

Package ‘cograph’

September 30, 2026

Title Analysis and Visualization of Complex Networks

Version 2.7.2

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

RoxygenNote 7.3.3

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2026-09-30 16:20:02 UTC

Contents

abbrev_label	10
add_edges	11
add_nodes	12
aggregate_layers	13
aggregate_weights	14
as.data.frame.cograph_motif_result	15
as.data.frame.cograph_network	16
assortativity	17
assortativity_attribute	19
as_cograph	20
as_mcml	22
as_tna	23
binarize	25
bind_networks	26
centrality	28
centrality_access_information	45
centrality_adaptive_leaderrank	46
centrality_alpha	48
centrality_authority	49
centrality_average_distance	49
centrality_barycenter	50
centrality_beta_measure	51
centrality_betweenness	52
centrality_bottleneck	53
centrality_bridging	53
centrality_bridging_capital	54
centrality_brokerage_coordinator	55
centrality_brokerage_gatekeeper	56
centrality_brokerage_itinerant	57
centrality_brokerage_liaison	58
centrality_brokerage_representative	59
centrality_cda	60
centrality_centroid	61
centrality_closeness	62
centrality_closeness_vitality	63
centrality_clusterrank	64
centrality_coleman_theil	64
centrality_communicability	66
centrality_communicability_betweenness	66
centrality_community_based	67
centrality_community_hub_bridge	69
centrality_constraint	70

centrality_controlrank	71
centrality_coreness	72
centrality_cross_clique	73
centrality_current_flow_betweenness	74
centrality_current_flow_closeness	74
centrality_dangalchev	75
centrality_decay	76
centrality_degree	77
centrality_degree_discount	78
centrality_diffusion	79
centrality_diffusion_centrality	80
centrality_dil	81
centrality_distance_entropy	84
centrality_diversity	85
centrality_dkgm	86
centrality_dmnc	87
centrality_dynamical_importance	88
centrality_dynamics_sensitive	89
centrality_eccentricity	91
centrality_effective_size	92
centrality_ehcc	92
centrality_eigenvector	93
centrality_entropy	94
centrality_entropy_variation	95
centrality_exogenous	96
centrality_expected	98
centrality_expected_force	98
centrality_expected_influence_1	100
centrality_expected_influence_2	101
centrality_extended_coreness	102
centrality_extended_gravity	103
centrality_extended_local_bridging	104
centrality_extended_mixed_gravity	105
centrality_flow_betweenness	106
centrality_gateway	107
centrality_generalized_closeness	107
centrality_giltschmidt	108
centrality_global_structure	109
centrality_graph_regularization	110
centrality_gravity	111
centrality_harary	113
centrality_harmonic	113
centrality_hcc	114
centrality_heatmap	116
centrality_hubbell	118
centrality_hybrid_global_structure	119
centrality_iec	120
centrality_iira	122

centrality_improved_closeness	124
centrality_improved_global_structure	125
centrality_information	126
centrality_integration	127
centrality_ira	128
centrality_katz	130
centrality_ked	131
centrality_kreach	133
centrality_lac	133
centrality_laplacian	134
centrality_leaderrank	135
centrality_length_scaled_betweenness	136
centrality_leverage	137
centrality_lhc	138
centrality_lin	140
centrality_linerank	141
centrality_lnc	142
centrality_load	144
centrality_lobby	145
centrality_localized_bridging	146
centrality_local_bridging	147
centrality_local_dimension	148
centrality_local_dimension_fixed	149
centrality_local_efficiency	150
centrality_local_information_dimension	152
centrality_malatya	153
centrality_map_equation	154
centrality_markov	156
centrality_mcc	157
centrality_mcgm	158
centrality_mixed_gravity	159
centrality_mnc	160
centrality_modified_expected_force	161
centrality_modularity_vitality	162
centrality_ncvoterank	163
centrality_neighborhood_connectivity	164
centrality_neighbor_distance	165
centrality_ninl	167
centrality_node_contraction	169
centrality_pagerank	170
centrality_pairwisedis	171
centrality_participation	172
centrality_percolation	172
centrality_power	173
centrality_prestige_domain	174
centrality_prestige_domain_proximity	175
centrality_proximal_betweenness	176
centrality_radiality	177

centrality_random_walk	178
centrality_random_walk_decay	179
centrality_reaching_local	180
centrality_relative_entropy	181
centrality_residual_closeness	184
centrality_resistance_curvature	185
centrality_rsp_betweenness	186
centrality_rumor	188
centrality_salsa	190
centrality_semilocal	190
centrality_shapley_game1	191
centrality_spectralrank	192
centrality_strength	194
centrality_stress	195
centrality_subgraph	196
centrality_s_shell	196
centrality_topological_coefficient	198
centrality_transitivity	198
centrality_truss	199
centrality_trust_pagerank	201
centrality_two_way_rw	204
centrality_volume	205
centrality_voterank	206
centrality_weighted_kshell	206
centrality_weighted_leaderrank	208
centrality_wiener	209
centrality_within_module_z	210
centrality_wvoterank	211
centrality_x_degree	212
centralization	213
cluster_quality	214
cluster_significance	215
cograph	217
CographLayout	219
CographNetwork	220
CographTheme	225
color_communities	228
communities	229
community_consensus	231
community_edge_betweenness	233
community_fast_greedy	234
community_fluid	235
community_infomap	236
community_label_propagation	238
community_leading_eigenvector	239
community_leiden	241
community_louvain	242
community_optimal	243

community_sizes	244
community_springlass	245
community_walktrap	247
compare_communities	248
complement_network	249
contract_nodes	250
core_periphery	251
csum	253
degree_distribution	256
detect_communities	258
disparity_filter	259
dispersion	261
dyad_census	262
edge centrality	264
edge_reciprocity	266
ego_networks	267
estrada_index	268
extract_motifs	269
extract_triads	273
filter_edges	274
filter_nodes	276
fit_degree_distribution	277
from_qgraph	279
from_tna	282
get_data	284
get_edges	284
get_edge_list	285
get_groups	286
get_labels	287
get_layout	287
get_meta	288
get_nodes	289
get_shape	289
get_source	290
get_theme	291
ggplot_robustness	291
group centrality	292
hai_datasets	294
invert_weights	296
is_bipartite	297
is_directed	298
is_tna_network	298
k_shortest_paths	299
layer_degree_correlation	301
layer_similarity	301
layer_similarity_matrix	302
layout_circle	303
layout_groups	304

layout_oval	305
layout_saqr	306
layout_spring	307
layout_target	308
list_centralities	309
list_layouts	310
list_palettes	311
list_shapes	311
list_svg_shapes	312
list_themes	312
membership	313
motifs	313
motif_census	319
mutate_edges	320
mutate_nodes	321
neighborhood_overlap	322
network_bridges	323
network_clique_size	324
network_cut_vertices	325
network_girth	325
network_global_efficiency	326
network_local_efficiency	327
network_radius	328
network_rich_club	329
network_small_world	330
network_summary	331
network_vertex_connectivity	333
network_wrangling	334
nodes	336
normalize_weights	337
n_communities	338
n_edges	339
n_nodes	339
overlay_communities	340
palettes	341
palette_blues	341
palette_colorblind	342
palette_diverging	342
palette_pastel	343
palette_rainbow	343
palette_reds	344
palette_viridis	344
panel_layout	345
plot.cograph_cluster_significance	346
plot.cograph_communities	347
plot.cograph_core_periphery	347
plot.cograph_degree_fit	348
plot.cograph_motifs	349

plot.cograph_motif_analysis	351
plot.cograph_network	353
plot.cograph_rich_club	353
plot.cograph_vulnerability	354
plot.tna_bootstrap	355
plot.tna_disparity	357
plot_alluvial	357
plot_bootstrap_forest	361
plot_centrality	365
plot_centrality_compare	367
plot_centrality_distribution	368
plot_centrality_heatmap	370
plot_chord	371
plot_compare	373
plot_comparison_heatmap	374
plot_degree_correlation	375
plot_difference	376
plot_edge_diff_forest	379
plot_edge_weights	380
plot_heatmap	382
plot_htna	384
plot_mcml	389
plot_mixed_network	399
plot_mlna	400
plot_ml_heatmap	404
plot_motifs	406
plot_mtna	408
plot_netobject_group	411
plot_netobject_ml	412
plot_network_evolution	413
plot_net_bootstrap_group	414
plot_net_stability	415
plot_robustness	416
plot_simplicial	417
plot_temporal	421
plot_tna	423
plot_trajectories	426
plot_transitions	429
print.cograph_communities	433
print.cograph_degree_fit	434
print.cograph_network	435
project_bipartite	435
reaching_global	437
register_layout	438
register_shape	439
register_svg_shape	439
register_theme	440
regulation_net	441

remove_edges	442
remove_isolates	443
remove_nodes	444
rename_nodes	444
render-ggplot	445
render-grid	446
reorder_nodes	446
reverse_edges	447
rich_club	448
rich_club_local	450
robustness	451
robustness_auc	453
robustness_summary	454
select_bridges	455
select_component	456
select_edges	457
select_edges_between	459
select_edges_involving	460
select_k_core	461
select_neighbors	462
select_nodes	463
select_top	465
select_top_edges	466
set_edges	467
set_groups	468
set_layout	469
set_nodes	470
shortest_paths	471
simmelian_strength	472
simplify	473
sn_edges	475
sn_ggplot	480
sn_layout	481
sn_nodes	482
sn_palette	486
sn_save	487
sn_save_ggplot	487
sn_theme	488
soplot	489
spanning_tree	498
split_components	499
splot.group_tna_permutation	500
splot.net_bootstrap	501
splot.tna_disparity	516
splot.tna_permutation	517
student_interactions	519
subgraphs	520
summarize_clusters	521

summarize_network	523
summary.cograph_network	524
supra_adjacency	525
supra_interlayer	526
supra_layer	527
symmetrize	528
themes-builtin	529
theme_cograph_classic	530
theme_cograph_colorblind	530
theme_cograph_dark	531
theme_cograph_gray	531
theme_cograph_minimal	532
theme_cograph_nature	532
theme_cograph_viridis	533
threshold_edges	533
to_data_frame	535
to_directed	536
to_igraph	537
to_matrix	538
to_network	539
to_undirected	540
triad_census	541
trophic_incoherence	542
unregister_svg_shape	543
verify_with_igraph	543
vulnerability	544
Index	547

abbrev_label	<i>Abbreviate Labels</i>
--------------	--------------------------

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": Adaptive abbreviation based on label count
n_labels	Number of labels (used for "auto" mode). If NULL, uses length(label).

