
centrality_modularity_vitality, *27*, *68*, 155
centrality_ncvoterank, *28*, *30*, *78*, 156, *196*, *202*
centrality_neighbor_distance, *32*, *38*, 157
centrality_neighborhood_connectivity, *27*, *96*, *133*, 159
centrality_ninl, *32*, 160
centrality_node_contraction, *28*, 161
centrality_node_contraction_improved *(*centrality_node_contraction*)*, 161

centrality_outcloseness
(centrality_closeness), 59
centrality_outdegree
(centrality_degree), 76
centrality_outeccentricity
(centrality_eccentricity), 88
centrality_outharmonic
(centrality_harmonic), 111
centrality_outstrength
(centrality_strength), 185
centrality_pagerank, *45*, *91*, *131*, *136*, *150*, 162, *173*, *182*, *193*
centrality_pairwisedis, *27*, 163
centrality_participation, *67*, *105*, *149*, *156*, 164, *200*
centrality_percolation, 165
centrality_power, *116*, 166
centrality_prestige_domain, *48*, *164*, 167, *169*
centrality_prestige_domain_proximity, 168, *168*
centrality_proximal_betweenness, *32*, 169
centrality_radiality, *88*, *123*, 170
centrality_random_walk, *118*, *150*, 171, *173*
centrality_random_walk_decay, *32*, 172
centrality_reaching_local, *27*, *112*, *168*, *169*, 173, *392*, *393*
centrality_redundancy, *89*
centrality_redundancy
(centrality_heatmap), 113
centrality_relative_entropy, *33*, *39*, 174
centrality_renewed_coreness
(centrality_weighted_kshell), 196
centrality_residual_closeness, *75*, 176
centrality_resistance_curvature, *38*, *86*, 177
centrality_rsp_betweenness, *33*, *39*, 178
centrality_rumor, *27*, 179
centrality_s_core, *71*
centrality_s_core
(centrality_local_efficiency), 143
centrality_s_shell, *27*, *30*, 180, *197*
centrality_salsa, 181
centrality_semilocal, *130*, *137*, *158*, *160*, 182, *191*
centrality_shapley_game1, *27*, 183
centrality_shapley_game2, *30*
centrality_shapley_game2
(centrality_shapley_game1), 183
centrality_shapley_game3, *30*
centrality_shapley_game3
(centrality_shapley_game1), 183
centrality_single_discount
(centrality_degree_discount), *77*
centrality_spectralrank, *32*, *70*, 184
centrality_strength, *58*, *76*, *77*, *83*, *98*, *99*, 185
centrality_stress, *49*, *50*, *138*, *170*, 186
centrality_subgraph, *64*, 187, *255*
centrality_support (centrality_truss), 190
centrality_topological_coefficient, 188, *190*
centrality_transitivity, *58*, *62*, *72*, *115*, 189, *189*
centrality_truss, *31*, *38*, 190
centrality_trust_pagerank, *33*, *39*, 192
centrality_two_way_rw, *28*, 193
centrality_volume, *31*, 194
centrality_voterank, *78*, *157*, 195, *202*
centrality_voterank_plus
(centrality_wvoterank), 200
centrality_weighted_h_index
(centrality_heatmap), 113
centrality_weighted_kshell, *28*, *181*, 196
centrality_weighted_leaderrank, *31*, *43*, *130*, *131*, 197
centrality_wiener, *61*, 198
centrality_within_module_z, *105*, *156*, *165*, 199
centrality_wvoterank, *28*, *157*, *196*, 200
centrality_x_degree, 202
centralization, 203
cluster_quality, 204, *204*, *207*
cluster_significance, 205
cnet (summarize_network), 465
coding (hai_datasets), 275
coding_detailed (hai_datasets), 275
cograph, 207, *267*, *268*, *427*, *429*, *432*, *433*, *436*, *444*, *462*
CographLayout, 208