Value

Character vector of (possibly abbreviated) labels.

Examples

```
labels <- c("VeryLongStateName", "Short", "AnotherLongName")

# No abbreviation
abbrev_label(labels, NULL)

# Fixed max length
abbrev_label(labels, 5) # "Very. . .", "Short", "Anot. . ."

# Auto-adaptive
abbrev_label(labels, "auto")
```

add_edges	<i>Add Edges to a Network</i>
-----------	-------------------------------

Description

Add Edges to a Network

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

```
adj <- matrix(0, 3, 3, dimnames = list(LETTERS[1:3], LETTERS[1:3]))
adj["A", "B"] <- adj["B", "A"] <- 1

add_edges(adj, from = "B", to = "C", weight = 0.5)
```

add_nodes	<i>Add Nodes to a Network</i>
-----------	-------------------------------

Description

Add Nodes to a Network

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 <code>length(labels)</code> . 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

```
adj <- matrix(c(0, 1, 1, 0), 2, 2)
rownames(adj) <- colnames(adj) <- c("A", "B")

add_nodes(adj, labels = c("C", "D"))
add_nodes(adj, labels = "C", group = "new")
```

aggregate_layers	<i>Aggregate Layers</i>
------------------	-------------------------

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
method	Aggregation: "sum", "mean", "max", "min", "union", "intersection"
weights	Optional layer weights (for weighted sum)

Value

Aggregated adjacency matrix

Examples

```

nodes <- c("A", "B", "C")
l1 <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3, dimnames = list(nodes, nodes))
l2 <- matrix(c(0, 1, 1, 1, 0, 0, 1, 0, 0), 3, 3, dimnames = list(nodes, nodes))
layers <- list(L1 = l1, L2 = l2)

aggregate_layers(layers, "sum")           # total edge weight
aggregate_layers(layers, "mean")         # average edge weight
aggregate_layers(layers, "union")        # edge present in any layer
aggregate_layers(layers, "intersection") # edge present in every layer

```

aggregate_weights	<i>Aggregate Edge Weights</i>
-------------------	-------------------------------

Description

Aggregates a vector of edge weights using various methods. Compatible with igraph's `edge.attr.comb` parameter.

Usage

```

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

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

```

Arguments

<code>w</code>	Numeric vector of edge weights. NA and zero entries are dropped before aggregation.
<code>method</code>	Aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", "geomean". Default "sum". Any other value is an error.
<code>n_possible</code>	Number of possible edges (used only by <code>method = "density"</code> ; 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

```

w <- c(0.5, 0.8, 0.3, 0.9)
aggregate_weights(w, "sum") # 2.5
aggregate_weights(w, "mean") # 0.625
aggregate_weights(w, "max") # 0.9

```

```
as.data.frame.cograph_motif_result
```

Motif Results as a Data Frame

Description

Returns the tables held by a motif result from [motifs](#) or [subgraphs](#) as tidy data frames.

Usage

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

Arguments

<code>x</code>	A <code>cograph_motif_result</code> object.
<code>row.names</code> , <code>optional</code>	Standard as.data.frame arguments; <code>row.names</code> replaces the default row names.
<code>...</code>	Unused.
<code>what</code>	Which table to return. "results" (default) returns the main table: one row per triad type for a census, or one row per node triple and type for <code>subgraphs()</code> . "types" returns one row per triad type with its count: the number of triads of that type in a census, or the number of node triples of that type in <code>subgraphs()</code> .

Value

A data.frame. For `what = "results"` in a census, the columns are type and count, plus expected, z, p and sig when significance was tested. For `subgraphs()`, the columns are triad, node1, node2, node3, type and observed, plus the significance columns when tested. For `what = "types"`, the columns are type and count.

See Also

[motifs](#), [subgraphs](#)

Examples

```
census <- motifs(regulation_net, significance = FALSE)
as.data.frame(census)
as.data.frame(census, what = "types")
```

as.data.frame.cograph_network

Cograph Network as a Data Frame

Description

The tidy accessor for a `cograph_network`: one row per edge (or per node), with endpoints given as labels rather than internal indices, so no caller has to reach into the object with `$` or translate integer ids by hand.

Usage

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

Arguments

<code>x</code>	A <code>cograph_network</code> object.
<code>row.names</code>	<code>NULL</code> or a character vector of row names, as for as.data.frame .
<code>optional</code>	Logical, as for as.data.frame . Ignored; the column names of the returned table are always the documented ones.
<code>...</code>	Unused, for compatibility with the generic.
<code>what</code>	Which table to return. "edges" (default) or "nodes".

Value

A base data frame. For `what = "edges"`, one row per edge with columns `from` and `to` (node labels), `weight`, and any extra edge columns the network carries (for example `session`). For `what = "nodes"`, one row per node with the node metadata columns (`id`, `label`, layout coordinates, and any custom columns).

This is the accessor, so it hands back everything the object holds, including columns `mutate_edges` computed. `to_df` is the narrower conversion verb: it returns `from`, `to` and `weight` only.

See Also

[to_df](#), [get_edges](#), [get_nodes](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
net <- as_cograph(adj)

as.data.frame(net)
as.data.frame(net, what = "nodes")
```

assortativity	<i>Degree Assortativity Coefficient</i>
---------------	---

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>type</code>	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.
<code>digits</code>	Integer or NULL. Round result to this many decimal places. Default NULL (no rounding).
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

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 (rather than excess) degrees at the two ends of each edge, counting every undirected edge in both orientations; this is numerically identical to the formula above.

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 Integer: number of nodes.

n_edges Integer: 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](https://doi.org/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](https://doi.org/10.1073/pnas.0912671107)

See Also

[assortativity_attribute](#), [centrality](#), [network_summary](#)

Examples

```
# Assortative network (high-degree connect to high-degree)
adj <- matrix(c(
  0, 1, 1, 1, 0,
  1, 0, 1, 1, 0,
  1, 1, 0, 0, 1,
  1, 1, 0, 0, 1,
  0, 0, 1, 1, 0
), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
cograph::assortativity(adj)
```

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 (names must match node names) or an unnamed vector in node order.
directed	Logical or NULL. If NULL (default), auto-detect.
digits	Integer or NULL. Round result. Default NULL.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

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.

n_nodes Integer.

n_edges Integer.

attribute_values The attribute values used.

network Original input.

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

```
adj <- matrix(c(0,1,1,0, 1,0,0,0, 1,0,0,1, 0,0,1,0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
groups <- c(A = "x", B = "x", C = "y", D = "y")
cograph::assortativity_attribute(adj, groups)
```

as_cograph

Convert to Cograph Network

Description

Creates a lightweight cograph_network object from various network inputs. The resulting object is a named list with all data accessible via \$.

Usage

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

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

Arguments

<code>x</code>	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 • An existing <code>cograph_network</code> object (returned as-is)
<code>directed</code>	Logical. Force directed interpretation. NULL for auto-detect.
<code>simplify</code>	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 edges are aggregated.
<code>...</code>	Additional arguments (currently unused).

Details

The `cograph_network` format is designed to be:

- Lean: Only essential data stored, computed values derived on demand
- Modern: Uses named list elements instead of attributes for clean \$ access
- Compatible: Works seamlessly with `splot()` and other `cograph` functions

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

Use getter functions for programmatic access: `get_nodes`, `get_edges`, `get_labels`, `n_nodes`, `n_edges`

Use setter functions to modify: `set_nodes`, `set_edges`, `set_layout`

Value

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 from, to, weight columns

`directed` Logical indicating if network is directed

`weights` Full nxn weight matrix when available for matrix/TNA round-trips, or NULL

`data` Original estimation data (sequence matrix, edge list, etc.), or NULL

`meta` Consolidated metadata list with sub-fields: `source` (input type string), `layout` (layout info list or NULL), `tna` (TNA metadata or NULL), and optionally `splot` (producer-supplied rendering hints read by `splot`)

`node_groups` Optional node groupings data frame

A `cograph_network` object. See `as_cograph`.

See Also

[get_nodes](#) to extract the nodes data frame, [get_edges](#) to extract edges as a data frame, [n_nodes](#) and [n_edges](#) for counts, [is_directed](#) to check directedness, [splot](#) for plotting

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_nodes(net)
get_edges(net)
splot(net)
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- to_cograph(mat)
```

as_mcml

*Convert to mcml***Description**

Convert various objects to the mcml class – a clean, tna-independent representation of a multilayer cluster network.

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	Object to convert.
...	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 <code>tna::group_model(cluster_data(...))</code>).
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 macro is the sequence of clusters, not a summary).

The input mcml object unchanged.

See Also

[summarize_clusters, as_tna](#)

Examples

```
# From cluster_summary
mat <- matrix(c(0.5, 0.2, 0.3,
               0.1, 0.6, 0.3,
               0.4, 0.1, 0.5), 3, 3, byrow = TRUE,
             dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
clusters <- list(G1 = c("A", "B"), G2 = c("C"))
cs <- csum(mat, clusters, type = "tna")
m <- as_mcml(cs)
m$macro$weights
```

as_tna

Convert cluster_summary to tna Objects

Description

Converts a cluster_summary object to proper tna objects that can be used with all functions from the tna package. Creates a macro (cluster-level) tna model and per-cluster tna models (internal transitions within each cluster), returned 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`. The `cluster_summary` should typically be created with `type = "tna"` to ensure row-normalized transition probabilities. If created with `type = "raw"`, the raw counts will be passed to `tna::tna()` which will normalize them.

Details

This is the final step in the MCML workflow, enabling full integration with the `tna` package for centrality analysis, bootstrap validation, permutation tests, and visualization.

Requirements:

The `tna` package must be installed. If not available, the function throws an error with installation instructions.

Workflow:

```
# Full MCML workflow
net <- cograph(edges, nodes = nodes)
net$nodes$clusters <- group_assignments
cs <- csum(net, type = "tna")
tna_models <- as_tna(cs)

# Now use tna package functions
plot(tna_models$macro)
tna::centralities(tna_models$macro)
tna::bootstrap(tna_models$macro, iter = 1000)

# Analyze per-cluster patterns
plot(tna_models$ClusterA)
tna::centralities(tna_models$ClusterA)
```

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 (S3 class) – a flat named list of `tna` objects. The first element is named `"macro"` and represents the cluster-level transitions. Subsequent elements are named by cluster name and represent internal transitions within each cluster.

macro A `tna` object representing cluster-level transitions. Contains `$weights` ($k \times k$ transition matrix), `$inits` (initial distribution), and `$labels` (cluster names). Use this for analyzing how learners/entities move between high-level groups or phases.

<cluster_name> Per-cluster tna objects, one per cluster. Each tna object represents internal transitions within that cluster. Contains `$weights` ($n_i \times 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"))
cs <- csum(regulation_net, clusters, type = "tna")
tna_models <- as_tna(cs)
names(tna_models)      # "macro", "C1", "C2", "C3"
splot(tna_models$macro) # cograph renderer avoids tna's plot deps
```

binarize

Binarize Edge Weights

Description

Replaces every surviving weight with 1, dropping edges at or below the threshold. The network equivalent of `sna::event2dichot()`.

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 <code>abs(weight)</code> . Default TRUE, so a correlation network keeps its strong negative edges.
signed	Logical. If TRUE, negative edges become -1 rather than 1, preserving 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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

binarize(adj, threshold = 0.45)
```

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

<code>x, y</code>	Network inputs.
<code>method</code>	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 <code>x</code> that are not in <code>y</code> , over the nodes of <code>x</code>
<code>weight</code>	How to combine the weights of an edge present in both: "sum" (default), "mean", "max", "min", or "first" (keep <code>x</code> 's weight).
<code>keep_format</code>	Logical. Return <code>x</code> 's format when TRUE.
<code>directed</code>	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

```
a <- matrix(0, 3, 3, dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
a["A", "B"] <- a["B", "A"] <- 1
b <- matrix(0, 3, 3, dimnames = list(c("B", "C", "D"), c("B", "C", "D")))
b["B", "C"] <- b["C", "B"] <- 2

bind_networks(a, b)
bind_networks(a, b, method = "difference")
```

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" (default) 6 canonical measures: degree, strength, closeness, betweenness, eigenvector, pagerank. "extended" Basic plus commonly-reported second-tier measures: 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, current_flow_closeness. "all" Every measure except the costly ones, which are held back (see include and list_centralities). Passing measures explicitly overrides type.
measures	Character vector of specific measure names to compute. When NULL (default) the tier selected by type is used. Accepts "all" as a shortcut for type = "all", i.e. every measure except the costly ones. Any custom vector of valid measure names is also accepted, and naming a costly measure there always computes it. Core (igraph-backed): "degree", "strength", "betweenness", "closeness", "eigenvector", "pagerank", "authority", "hub", "eccentricity", "coreness", "constraint", "transitivity", "harmonic", "alpha", "power", "subgraph". Native: "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/walk: "communicability", "communicability_betweenness", "random_walk".

Path-based: "stress", "flow_betweenness". **Local/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", "brokerage_coordinator", "brokerage_itinerant", "brokerage_representative", "brokerage_gatekeeper", "brokerage_liaison" (the last 5 also require a directed graph; see [centrality_brokerage_coordinator](#)). **Zoo (batch 2):** "gravity", "collective_influence", "local_hindex", "hindex_strength", "onion", "second_order", "infection", "nonbacktracking", "spanning_tree". **Classical (batch 3, reference-validated):** "katz" (Katz 1953), "hubbell" (Hubbell 1965), "information" (Stephenson-Zelen 1989), "reaching_local" (Mones et al. 2012). See [centrality_katz](#), [centrality_hubbell](#), [centrality_information](#), [centrality_pairwisedis](#), [centrality_reaching_local](#). **Psychometric (signed-weight):** "expected_influence_1", "expected_influence_2" (Robinaugh, Millner & McNally 2016). Expected influence keeps signed edge contributions, which is important when edges can be negative (partial-correlation, glasso, signed correlation networks). **Zoo (batch 7, lowest rank-redundancy with the rest of the package per the Centrality Zoo comparison):** "distance_entropy" (Stella & De Domenico 2018), "local_dimension" (Pu et al. 2014), "local_information_dimension" (Wen & Deng 2020), "neighborhood_connectivity" (Maslov & Sneppen 2002), and "modularity_vitality" (Magelinski et al. 2021; requires membership). The first three are hop-count measures and ignore edge weights. See [centrality_distance_entropy](#), [centrality_local_dimension](#), [centrality_local_information_dimension](#), [centrality_neighborhood_connectivity](#), [centrality_modularity_vitality](#). **Zoo (batch 8, the measures the Zoo comparison left "on the way"):** "shapley_game1", "shapley_game2", "shapley_game3" (Michalak et al. 2013), "access_information", "hide_information" (Rosvall et al. 2005), "rumor" (Shah & Zaman 2011), "community_hub_bridge" (Ghalmane et al. 2019; requires membership), "entropy_variation_degree", "entropy_variation_betweenness" (Ai 2017), "s_shell" (Liu et al. 2017), "degree_discount", "single_discount" (Chen, Wang & Yang 2009), "ncvoterank" (Kumar & Panda 2020). All are hop-count or topology-only measures; edge weights are ignored. See the per-measure pages, e.g. [centrality_shapley_game1](#), [centrality_access_information](#), [centrality_rumor](#), [centrality_community_hub_bridge](#), [centrality_entropy_variation](#), [centrality_s_shell](#), [centrality_degree_discount](#), [centrality_ncvoterank](#). **Zoo (batch 9, the remaining measures with a pinned definition):** community-aware "community_based" (Zhao et al. 2015), "comm_centrality" (Gupta et al. 2016), "community_mediator" (Tulu et al. 2018), all requiring membership; dimension family "local_dimension_fixed" (Silva & Costa 2013), "fuzzy_local_dimension" (Wen & Jiang 2019), "local_volume_dimension" (Li & Deng 2021); VoteRank family "wvoterank" (Sun et al. 2019), "enrenew" (Guo et al. 2020), "voterank_plus" (Liu et al. 2021); "node_contraction", "node_contraction_improved" (Tan et al. 2006; Wang et al. 2011); "two_way_rw" (Curado et al. 2022); local measures "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 & Everett 2006). Only "wvoterank", "two_way_rw" and "weighted_kshell" use edge weights. See [centrality_community_based](#), [centrality_local_dimension_fixed](#), [centrality_wvoterank](#), [centrality_node_contraction](#), [centrality_two_way_rw](#), [centrality_heatmap](#), [centrality_weighted_kshell](#).

Batch 10 closes the gaps other centrality packages had and cograph did not: "local_efficiency" (Latora & Marchiori 2001), "s_core" (Eidsaa & Almaas 2013), "fragmentation" (Borgatti 2006), "kpath" (Sade 1989) and "epc" (Lin et al. 2008). "fragmentation" and "epc" are costly, so `type = "all"` holds them back. See [centrality_local_efficiency](#).

Batch 11 tunes families cograph already had: "length_scaled_betweenness" (Brandes 2008), "delta_betweenness" and "delta_closeness" (Agneessens et al. 2017), "ego_betweenness" (Everett & Borgatti 2005). "gravity" gained `gravity_mass` and `gravity_radius`, and its formula was corrected – see [centrality_gravity](#). Bounded-distance ("k-") betweenness needs no measure of its own: it is `cutoff = k`. See [centrality_length_scaled_betweenness](#).

<code>include</code>	Character vector of costly measures to add back to a tier, or "costly" for all of them. <code>type = "all"</code> holds back the measures whose cost grows steeply with network size (see list_centralities), so that one call cannot take minutes by accident. Naming a measure in <code>measures</code> always computes it, whatever its cost. Default NULL.
<code>mode</code>	For directed networks: "all", "in", or "out". Affects measures whose output columns carry a mode suffix, including degree, strength, closeness, eccentricity, coreness, harmonic, diffusion, leverage, k-reach, distance-based measures, community-aware measures, and expected influence.
<code>normalized</code>	Logical. Normalize values by dividing by max. Most measures are scaled to 0-1; signed expected-influence measures can retain negative values under psychometric normalization. For closeness, this is passed directly to <code>igraph</code> .
<code>weighted</code>	Logical. Use edge weights if available. Default TRUE.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>loops</code>	Logical. If TRUE (default), keep self-loops. Set to FALSE to remove them before calculation.
<code>simplify</code>	How to combine multiple edges between the same node pair (possible only from edge-list, <code>cograph_network</code> or <code>igraph</code> input). Options: "sum" (default), "mean", "max", "min". FALSE and "none" also sum them: the network is held as a dense weight matrix, which cannot carry parallel edges.
<code>digits</code>	Integer or NULL. Round all numeric columns to this many decimal places. Default NULL (no rounding).
<code>sort_by</code>	Character or NULL. Column name to sort results by (descending order). Default NULL (original node order).
<code>cutoff</code>	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). Set to a positive value for faster computation on large networks at the cost of accuracy.

<code>invert_weights</code>	Logical or NULL. For path- and distance-based measures (for example betweenness, closeness, harmonic, eccentricity, k-reach, radiality, decay, stress, flow betweenness, and related variants), should weights be inverted so that higher weights mean shorter paths? Default NULL auto-detects: TRUE for tna objects (transition probabilities), FALSE otherwise (matching igraph/sna). Set explicitly to TRUE for strength/frequency weights (qgraph style) or FALSE for distance/cost weights.
<code>alpha</code>	Numeric. Exponent for weight transformation when <code>invert_weights = TRUE</code> . Distance is computed as $1 / \text{weight}^{\alpha}$. Default 1. Higher values increase the influence of weight differences on path lengths.
<code>damping</code>	PageRank damping factor. Default 0.85. Must be between 0 and 1.
<code>personalized</code>	Named numeric vector for personalized PageRank. Default NULL (standard PageRank). Values should sum to 1.
<code>transitivity_type</code>	Type of transitivity to calculate: "local" (default), "global", "undirected", "localundirected", "barrat" (weighted), "weighted", or "onnella". The first six dispatch to <code>igraph::transitivity()</code> ; "onnella" computes the Onnella / Holme weighted clustering coefficient on the symmetrized matrix ($wcc(x + t(x))$) and matches <code>tna::centralities(., "Clustering")</code> byte-for-byte. Auto-set to "onnella" when <code>tna_network = TRUE</code> and the user did not pass an explicit value.
<code>isolates</code>	How to handle isolate nodes in transitivity calculation: "nan" (default) returns NaN, "zero" returns 0.
<code>lambda</code>	Diffusion scaling factor for diffusion centrality. Default 1. Only used when <code>diffusion_method = "kandhway_kuri"</code> .
<code>diffusion_method</code>	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 $\text{rowSums}(P + P^2 + \dots + P^n)$ on the (optionally diagonal-zeroed) weighted matrix and matches <code>tna::centralities(., measures = "Diffusion")</code> when <code>loops = FALSE</code> . Default NULL auto-detects: "power_series" for tna objects (transition probabilities), "kandhway_kuri" otherwise.
<code>k</code>	Path length parameter for geodesic k-path centrality. Default 3.
<code>states</code>	Named numeric vector of percolation states (0-1) for percolation centrality. Each value represents how "activated" or "infected" a node is. Default NULL (all nodes get state 1, equivalent to betweenness).
<code>decay_parameter</code>	Numeric. Decay parameter for decay and generalized closeness centrality. Default 0.5. Must be between 0 and 1.
<code>dmnc_epsilon</code>	Numeric. Epsilon exponent for DMNC (Density of Maximum Neighborhood Component). Default 1.7 as recommended by Lin et al. (2008). centiserve uses 1.67 (four-community assumption). Must be between 1 and 2.
<code>membership</code>	Integer vector of community assignments (one per node) for community-aware measures: <code>participation</code> , <code>within_module_z</code> , <code>gateway</code> , <code>modularity_vitality</code> , and the Gould-Fernandez brokerage roles. Default NULL. Required when requesting these measures.