CographNetwork, 210, *270*
CographTheme, 215
color_communities, 217
com_consensus *(*community_consensus*)*, 220
com_eb *(*community_edge_betweenness*)*, 222
com_fg *(*community_fast_greedy*)*, 223
com_fl *(*community_fluid*)*, 224
com_im *(*community_infomap*)*, 225
com_ld *(*community_leiden*)*, 229
com_le *(*community_leading_eigenvector*)*, 228
com_lp *(*community_label_propagation*)*, 227
com_lv *(*community_louvain*)*, 231
com_op *(*community_optimal*)*, 232
com_sg *(*community_spinglass*)*, 233
com_wt *(*community_walktrap*)*, 235
communities, *205*, *207*, 218, *221*, *223*, *225*, *226*, *228–232*, *235*, *236*, *315*
community_consensus, 220
community_edge_betweenness, *220*, 222
community_fast_greedy, *220*, 223
community_fluid, *220*, 224
community_infomap, *220*, 225
community_label_propagation, *220*, 227
community_leading_eigenvector, *220*, 228
community_leiden, *220*, 229
community_louvain, *220*, *221*, 231
community_optimal, *220*, 232
community_sizes, 233
community_spinglass, *220*, 233
community_walktrap, *220*, 235
compare_communities, 236
complement_network, 237, *310*
contract_nodes, 238, *310*
core_periphery, 239, *315*
cqual *(*cluster_quality*)*, 204
csig *(*cluster_significance*)*, 205
csum, *17*, *18*, *204*, 241, *357*, *361*, *362*, *372*, *464–466*, *480*

degree_distribution, 244, *264*, *342*
detect_communities, *21*, *66*, *68*, *155*, *217*, *218*, *239*, 246, *271*, *272*, *310*, *353*, *357*, *362*, *367*, *370*, *459*
disparity_filter, 247, *310*, *316*, *445*, *473*
dispersion, 248, *255*
dyad_census, 249

edge_betweenness *(*edge_centrality*)*, 251
edge_centrality, 251, *252*, *253*, *298*, *299*, *420*
edge_reciprocity, *250*, 252
ego_networks, 253, *310*
estrada_index, 255
extract_interlayer *(*supra_interlayer*)*, 468
extract_layer *(*supra_layer*)*, 469
extract_motifs, 256, *260*, *269*, *292*, *296*, *315*, *463*, *479*
extract_motifs(), *260*, *269*, *292*, *296*
extract_triads, *258*, 259, *269*, *292*, *296*, *463*, *479*
extract_triads(), *258*

filter_edges, 260, *263*, *309*, *394*, *395*, *410*, *422*, *473*
filter_nodes, 261, *261*, *309*, *395*, *396*, *415*
fit_degree_distribution, 263, *315*
from_qgraph, *208*, 265, *268*, *444*, *462*
from_tna, *208*, 267, *267*, *444*, *462*

get_data *(*get_nodes*)*, 269
get_edge_list, *258*, *260*, 268, *292*, *296*, *463*, *479*
get_edges, *474*
get_edges *(*get_nodes*)*, 269
get_groups *(*get_nodes*)*, 269
get_labels *(*get_nodes*)*, 269
get_layout *(*layout_registry*)*, 285
get_meta *(*get_nodes*)*, 269
get_nodes, *14*, *15*, 269
get_shape *(*shapes*)*, 418
get_source *(*get_nodes*)*, 269
get_theme *(*themes*)*, 471
ggplot_robustness, 272
group_centrality, 273

hai_datasets, 275
hist, *245*, *320*
homophily *(*assortativity_attribute*)*, 20
htna *(*plot_htna*)*, 351
human_ai *(*hai_datasets*)*, 275
human_ai_detailed *(*hai_datasets*)*, 275

invert_weights, 276, *309*, *312*
is_bipartite, 277, *392*
is_directed *(*get_nodes*)*, 269

is_tna_network, 278

k_shortest_paths, 278, *419*

label_abbrev *(*abbrev_label*)*, 9
lagg *(*aggregate_layers*)*, 12
layer_degree_correlation, 280
layer_similarity, 280, *281*
layer_similarity_matrix, 281
layout_circle, 282, *285*
layout_groups, 283, *285*
layout_oval, 284, *285*
layout_registry, 285
layout_saqr, 286
layout_spring, *285*, 287
layout_target, 288
ldegcor *(*layer_degree_correlation*)*, 280
list_centralities, *26*, *28*, *40*, *175*, *262*, 289, *414*, *416*
list_layouts, *207*
list_layouts *(*layout_registry*)*, 285
list_layouts(), *209*
list_palettes, *433*
list_palettes *(*palettes*)*, 313
list_shapes, *430*
list_shapes *(*shapes*)*, 418
list_svg_shapes *(*shapes*)*, 418
list_themes, *436*
list_themes *(*themes*)*, 471
lsim *(*layer_similarity*)*, 280
lsim_matrix *(*layer_similarity_matrix*)*, 281