katz_alpha	Attenuation factor for Katz centrality. Must satisfy $\alpha < 1/\rho(A)$. Default 0.1 (matches centiserve and NetworkX conventions). 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, which leaves the caller's stream alone and lets the estimate vary between calls.
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" for cograph's pre-2.4.8 form.
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 auto radius uses finite positive distances, rounds to the nearest integer (ties to even), and has minimum 1; these are cograph conventions.

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. Independent of the existing lambda argument.
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_sensitive .
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_leaderrank .
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 .
linrank_aggregation	LineRank endpoint aggregation: probability (default) or weight. See centrality_linrank .
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_betweenness .
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 cograph_no_converge, which a bipartite component with unequal vertex classes always does. 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; one recovers the classical degree and zero drops the node's own degree entirely. Values outside [0, 1] are refused. 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 the source sets it. The source sweeps it and reports 2-3 as optimal. At one the ball collapses to the neighbors; at or above the diameter the score stops moving. Values below one and non-integers are refused. 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 the trust-value puts on the degree ratio rather than the similarity ratio in "trust_pagerank", the k of the source's equation (6); a single number in [0, 1], default 0.85, the value the source's section 3.3 selects from a Kendall-against-SIR sweep. One drops the similarity entirely and zero drops the degree.

<code>tpr_decay</code>	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 the source fixes it. The source's claim that C does not affect the result holds only for a homogeneous recursion and not for this one; see centrality_trust_pagerank .
<code>tpr_tol</code>	Convergence tolerance on the largest <i>relative</i> change of either trust-PageRank recursion, a single positive number, default $1e-14$. The source fixes no iteration count because it does not need one: both recursions have unique fixed points. The test is relative rather than absolute because the similarities on one graph span many orders of magnitude; see centrality_trust_pagerank .
<code>tpr_max_iter</code>	Iteration bound for both trust-PageRank recursions, a whole number of at least one, default 1000. Reaching it raises <code>cograph_no_converge</code> .
<code>rsp_beta</code>	Inverse temperature of the randomized-shortest-paths model used by "rsp_betweenness", a single finite number strictly above zero, default 0.01. The source fixes no default; 0.01 is the value <code>NetworkToolbox:rspbc()</code> recommends, and it sits near the random-walk limit, so raise it towards 1 and beyond to move the reading towards shortest paths. See centrality_rsp_betweenness .
<code>rsp_cost</code>	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. The source leaves the cost matrix free; both settings give unit cost per arc on a binary graph. See centrality_rsp_betweenness .
<code>re_indexes</code>	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 .
<code>re_negative</code>	Which of <code>re_indexes</code> are negative indexes, NULL for the source's own declarations. See centrality_relative_entropy .
<code>tna_network</code>	Logical or NULL. Umbrella switch that forces tna-style conventions across all measures. NULL (default) auto-detects from the input class — TRUE iff <code>x</code> is a tna or related sequence-network object. TRUE forces tna conventions even on raw matrices: <code>invert_weights = TRUE</code> , <code>loops = FALSE</code> , <code>diffusion_method = "power_series"</code> , <code>transitivity_type = "onnella"</code> . FALSE suppresses all tna defaults even for tna inputs, giving the cograph defaults verbatim. Precedence: any arg the user passes explicitly always wins over <code>tna_network</code> .
<code>psych_network</code>	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, preserving sign and bounding the result from -1 to 1. FALSE keeps the generic cograph normalization convention.
<code>...</code>	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)

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 - handles disconnected graphs better than closeness (supports mode: in/out/all)

diffusion Diffusion degree centrality - sum of scaled degrees of node and its neighbors (supports mode: in/out/all, lambda scaling)

leverage Leverage centrality - measures influence over neighbors based on relative degree differences (supports mode: in/out/all)

kreach Geodesic k-path centrality - count of nodes reachable within distance k (supports mode: in/out/all, k parameter)

alpha Alpha/Katz centrality - influence via paths, penalized by distance. Similar to eigenvector but includes exogenous contribution

power Bonacich power centrality - measures influence based on connections to other influential nodes

subgraph Subgraph centrality - participation in closed loops/walks, weighting shorter loops more heavily

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

load Load centrality - fraction of all shortest paths through node, similar to betweenness but weights paths by $1/\text{count}$

current_flow_closeness Information centrality - closeness based on electrical current flow (requires connected graph)

current_flow_betweenness Random walk betweenness - betweenness based on current flow rather than shortest paths (requires connected graph)

voterank VoteRank - identifies influential spreaders via iterative voting mechanism. Returns normalized rank (1 = most influential)

percolation Percolation centrality - importance for spreading processes. Uses node states (0-1) to weight paths. When all states equal, equivalent to betweenness. Useful for epidemic/information spreading analysis.

radiality Radiality centrality (`centiserve`). Sum of $(\text{diam} + 1 - d)$ normalized by $n-1$.

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

decay Decay centrality. Sum of δ^d for parameter δ .

residual_closeness Residual closeness. Sum of $1/2^d$.

dangalchev Dangalchev closeness (alias for residual closeness).

generalized_closeness Generalized closeness. Sum of α^d .

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. $1 / \text{sum of distances}$.

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. $(1/\text{degree})$ 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; 1 = distances spread evenly, 0 = all at one distance.
- local_dimension** Growth exponent of the ball around a node (slope of $\ln B_i(r)$ on $\ln r$); lower = more influential.
- local_information_dimension** Entropy-weighted local dimension over boxes up to half the node's eccentricity; higher = more influential.
- 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 = community hub, negative = bridge (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 = well connected.
- hide_information** Mean bits others need to find the node; high = hidden.
- 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 (higher = more influential); slope of the degree volume (lower = more important).
- 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 = more central.

- 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 = more central.
- 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. Note that `igraph::local_efficiency()` instead 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.
- length_scaled_betweenness** Betweenness with each separated pair weighted by $1/d(s, t)$.
- delta_betweenness** Betweenness with the pair weight $(d(s, t) - 1)^{-\delta}$ (betweenness_delta).
- 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; the Zoo's neighbor distance centrality at the defaults. 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 $M = (I - Z + EZ_{dg}) \text{diag}(1/c)$ of the influence chain $W = A / \text{rowSums}(A)$ 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`.

linerrank PageRank on the line graph, aggregated at endpoints; uses damping and `linerrank_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_leadrerank** 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

A few measures are undefined on some graphs – the community-partition measures without membership, or "relative_entropy" when one of its constituent indexes is zero at every node. Naming such a measure in `measures` or `include` raises a classed condition, because you asked for that measure. When a tier (`type = "basic"`, `"extended"` or `"all"`) supplied it, the condition becomes a `cograph_undefined_measure` warning and the column is NA, so one undefined measure does not take the rest of the tier with it.

Examples

```
# Built-in edge-list data
data(student_interactions)
centrality(student_interactions)

# Matrix input also works
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality(adj)

# Specific measures
centrality(adj, measures = c("degree", "betweenness"))

# Directed network with normalization
centrality(adj, mode = "in", normalized = TRUE)
```

```
# Sort by pagerank
centrality(adj, sort_by = "pagerank", digits = 3)

# PageRank with custom damping
centrality(adj, measures = "pagerank", damping = 0.9)

# Harmonic centrality (better for disconnected graphs)
centrality(adj, measures = "harmonic")

# Global transitivity
centrality(adj, measures = "transitivity", transitivity_type = "global")
```

centrality_access_information

Access and Hide Information

Description

Search-information centralities of Rosvall, Trusina, Minnhagen and Sneppen (2005) and Sneppen, Trusina and Rosvall (2005). A walker who knows only the shortest paths from i to j but has no map must be told which link to take at each step; the number of bits needed is

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

summed over all shortest paths, with k_i the degree of the source and $k_l - 1$ the choices left at each intermediate node (the link the walker arrived on is excluded). Then

$$A_i = \frac{1}{N} \sum_j S(i \rightarrow j), \quad H_i = \frac{1}{N} \sum_j S(j \rightarrow i),$$

with $S(i \rightarrow i) = 0$.

Usage

```
centrality_access_information(x, ...)
```

```
centrality_hide_information(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Details

Access information A_i : how many bits it costs, on average, to reach the rest of the network from i . A low value means the node reaches others with few decisions. Hubs score *high*: a walker leaving a hub has many links to choose from (on a star with five leaves the hub scores 1.93 bits, a leaf 1.33). **Hide information** H_i : how many bits it costs the rest of the network to find i . High values mark hidden, peripheral nodes; hubs score low (the star hub scores 0). The encyclopedia's prose states the star case the other way round; the formulas and the source papers give the values above.

On a directed graph every step uses the out-degree, $1/k^{out}$. On a disconnected graph the average runs over the nodes a walker can actually reach (or be reached from), so values stay finite; on a connected graph this is exactly the paper's $1/N$. Distances are hop counts; edge weights are ignored. Cost is $O(N(N + M))$ with an $N \times N$ matrix in memory.

Validated against an independent enumeration of all shortest paths and against the worked values in both papers (star and complete bipartite graphs).

Value

Named numeric vector, one value per node, 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](#) for computing multiple measures at once.

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_access_information(star5)
centrality_hide_information(star5)
```

Description

Xu and Wang's adaptive LeaderRank computes original node H-indices, then adds a ground node with H-index one, joined bidirectionally to every original node. Each augmented arc from j to i has weight $a_{ji}h_i$. Row-normalized weights define the resource transition matrix. Raw stationary scores retain total augmented mass N , following initial score one on ordinary nodes and zero on ground. The ground score is omitted without redistribution. H-indices are not recomputed after ground edges are added.

Usage

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

Arguments

<code>x</code>	Network input accepted by centrality .
<code>alr_h_mode</code>	Original H-index convention: all (default), out or in.
<code>...</code>	Additional arguments to centrality .

Details

The H-index is the largest integer h for which at least h original neighbors have degree at least h . The focal node is excluded from that neighbor list. This differs from [cograph's](#) existing closed-neighborhood [centrality_lobby](#) convention.

The paper evaluates directed and undirected networks but does not pin a directed H-index convention. `alr_h_mode` makes that choice explicit. Default "all" computes H-indices on the simple undirected skeleton, merging reciprocal arcs. "out" uses outgoing neighbors' out-degrees; "in" uses incoming neighbors' in-degrees. These directed H-index choices are explicit [cograph](#) conventions, not claims of the authors' directed-software behavior. In every case, resource flow retains the original directed arcs. Undirected edges become opposite arcs, and all H-index modes then coincide.

Input weights are ignored; the algorithm generates its own destination weights. Loops are removed and parallel arcs count once. The generic mode, inversion and cutoff arguments are ignored. Original nodes with H-index zero receive zero stationary score. If every H-index is zero, the ground transition row is undefined and all scores are NaN. This can occur on edgeless inputs or some directed inputs in in/out H-index modes. Empty input returns an empty vector. No H-index pseudocount is added.

A native ground-elimination solve obtains the unique stationary solution in $O(N^3)$ time and $O(N^2)$ memory, including periodic chains for which ordinary iteration need not converge. Optional final max normalization acts on the returned ordinary-node scores. Numerical definition agreement does not establish superior spreading predictions or author-code parity.

Value

Named numeric vector 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.

Examples

```
centrality_adaptive_leaderrank(igraph::make_ring(4))
```

centrality_alpha	<i>Alpha (Katz) Centrality</i>
------------------	--------------------------------

Description

Influence via all paths penalized by distance. Similar to eigenvector centrality but includes an exogenous contribution, making it well-defined even for directed acyclic graphs.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of alpha centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_eigenvector](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_alpha(adj)
```

centrality_authority *HITS Authority and Hub Scores*

Description

Kleinberg's HITS algorithm. centrality_authority scores nodes pointed to by good hubs. centrality_hub scores nodes that point to good authorities.

Usage

```
centrality_authority(x, ...)
```

```
centrality_hub(x, ...)
```

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
... Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of authority or hub scores.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_authority(adj)
centrality_hub(adj)
```

centrality_average_distance
 Average Distance Centrality

Description

Sum of shortest path distances divided by (n + 1). Lower values indicate more central nodes.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of average distance values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_average_distance(adj)
```

centrality_barycenter *Barycenter Centrality*

Description

Inverse of the total distance to all reachable nodes.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of barycenter centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_barycenter(adj)
```

centrality_beta_measure

BG-index or beta power measure

Description

The positive beta-measure of node i is the sum, over its successors j , of one divided by the in-degree of j . Each node with predecessors shares one unit of domination power equally among those predecessors. This is van den Brink and Gilles' BG-measure (1992, definition 2.1), subsequently called the beta-measure (2000, definition 2.1). The negative variant applies the positive measure to the reversed graph (Boldi and Vigna 2014). It sums reciprocal source out-degrees over incoming neighbors.

Usage

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

Arguments

<code>x</code>	Network input accepted by centrality .
<code>beta_direction</code>	Either "positive" (default, credits sources) or "negative" (credits destinations).
<code>...</code>	Additional arguments to centrality . <code>normalized = TRUE</code> divides scores by their maximum; all-zero scores remain zero. This differs from normalizing to unit total mass.

Details

Uses the simple unweighted graph, retaining direction. Loops and duplicate arcs are removed; weights, mode, inversion and cutoff are ignored. Undirected edges represent reciprocal arcs, so both variants coincide with the sum of reciprocal neighbor degrees. This does not implement the separately defined weighted extension of the original paper.

Nodes without successors have positive score zero; nodes without predecessors have negative score zero. Isolates score zero, and empty graphs return no scores. There is no division by a zero degree: every contributing successor has at least one predecessor. Raw positive scores sum to the number of nodes with nonzero in-degree; raw negative scores sum to the number with nonzero out-degree. In disconnected graphs this accounting applies independently to each component.

Dense matrix preparation and evaluation take $O(n^2)$ time and memory. The score is an expected number of predecessor selections, not a probability distribution or a stationary random-walk centrality.

Value

Named numeric vector 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](https://doi.org/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](https://doi.org/10.1080/15427951.2013.865686).

Examples

```
centrality_beta_measure(igraph::make_graph("Zachary"))
centrality_beta_measure(igraph::make_star(5, mode = "out"),
                        beta_direction = "negative")
```

centrality_betweenness

Betweenness Centrality

Description

Fraction of shortest paths passing through each node. Nodes with high betweenness act as bridges connecting different parts of the network.

Usage

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

Arguments

- | | |
|-----|---|
| x | Network input (matrix, igraph, network, cograph_network, tna object). |
| ... | Additional arguments passed to centrality (e.g., normalized, weighted, directed, cutoff, invert_weights). |

Value

Named numeric vector of betweenness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_load](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_betweenness(adj)
```

centrality_bottleneck *Bottleneck Centrality*

Description

Number of shortest path trees where the node appears in more than $n/4$ paths.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named integer vector of bottleneck centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_bottleneck(adj)
```

centrality_bridging *Bridging Centrality*

Description

Product of betweenness and bridging coefficient. Identifies nodes that bridge communities.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of bridging centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_localized_bridging](#) for the ego-network variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_bridging(adj)
```

centrality_bridging_capital

Bridging capital from lost information walks

Description

Implements Jackson's section 3.3 definition:

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

P contains per-contact transmission probabilities between zero and one. Rows need not sum to one: this is broadcast information flow, not a Markov chain. bridging_steps is the finite horizon T, default two, with zero giving an empty sum. Input edge weights supply P; unweighted edges use probability one. Finite nonnegative pair values v_st default to one, including diagonal entries. Named value matrices are reordered by labels.

Usage

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

Arguments

x Network input accepted by [centrality](#).

bridging_steps Nonnegative integer horizon, default two.

bridging_values Optional nonnegative n by n source-destination information-value matrix; NULL uses ones. Both dimensions may be named.

... Additional arguments to [centrality](#).

Details

The source explicitly deletes one matrix entry P_{ij} and credits its criticality to i . On undirected input, opposite entries are therefore tested separately; deleting one leaves the reverse entry present. This is not simultaneous deletion of an undirected edge or of a whole node. Walks can repeat nodes and edges. A walk using the selected entry several times contributes once to that entry's deletion loss, not once per use.

Direction and loops are retained, as allowed by the source's formal definitions. Generic loops/simplify apply first. Remaining parallel weights sum into one matrix entry and must still be at most one; removal deletes that aggregate entry. Zero weights are absent. Mode, inversion and shortest-path cutoff do not affect results. Isolates score zero, empty inputs return no scores, and all-zero values or zero horizon give zeros. No renormalization follows entry removal.

The native implementation tracks walks that have and have not used the selected entry, avoiding cancellation in matrix-power subtraction. Dense cost is $O(m T n^3)$ time and $O(n^2)$ memory, where m is the number of positive directed matrix entries. Request this costly measure explicitly. Nonrepresentable intermediate walk masses raise errors, even if a final rescaled result might exist. Raw valued-score overflow may be avoided by `normalized=TRUE`, which scales values first then divides final node scores by their maximum. This implements expected walk counts $EInf$, not the source's alternative probability-of-ever-hearing measure PI_{inf} .

Value

Named numeric vector 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.

Examples

```
centrality_bridging_capital(igraph::make_ring(4), bridging_steps = 2)
```

centrality_brokerage_coordinator

Gould-Fernandez Brokerage — Coordinator Role

Description

Coordinator brokerage (w_I): count of open directed 2-paths $A \rightarrow V \rightarrow A$ passing through node V , where all three nodes belong to V 's group. The broker mediates contact between two in-group members.

Usage

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

Arguments

x	Directed network input (matrix, igraph, cograph_network, tna object).
membership	Integer or character vector of group assignments, length equal to the number of nodes. Required.
...	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::brokerage$raw.nli[, "w_I"]`. Counts OPEN 2-paths only — those where no direct edge from *a* to *c* exists. Directed-only; returns NA with a warning on undirected input.

Value

Named integer vector of coordinator role counts.

References

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

See Also

[centrality](#), [centrality_brokerage_itinerant](#), [centrality_brokerage_representative](#), [centrality_brokerage_...](#)
[centrality_brokerage_liaison](#).

Examples

```
adj <- matrix(c(0,1,1,0, 0,0,1,1, 0,0,0,1, 1,0,0,0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_brokerage_coordinator(adj, membership = c(1, 1, 2, 2))
```

centrality_brokerage_gatekeeper
<i>Gould-Fernandez Brokerage — Gatekeeper Role</i>

Description

Gatekeeper brokerage (b_OI): count of open directed 2-paths $A \rightarrow V \rightarrow B$ where *V* and *B* are in the same group and *A* is in a different group. The broker acts as a gate letting in-group members receive contact from outside.

Usage

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


Arguments

x	Directed network input (matrix, igraph, cograph_network, tna object).
membership	Integer or character vector of group assignments, length equal to the number of nodes. Required.
...	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::brokerage$raw.nli[, "b_OI"]`. Directed-only.

Value

Named integer vector of gatekeeper role counts.

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](https://doi.org/10.2307/270949).

See Also

[centrality_brokerage_coordinator](#).

Examples

```
adj <- matrix(c(0,1,1,0, 0,0,1,1, 0,0,0,1, 1,0,0,0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_brokerage_gatekeeper(adj, membership = c(1, 1, 2, 2))
```

centrality_brokerage_itinerant

Gould-Fernandez Brokerage — Itinerant (Consultant) Role

Description

Itinerant brokerage (w_O): count of open directed 2-paths $A \rightarrow V \rightarrow A$ where the two endpoints are in the same group but the broker V is in a different group. The broker mediates within another group as an outsider.

Usage

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

Arguments

x	Directed network input (matrix, igraph, cograph_network, tna object).
membership	Integer or character vector of group assignments, length equal to the number of nodes. Required.
...	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::brokerage$raw.nli[, "w_0"]`. Directed-only.

Value

Named integer vector of itinerant role counts.

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](#).

Examples