membership, 290
mlna *(*plot_mlna*)*, 366
motif_census, *258*, *260*, *269*, 291, *296*, *315*, *463*, *479*
motif_census(), *257–260*, *295, 296*, *479*
motifs, *258*, *260*, *269*, 292, *292*, *315*, *463*, *479*
motifs(), *257, 258*, *260*, *292*, *463*, *479*
mtna *(*plot_mtna*)*, 369
mutate_edges, *15*, 296, *297*, *310*
mutate_nodes, *11*, 297, *297*, *310*

n_communities, 298
n_edges *(*get_nodes*)*, 269
n_nodes *(*get_nodes*)*, 269
neighborhood_overlap, *255*, 298, *420*
network_bridges, 299
network_clique_size, 300
network_cut_vertices, 300
network_girth, 301
network_global_efficiency, 302, *483*
network_local_efficiency, *145*, 303
network_radius, 304
network_rich_club, 304
network_small_world, 305
network_summary, *20*, *240*, *250*, 306, *342*, *419*
network_vertex_connectivity, 308
network_wrangling, 309
nodes *(*get_nodes*)*, 269
normalize_weights, *22*, *277*, *309*, 311, *470*

overlay_communities, 312

palette_blues *(*palettes*)*, 313
palette_colorblind *(*palettes*)*, 313
palette_diverging *(*palettes*)*, 313
palette_pastel *(*palettes*)*, 313
palette_rainbow *(*palettes*)*, 313
palette_reds *(*palettes*)*, 313
palette_viridis *(*palettes*)*, 313
palettes, 313
panel_layout, 314, *320*, *344*, *373*, *451, 452*
plot, *245*, *320*, *336*, *342*
plot-results, 315
plot.cograph_communities *(*plot-results*)*, 315
plot.cograph_core_periphery *(*plot-results*)*, 315
plot.cograph_degree_fit *(*plot-results*)*, 315
plot.cograph_motif_analysis *(*plot-results*)*, 315
plot.cograph_motif_result *(*plot-results*)*, 315
plot.cograph_motifs *(*plot-results*)*, 315
plot.cograph_rich_club *(*plot-results*)*, 315
plot.cograph_vulnerability *(*plot-results*)*, 315
plot.tna_disparity *(*plot-results*)*, 315
plot_alluvial, 324, *386*
plot_bootstrap_forest, 327
plot_centrality, 331
plot_centrality_compare, 333
plot_centrality_distribution, 335
plot_centrality_heatmap, 336

plot_chord, 337
plot_compare, 340, *344*
plot_comparison_heatmap, 340, *344*
plot_degree_correlation, 342
plot_difference, *23*, *340*, 343
plot_edge_diff_forest, 345
plot_edge_weights, 347
plot_group_permutation *(*plot-results*)*, 315
plot_heatmap, 348, *392*
plot_htna, 351
plot_mcml, *18*, *242*, *244*, 355, *372*, *465, 466*
plot_mixed_network, 362, *451, 452*
plot_ml_heatmap, 364
plot_mlna, *361, 362*, *364*, 366, *368*, *380*
plot_motifs *(*plot-results*)*, 315
plot_mtna, *244*, *361, 362*, 369, *372*
plot_net_bootstrap_group *(*plot-results*)*, 315
plot_net_stability *(*plot-results*)*, 315
plot_netobject_group *(*plot-results*)*, 315
plot_netobject_ml *(*plot-results*)*, 315
plot_network_evolution, 372, *380*
plot_permutation *(*plot-results*)*, 315
plot_robustness, 374, *405*
plot_simplicial, 375
plot_temporal, 378
plot_tna, *371, 372*, 380
plot_trajectories, *327*, 383
plot_transitions, *327*, 386, *386*
project_bipartite, 391

reaching_global, *174*, 392
register_layout *(*layout_registry*)*, 285
register_shape *(*shapes*)*, 418
register_svg_shape, *430*
register_svg_shape *(*shapes*)*, 418
register_theme *(*themes*)*, 471
regulation_net, 393
remove_edges, *10*, *310*, 394
remove_isolates, *261*, *310*, 395, *395, 396*, *409*, *446*, *473*
remove_nodes, *11*, *310*, 396
rename_nodes, *310*, 396, *399*
render-ggplot, 397
render-grid, 398
reorder_nodes, *310*, *397*, 398
reverse_edges, *310*, 399, *475*
rich_club, *305*, *315*, 400, *403*
rich_club_local, *401*, 402
robustness, *164*, *374, 375*, *401*, 403, *405, 406*, *483*
robustness_auc, 405, *405*
robustness_summary, 406