```
adj <- matrix(c(0,1,1,0, 0,0,1,1, 0,0,0,1, 1,0,0,0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_brokerage_itinerant(adj, membership = c(1, 1, 2, 2))
```

centrality_brokerage_liaison

Gould-Fernandez Brokerage — Liaison Role

Description

Liaison brokerage (b_O): count of open directed 2-paths $A \rightarrow V \rightarrow B$ where all three nodes belong to different groups. The broker mediates between two groups to neither of which they belong.

Usage

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

Arguments

x	Directed network input (matrix, igraph, cograph_network, tna object).
membership	Integer or character vector of group assignments, length equal to the number of nodes. Required.
...	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::brokerage$raw.nli[, "b_0"]`. Directed-only.

Value

Named integer vector of liaison role counts.

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](https://doi.org/10.2307/270949).

See Also

[centrality_brokerage_coordinator](#).

Examples

```
adj <- matrix(c(0,1,1,0, 0,0,1,1, 0,0,0,1, 1,0,0,0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_brokerage_liaison(adj, membership = c(1, 1, 2, 2))
```

centrality_brokerage_representative

Gould-Fernandez Brokerage — Representative Role

Description

Representative brokerage (b_IO): count of open directed 2-paths $A \rightarrow V \rightarrow B$ where A and V are in the same group and B is in a different group. The broker represents their group outward.

Usage

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

Arguments

<code>x</code>	Directed network input (matrix, igraph, cograph_network, tna object).
<code>membership</code>	Integer or character vector of group assignments, length equal to the number of nodes. Required.
<code>...</code>	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::brokerage$raw.nli[, "b_IO"]`. Directed-only.

Value

Named integer vector of representative role counts.

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](#).

Examples

```
adj <- matrix(c(0,1,1,0, 0,0,1,1, 0,0,0,1, 1,0,0,0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_brokerage_representative(adj, membership = c(1, 1, 2, 2))
```

centrality_cda	<i>Clustering degree algorithm centrality</i>
----------------	---

Description

Wang et al.’s CDA returns the propagation-capability score $PC_i = CD_i + \sum_j (w_{ij}/w_{\max})CD_j$, where $CD_i = [\alpha d_i + (1 - \alpha)s_i]/[1 + \exp(-C_i^w)]$. Here d is degree, s is strength, and Cw is Barrat’s weighted local clustering coefficient. The maximum edge weight is taken over the whole graph, including other connected components. Inner CD scores remain raw until the final optional normalization of PC.

Usage

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

Arguments

- x Network input accepted by [centrality](#).
- cda_alpha Degree-versus-strength mixing weight between zero and one. Default 0.5 follows the source; endpoints select strength and degree respectively while retaining weighted clustering and neighbor contributions.
- ... Additional arguments to [centrality](#). With normalized = TRUE, positive final scores are divided by their maximum.

Details

Uses finite nonnegative weights on an undirected graph. Zero-weight edges are absent connections. Clustering is set to zero for nodes with fewer than two positive-weight neighbors; this convention agrees with the source's leaf example. Isolates and edgeless graphs score zero. Weights retain their original units: scaling all weights can change scores and rankings because degree and strength are combined. At alpha zero, uniform weight scaling scales scores proportionally; at alpha one, scores are invariant to that scaling. Binary inputs are independent of alpha because their degree and strength coincide.

Self-loops are removed. For weighted directed inputs, opposite arcs are added into undirected edge weights. Parallel weights are combined by `simplify` first, with any remaining parallel edges added. Without weights, the simple undirected skeleton is used. These are explicit cograph projections to the source's undirected domain. `mode` and shortest-path weight inversion do not affect CDA. Nonfinite intermediate strengths or scores raise an error, including when normalization is requested.

Value

Named numeric vector 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](https://doi.org/10.1109/ACCESS.2018.2822844).

Examples

```
centrality_cda(igraph::make_ring(5), cda_alpha = 0.5)
```

centrality_centroid	<i>Centroid Value</i>
---------------------	-----------------------

Description

Minimum difference between own and competitor's closer-node count. Measures how much a node is at the center of the graph.

Usage

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

Arguments

<code>x</code>	Network input (matrix, <code>igraph</code> , <code>network</code> , <code>cograph_network</code> , <code>tna</code> object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., <code>normalized</code> , <code>weighted</code> , <code>directed</code>).

Value

Named numeric vector of centroid values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_centroid(adj)
```

centrality_closeness *Closeness Centrality*

Description

Inverse of the average shortest path distance from a node to all others. For directed networks, `centrality_incloseness` and `centrality_outcloseness` measure incoming and outgoing closeness.

Usage

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

centrality_incloseness(x, ...)

centrality_outcloseness(x, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of closeness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_harmonic](#) for a variant that handles disconnected graphs.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_closeness(adj)
```

centrality_closeness_vitality
Closeness Vitality

Description

Drop in the Wiener index when a node is removed. Higher values indicate more critical nodes for overall connectivity.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of closeness vitality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_closeness_vitality(adj)
```

centrality_clusterrank

ClusterRank Centrality

Description

Product of clustering coefficient and sum of (neighbor degree + 1).

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of ClusterRank values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_clusterrank(adj)
```

centrality_coleman_theil

Coleman-Theil hierarchy index

Description

Measures concentration of Burt's dyadic constraints over a node's contacts. Let mutual tie strength be $z_{ij}+z_{ji}$, and p_{ij} its proportion of all mutual strength incident to i . With organizational weights fixed at one, define

$$c_{ij} = (p_{ij} + \sum_q p_{iq}p_{qj})^2, \quad r_{ij} = c_{ij} / \text{mean}_{k \in N(i)} c_{ik}.$$

The index is $\sum_{j \in N(i)} r_{ij} \log(r_{ij}) / (d_i \log(d_i))$. Contacts are distinct nodes with positive mutual strength. Investment proportions use the full supplied graph, including alters' outside ties.

Usage

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

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality .

Details

Follows Burt's STRUCTURE 4.2 manual (pages 181-183): isolates score zero and nodes with one contact score one. The general formula is undefined in these two cases; these are the author's explicit conventions. JUNG's documented implementation instead returns NaN for isolates. Values range from zero for equal constraints to one for complete concentration. Input organizational/oligopoly multipliers from STRUCTURE are not implemented; they are fixed at one, as in the Zoo's formula.

Finite nonnegative weights are supported. Zero-weight ties are absent, loops are removed, and remaining parallel edges sum after generic simplification. Directed ties are combined by summing both directions; `weighted=FALSE` assigns unit weight to each retained edge before combining them, so reciprocity can affect mutual investment. Generic mode, shortest-path inversion and cutoff do not affect the result. Empty input returns no scores. Components are independent before global normalization.

The default output is already the unit-interval hierarchy index. `normalized=TRUE` additionally divides by the largest node score; an all-zero vector remains zero. Dense native arithmetic costs $O(n^3)$ time and $O(n^2)$ memory. Global weight scaling precedes mutual sums. Unrepresentable positive weight or investment ranges raise an error; tiny squared constraints may underflow and use the zero-log-zero limit. Relative deviations of local constraints within 16 machine epsilons are treated as uniform; a series stabilizes the entropy near uniformity.

Value

Named numeric vector in input node order.

References

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

Examples

```
centrality_coleman_theil(igraph::make_star(5, mode = "undirected"))
```

centrality_communicability

Communicability Centrality

Description

Total communicability: row sums of the matrix exponential of the adjacency matrix. Measures a node's ability to broadcast information through all paths.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#).

Value

Named numeric vector of communicability values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_subgraph](#) for the diagonal-only variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_communicability(adj)
```

centrality_communicability_betweenness

Communicability Betweenness Centrality

Description

Fraction of total communicability that passes through each node.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#).

Value

Named numeric vector of communicability betweenness values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_communicability_betweenness(adj)
```

centrality_community_based

Community-Based Centrality, Comm Centrality and Community-Based Mediator

Description

Three community-aware measures that need a partition (membership).

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 (matrix, igraph, network, cograph_network, tna object).
`membership` Community labels, one per node. Required; without it the function warns and returns NA.
`mode` For directed networks: "all" (default), "out", or "in".

... Additional arguments passed to [centrality](#).
 comm_r Scale R of Comm centrality: "max_intra" (default) or a single positive number. Anything else raises a `cograph_bad_parameter` error.

Details

community_based (**Zhao, Wang, Zhang & Zhu 2015**) $CbC(i) = \sum_w d_{iw} S_w / N$: every link of i counts the size S_w of the community it lands in. No parameters. Reproduces Table 1 of the paper and Table 1 of Tulu et al. (2018).

comm_centrality (**Gupta, Singh & Cherifi 2016**)

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

where k_i^{in}, k_i^{out} are the intra- and inter-community degrees, μ_C the mean inter-link fraction in i 's community, and R a scale. The default `comm_r` = "max_intra" is the paper's recommended $R = \max_{j \in C} k_j^{in}$ per community; a number applies one global R . The equation uses $1 + \mu_C$ although the paper's prose says μ_C ; the equation is implemented. A community without intra (inter) links contributes 0 through that term.

community_mediator (**Tulu, Hou & Younas 2018**) $CbM(i) = H_i d_i / \sum_j d_j$, with H_i the base-2 Shannon entropy of i 's link distribution over the communities. Nodes linked to one community only score 0. Base 2 is what reproduces the paper's Table 1.

Higher = more central in all three. Under `mode` = "out" or "in" only out- or in-links count; edge weights are ignored.

Value

Named numeric vector, one value per node.

Conditions

Raises an error of class `cograph_bad_membership` when membership is not one non-missing label per node.

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](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_community_based(adj, membership = c(1, 1, 1, 2, 2, 2))
centrality_comm_centrality(adj, membership = c(1, 1, 1, 2, 2, 2))
centrality_community_mediator(adj, membership = c(1, 1, 1, 2, 2, 2))
```

centrality_community_hub_bridge

Community Hub-Bridge Centrality

Description

Ghalmane, El Hassouni and Cherifi's (2019) score for nodes that are both 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 number of nodes in i 's own community, 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 is linked to (eqs. 2 to 4 of the paper). Higher values mark nodes whose removal both fragments their community and cuts links between communities. A normalized variant with the same name exists in later work by the same group; this is the original raw form.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
membership	Community labels, one per node. Required; without it the function warns and returns NA. Obtain one from detect_communities .
mode	For directed networks: "all" (default), "out", or "in".
...	Additional arguments passed to centrality .

Details

Under mode = "out" or "in" only out- or in-links count; the default ignores direction. Edge weights are ignored.

Value

Named numeric vector, one value per node.

Conditions

Raises an error of class `cograph_bad_membership` when membership is not one non-missing label per node.

References

Ghalmame, 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_modularity_vitality](#), [centrality_participation](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_community_hub_bridge(adj, membership = c(1, 1, 1, 2, 2, 2))
```

centrality_constraint *Burt's Constraint*

Description

Network constraint measuring the extent to which a node's connections are redundant. Low constraint indicates access to structural holes (brokerage opportunities).

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of constraint values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_constraint(adj)
```

centrality_controlrank

ControlRank centrality

Description

Zhou, Yu and Lu's ControlRank is the smallest eigenvalue after deleting a node's row and column from the symmetric part of the graph Laplacian. With $L = D - A$, this is $CR_i = \lambda_{\min}(((L + L^T)/2)_{-i, -i})$. D retains the original graph's degrees: the Laplacian is not recomputed on the vertex-deleted graph. Larger values receive higher rank.

Usage

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

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality . <code>normalized = TRUE</code> divides by the maximum if it is positive; otherwise raw scores are retained. This package normalization is optional and is not part of the published definition.

Details

Uses finite nonnegative interaction weights. For directed input, A_{ij} denotes an arc from i to j and D contains outgoing strengths. This fixes the row-Laplacian orientation explicitly; transpose the input to use incoming strengths. Symmetrizing L preserves its diagonal, so this is different from constructing a Laplacian of the undirected projection. Directed scores can be negative and are not clipped. For matrix inputs with very small weights, supply `directed = TRUE` explicitly (or use a directed igraph object): the shared input parser's approximate symmetry detection can otherwise infer an undirected graph.

Loops are removed and zero weights are absent connections. Parallel weights follow the generic simplify rule; remaining parallel edges sum. With `weighted = FALSE`, each remaining edge contributes one. Mode, weight inversion for shortest paths and cutoff are ignored.

Connected undirected graphs with at least two nodes have positive scores. Disconnected undirected graphs score zero for every node because at least one component remains ungrounded. Empty graphs return no scores; singletons return zero as an explicit extension of the undefined empty minor. The source excludes isolates; the matrix formula here also applies to disconnected directed graphs, whose scores may remain negative.

This implements the spectral index, not a controller simulation, a finite-feedback convergence rate, or an optimization over controller sets. In particular, no general directed stability guarantee is inferred from these scores. The paper's multi-node selection problem is separate.

Dense eigensolves take $O(n^4)$ time and $O(n^2)$ memory; this measure is marked costly and excluded from the default all tier. Disconnected blocks are solved separately, preserving isolated zeros before normalization. Global scaling avoids intermediate overflow. Unrepresentable weight ranges and unresolved positive spectra raise errors. Signed directed scores near zero can retain floating-point roundoff; very small raw scores can underflow. Uniform weight scaling multiplies raw scores by the same factor.

Value

Named numeric vector 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](https://doi.org/10.1109/TCSII.2018.2845940).

Examples

```
centrality_controlrank(igraph::make_ring(5))
centrality_controlrank(igraph::make_star(6, mode = "undirected"))
```

centrality_coreness	<i>K-Core Decomposition (Coreness)</i>
---------------------	--

Description

Assigns each node to its maximum k-core. A k-core is a maximal subgraph where every node has at least k connections within the subgraph.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of coreness values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_coreness(adj)
```

centrality_cross_clique

Cross-Clique Connectivity

Description

Count of all cliques (not just maximal) containing each node. Measures embeddedness in dense substructures.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named integer vector of cross-clique counts.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_cross_clique(adj)
```

centrality_current_flow_betweenness

Current Flow Betweenness Centrality

Description

Betweenness based on electrical current flow rather than shortest paths. Uses the Laplacian pseudoinverse. Requires a connected graph.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of current flow betweenness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_betweenness](#) for the shortest-path variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_current_flow_betweenness(adj)
```

centrality_current_flow_closeness

Current Flow Closeness Centrality

Description

Information centrality based on electrical current flow through the network. Uses the pseudoinverse of the Laplacian matrix. Requires a connected graph.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of current flow closeness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_closeness](#) for the shortest-path variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_current_flow_closeness(adj)
```

centrality_dangalchev *Dangalchev Closeness Centrality*

Description

Alias for residual closeness centrality: sum of $1/2^d$.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`mode` For directed networks: "all" (default), "in", or "out".
`...` Additional arguments passed to [centrality](#) (e.g., normalized, weighted, directed).

Value

Named numeric vector of Dangalchev closeness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_residual_closeness](#) (equivalent).

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_dangalchev(adj)
```

centrality_decay	<i>Decay Centrality</i>
------------------	-------------------------

Description

Sum of δ^d over all nodes, where d is the shortest path distance. Nodes near many others get higher scores. The `decay_parameter` controls the distance penalty.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>decay_parameter</code>	Numeric between 0 and 1. Default 0.5.
<code>...</code>	Additional arguments passed to centrality .

Value

Named numeric vector of decay centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_decay(adj, decay_parameter = 0.5)
```

centrality_degree	<i>Degree Centrality</i>
-------------------	--------------------------

Description

Number of edges connected to each node. For directed networks, `centrality_indegree` counts incoming edges and `centrality_outdegree` counts outgoing edges.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of degree values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_strength](#) for the weighted version.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)  
rownames(adj) <- colnames(adj) <- c("A", "B", "C")  
centrality_degree(adj)
```

centrality_degree_discount

DegreeDiscountIC and SingleDiscount Rankings

Description

Chen, Wang and Yang’s (2009) degree-discount heuristics for choosing spreaders under the independent-cascade model. Nodes are selected one at a time by the largest *discounted* degree; after each selection every unselected neighbor v of the new seed counts one more selected neighbor, t_v , and its discounted degree becomes

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

for DegreeDiscountIC (Algorithm 4 of the paper, with propagation probability p , default 0.01), or simply $d_v - t_v$ for SingleDiscount, where each neighbor of a new seed discounts its degree by one. Every node is placed, so the result is a full ranking, returned as a score: the first node selected scores 1, the last $1/n$.

Usage

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

```
centrality_single_discount(x, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>discount_p</code>	Propagation probability p for DegreeDiscountIC. Default 0.01.
<code>...</code>	Additional arguments passed to centrality .

Details

Ties are broken by node order, which the paper does not specify. Direction, edge weights and self-loops are ignored, as in the paper’s setting. Validated against an independent implementation of the algorithm and against the reference code of the influence-maximization literature on the karate club graph.

Value

Named numeric vector in $(0, 1]$, one score per node.

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](#) for the voting-based alternative.

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_degree_discount(adj)
centrality_single_discount(adj)
```

centrality_diffusion *Diffusion Centrality*

Description

Sum of scaled degrees of a node and its neighbors, measuring the node's potential for spreading information through the network.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out". Only used when <code>diffusion_method = "kandhway_kuri"</code> (the default for non-tna inputs); ignored under "power_series", which always treats the matrix as the row transition operator.
<code>lambda</code>	Scaling factor for neighbor contributions. Default 1. Only used when <code>diffusion_method = "kandhway_kuri"</code> .
<code>...</code>	Additional arguments passed to centrality (e.g., <code>diffusion_method</code> , <code>loops</code> , <code>weighted</code> , <code>directed</code>).

Details

Two methods are supported. "kandhway_kuri" (Kandhway & Kuri, 2014) computes the 1-hop binary-degree neighborhood sum and is the default for raw matrices, igraph objects, and other non-tna inputs. "power_series" computes $\text{rowSums}(P + P^2 + \dots + P^n)$ on the weighted matrix (with $\text{diag}(P) := 0$ when `loops = FALSE`) and matches `tna::centralities(., measures = "Diffusion")` byte-for-byte. For tna inputs, the default switches to "power_series" to match user expectation; pass `diffusion_method = "kandhway_kuri"` to force the binary-degree formula.

Value

Named numeric vector of diffusion centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_diffusion(adj)
```

centrality_diffusion_centrality

Finite-horizon diffusion centrality

Description

Banerjee et al.'s diffusion centrality is $DC(A; q, T) = \sum_{t=1}^T (qA)^t \mathbf{1}$. It sums weighted walks starting at each node, allowing revisits and returns to the source. Directed edges carry information from their source to their target: the result uses row sums, regardless of mode. Transpose the input adjacency matrix to measure incoming walks.

Usage

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

Arguments

x	Network input accepted by centrality .
diffusion_q	Finite multiplier between 0 and 1, default 1.
diffusion_steps	Nonnegative integer horizon, default 3. Must be no larger than <code>.Machine\$integer.max</code> .
...	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive scores are divided by their maximum.

Details

A is the adjacency matrix with the original edge weights when `weighted = TRUE`, or unit edge weights otherwise. Self-loops follow loops; an undirected self-loop contributes its weight once on the diagonal. The `simplify` argument combines parallel edges first; any remaining parallel edges contribute additively to A. Weight inversion for shortest paths does not affect this measure.

When every entry of qA is between zero and one, scores have the paper's interpretation as expected total hearings of information. Larger weights are accepted as a mathematical weighted-walk extension of that polynomial, without a probability interpretation. Scores count repeated hearings, not distinct recipients. They need not be bounded by the number of nodes.

Default $q = 1$ and $T = 3$ are explicit cograph choices, not estimates of a diffusion process or the parameters used by the Zoo. $T = 0$ returns zero; $T = 1$ gives q times outgoing strength (degree for

a binary graph). A finite horizon requires no spectral convergence condition. Numerical overflow raises an error, including when normalization is requested.

This is distinct from [centrality_diffusion](#): its default is diffusion degree, and its TNA variant fixes $q = 1$ and $T = n$. The existing `lambda` and `diffusion_method` arguments do not affect this measure. Computation uses T matrix-vector products.

Value

Named numeric vector 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](#).