select_bridges, 406, *410*
select_component, *309*, 407, *413*, *415, 416*, *446*
select_edges, *296, 297*, *309*, *407*, 408, *410, 411*, *417*
select_edges_between, 410, *411*
select_edges_involving, *410*, 411
select_k_core, *309*, 412
select_neighbors, *255*, *309*, *408*, 413, *415*
select_nodes, *262*, *297*, *309, 310*, *399*, *407, 408*, *410*, *412, 413*, 414, *416*
select_top, *309*, *415*, 416, *417*
select_top_edges, *410*, 417
set_edges *(*get_nodes*)*, 269
set_groups *(*get_nodes*)*, 269
set_layout *(*get_nodes*)*, 269
set_nodes, *397*
set_nodes *(*get_nodes*)*, 269
shapes, 418
shortest_paths, *277*, *279*, *310*, 419
simmelian_strength, *299*, 420
simplify, *310*, 421
sn_edges, *208*, 423, *429*, *432*, *436*, *444*, *462*
sn_ggplot, *397, 398*, 427
sn_layout, *207, 208*, *285*, *427*, 428, *432*, *444*, *462*
sn_nodes, *208*, *418*, *427*, 429, *429*, *433*, *436*, *444*, *462*
sn_palette, *208*, *313*, 433, *436*
sn_render, *14*
sn_render *(*soplot*)*, 436
sn_save, 434
sn_save_ggplot, 434
sn_theme, *208*, *427*, *429*, *432, 433*, 435, *444*, *462*, *471*
soplot, *208*, *267, 268*, *285*, *398*, *418*, *423*, *427*, *429*, *432, 433*, 436, *436*, *447*, *462*, *471*
spanning_tree, *310*, 444
split_components, *239*, *309*, *395*, 446
splot, *14, 15*, *208*, *218*, *261*, *263*, *267, 268*, *272*, *285*, *312, 313*, *315*, *320*, *323*,

343, *344*, *354*, *363*, *373*, *418*, *423*, *429*, *432*, *433*, *436*, *444*, *465*, *471*
splot *(*splot.net_bootstrap*)*, 447
splot.group_tna_permutation *(*plot-results*)*, 315
splot.net_bootstrap, 447
splot.tna_bootstrap *(*plot-results*)*, 315
splot.tna_disparity *(*plot-results*)*, 315
splot.tna_permutation *(*plot-results*)*, 315
student_interactions, 462
subgraphs, *258*, *260*, *269*, *292*, *296*, *315*, 463, *479*
subgraphs(), *258*, *260*, *296*
subset_edges, *261*
subset_edges *(*filter_edges*)*, 260
subset_nodes, *263*
subset_nodes *(*filter_nodes*)*, 261
summarize_clusters, *16*, *17*, *238*, *239*, *310*, 464
summarize_network, *393*, 465, *466*
supra *(*supra_adjacency*)*, 467
supra_adjacency, 467, *468*, *469*
supra_interlayer, 468, *468*
supra_layer, *468*, 469
symmetrize, *309*, *312*, 469, *478*

theme_cograph_classic *(*themes*)*, 471
theme_cograph_colorblind *(*themes*)*, 471
theme_cograph_dark *(*themes*)*, 471
theme_cograph_gray *(*themes*)*, 471
theme_cograph_minimal *(*themes*)*, 471
theme_cograph_nature *(*themes*)*, 471
theme_cograph_viridis *(*themes*)*, 471
themes, 471
threshold_edges, *22*, *309*, *445*, 472
tna::tna(), *269*
to_cograph *(*as_cograph*)*, 13
to_data_frame, 473, *476*
to_df, *15*, *310*, *474*, *476*, *477*
to_df *(*to_data_frame*)*, 473
to_directed, *310*, *399*, 474, *478*
to_igraph, *223–227*, *229–232*, *234*, *236*, *239*, *250*, *254*, *299–306*, *309*, *310*, *400*, *402*, *404*, *474*, 475, *476*, *477*, *482*
to_matrix, *310*, 476, *477*
to_network, *310*, *476*, 477
to_undirected, *231*, *238*, *310*, *399*, *470*, *475*, *477*
tplot, *355*
tplot *(*plot_tna*)*, 380
triad_census, *249*, *250*, *258*, *260*, *269*, *292*, *296*, *463*, 478
trophic_incoherence, 479

unregister_svg_shape *(*shapes*)*, 418

verify_igraph *(*verify_with_igraph*)*, 480
verify_with_igraph, 480
vulnerability, *315*, 481

wagg *(*aggregate_weights*)*, 13