Examples

```
g <- igraph::make_graph(c(1, 2, 2, 3), directed = TRUE)
centrality_diffusion_centrality(g, diffusion_q = 0.5, diffusion_steps = 2)
```

centrality_dil	<i>Degree and Importance of Lines</i>
----------------	---------------------------------------

Description

Liu, Xiong, Shi, Shi and Wang rank a node by its degree plus the share it can claim of the importance of the lines that touch it. A line matters when its two endpoints reach far beyond it and when no triangle offers a way round it, so the importance of the line e_{mn} is $I_{e_{mn}} = U/\lambda$ with $U = (k_m - p - 1)(k_n - p - 1)$ and $\lambda = p/2 + 1$, where p is the number of triangles one of whose edges is e_{mn} . That importance is then split between the endpoints in proportion to their own degrees, $W_{v_i v_j} = I_{e_{ij}}(k_i - 1)/(k_i + k_j - 2)$, and the score is $L_{v_i} = k_i + \sum_{v_j \in \Gamma_i} W_{v_i v_j}$ over the open neighborhood Γ_i . The measure is strictly two-hop local: only the degrees of a node, of its neighbors and the triangles on its incident lines enter, so it costs $O(n\langle k \rangle^2)$ and its raw scores are component-local.

Usage

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

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality .

Details

λ is $p/2 + 1$, and reading it from a text layer gets it wrong. The stacked fraction extracts from the published PDF as $\lambda = 2p + 1$, in Liu et al.'s original as much as in the Almasi and Hu (2019) reproduction of it. The page image shows p over 2; so does the paper's own worked example, in printed prose, on page 210: for the seven-line network of its Fig. 1(b) it writes $p = 1$, $U = 4$, " $\lambda = 1/2 + 1 = 1.5$ " and $I_{e_{45}} = 4/1.5 \approx 2.6667$. The wrong reading returns $4/3$ there. cograph reproduces $8/3$.

U is never negative, so a score never falls below the node's degree. For a line (i, j) , j belongs to $N(i)$ but to neither $N(j)$ nor the intersection, so $p = |N(i) \cap N(j)| \leq k_i - 1$ and both factors of U are at least zero. Since $\lambda \geq 1$, every I and every W is at least zero and $L_{v_i} \geq k_i$. Equality is common rather than exceptional: every line of a complete graph, of a star, or of any network whose lines all touch a degree-one node has $U = 0$, so K_n scores $n - 1$ at every node and a star scores its degree at every node.

The importance of a line is conserved when it is split. The two shares $(k_i - 1)/(k_i + k_j - 2)$ and $(k_j - 1)/(k_i + k_j - 2)$ sum to one, so $\sum_i (L_{v_i} - k_i) = \sum_e I_e$: the network's total excess over degree is exactly the total importance of its lines. That identity is asserted over the package's whole verification collection.

An isolated K_2 is the one undefined split, and it is resolved rather than refused. The denominator $k_i + k_j - 2$ vanishes only when $k_i = k_j = 1$, since both endpoints of a line have degree at least one – that is a two-node component – and there $p = 0$ and $U = (1 - 0 - 1)(1 - 0 - 1) = 0$, so the importance being divided is exactly zero while the split of it is $0/0$. Because W is a *share* of I , and the two shares sum to one wherever they are defined, every admissible split of an exactly zero importance gives an exactly zero contribution: the answer does not depend on resolving the indeterminacy. cograph therefore writes the share as zero, taking the test before the division so that no $0/0$ is ever evaluated, and both nodes of a K_2 score 1. **The source says nothing about this case;** the choice is cograph's, and it follows the precedent of [centrality_lhc](#), whose $0/0$ on a triangle-free graph is likewise written as zero because the denominator vanishes exactly where every numerator does. It deliberately does not follow [centrality_iec](#), which returns NA on reducible input: there the closed form returns a finite number in place of an infinite one, so a value would be wrong, where here every candidate value is the same value.

Direction and weights are dropped, because the authors exclude them. Page 210 opens the derivation with "we assume that a network $G = (V, E)$ is an undirected and unweighted network", and every quantity in the three equations is a count: a degree, a triangle census, a difference of integers. A directed, weighted or multigraph input is therefore projected onto its simple undirected skeleton – arcs symmetrized, weights and parallel edges collapsed to a single line, loops dropped – rather than refused, which is the convention every other undirected-domain measure in [centrality](#) already follows, and the projection is silent rather than warned for the same reason. There is no in/out/all reading to choose between, so the measure sits in the no-mode family and cutoff and invert_weights are ignored as well. The source states no normalization, so normalized = TRUE max-scales the finished vector as elsewhere in [centrality](#).

Isolates, singletons and disconnected input need no special rule. An isolate has degree zero and an empty sum, so it scores zero; the single node of a one-node graph and every node of an edgeless graph score zero for the same reason, and an empty graph returns no scores. Because nothing in equations (1)-(3) reaches past a node's second neighbors, the raw scores are component-local: attaching a disjoint component leaves every existing score unchanged.

The source prints three numerical fixtures and all three are reproduced. Fig. 1 on page 210

prints $I_{e_{45}} = 9$ at $p = 0$ and $8/3$ at $p = 1$; Fig. 2 on page 211 prints $L_{v_2} = 26/9$ and $L_{v_5} = 52/15$ on a 27-node tree; and Table 3 on page 217 prints a DIL value for every one of the 21 nodes of the ARPA network, whose topology is Fig. 6 on the same page. All 21 printed values are reproduced, and the edge list read off the figure is corroborated independently by the paper's own degree column. See the batch 50 published audit in the package's verification directory.

Value

Named numeric vector in input node order, one score per node, each at least the node's degree in the simple undirected skeleton.

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](https://doi.org/10.1016/j.physa.2016.02.049).

See Also

[centrality_lhc](#) and [centrality_hcc](#) for other degree-and-triangle hybrids, [centrality_bridging](#) for another measure that scores a node by the lines it carries, and [list_centralities](#) for the catalogue.

Examples

```
# Every line of a complete graph is shortcut by n - 2 triangles, so U is
# zero throughout and the score is the degree.
centrality_dil(igraph::make_full_graph(5))

# A triangle-free k-regular graph scores k + k(k-1)^2/2 at every node:
# 3 for a ring and 9 for the Petersen graph.
centrality_dil(igraph::make_ring(6))

# The path 1-2-3-4-5 scores 1, 2.5, 3, 2.5, 1: a line to a leaf carries
# no importance, and the two interior lines carry one each, split evenly.
centrality_dil(igraph::make_graph(c(1, 2, 2, 3, 3, 4, 4, 5),
                                directed = FALSE))

# A triangle on two degree-three nodes is the case that needs
# lambda = p/2 + 1: I = 1 / 1.5 = 2/3, split evenly, so the two hubs
# score 3 + 1/3. Reading lambda as 2p + 1 would give 3 + 1/6.
centrality_dil(igraph::make_graph(c(1, 2, 1, 3, 2, 3, 1, 4, 2, 5),
                                directed = FALSE))
```

centrality_distance_entropy
Distance Entropy

Description

Shannon entropy of the distribution of hop distances from a node to every node it can reach (Stella & De Domenico 2018), normalized so that a uniform spread over the node's distance range scores 1:

$$h(i) = -\frac{1}{\log(M_i - m_i + 1)} \sum_{k=m_i}^{M_i} p_k^{(i)} \log p_k^{(i)}, \quad p_k^{(i)} = n_k^{(i)} / R_i,$$

where $n_k^{(i)}$ is the number of nodes at distance k from i , R_i the number of reachable nodes, and m_i, M_i the minimum and maximum distance. High values mark nodes whose reach is spread evenly across many network layers; a node whose reachable nodes all sit at one distance scores 0. Closeness summarizes the mean of the same distribution; distance entropy summarizes its spread.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
...	Additional arguments passed to centrality .

Details

Distances are hop counts (edge weights are ignored). The original paper normalizes by $\log(M_i - m_i)$, which is undefined when only two distinct distances occur; $\log(M_i - m_i + 1)$ is used here so the index is bounded by 1 for a uniform distribution.

Value

Named numeric vector, one value per node, in $[0, 1]$. NaN for a node that reaches no other node.

References

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

See Also

[centrality](#) for computing multiple measures at once, [centrality_local_dimension](#) for the growth-rate view of the same distance profile.

Examples

```
path4 <- matrix(c(0,1,0,0, 1,0,1,0, 0,1,0,1, 0,0,1,0), 4, 4)
rownames(path4) <- colnames(path4) <- c("A", "B", "C", "D")
centrality_distance_entropy(path4)
```

centrality_diversity	<i>Diversity Centrality</i>
----------------------	-----------------------------

Description

Shannon entropy of the edge weight distribution per node. Measures how evenly a node distributes its connections.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of diversity centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
mat <- matrix(c(0, .5, .3, .5, 0, .8, .3, .8, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
centrality_diversity(mat)
```

centrality_dkgm	<i>DK-based gravity model</i>
-----------------	-------------------------------

Description

The DK-based gravity model (DKGM) puts the degree k-shell index of both ends into a truncated squared-distance gravity sum: $DKGM_i = \sum_{j \neq i, d(i,j) \leq R} DK(i)DK(j)/d(i,j)^2$. The mass $DK(i) = k(i) + k_s^*(i)$ adds the degree to an improved shell index $k_s^*(i) = k_s(i) + p(i) / (\max_k q(k) + 1)$, where $p(i)$ is the removal stage inside the node's shell and $q(k)$ the number of stages the k-level needed. The stage breaks the ties that degree and k-shell leave behind: two nodes of the same shell are separated by how late the peeling reached them.

Usage

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

Arguments

x	Network input accepted by centrality .
dkgm_radius	Nonnegative hop-distance cutoff, default two, the value used for the paper's Table 5 and one of the two the paper recommends in general. NULL or infinity includes every reachable partner. Fractional cutoffs include exactly the integer hop distances not exceeding them; values below one give zero. "auto" applies the paper's own equation 4, $R^* \approx \langle d \rangle / 2$, with cograph conventions: half the mean finite positive hop distance, rounded to the nearest integer with ties to even, minimum one. Those conventions and the treatment of disconnected graphs are cograph's, not the paper's.
...	Additional arguments to centrality .

Details

Stages restart at one inside every shell, but the denominator $\max_k q(k) + 1$ is a single global maximum. A node's mass therefore depends on the whole graph: adding a disconnected component that peels in more stages lengthens that denominator and changes every raw score. This is a property of the published definition, not a cograph choice, and it distinguishes DKGM from [centrality_mixed_gravity](#).

The paper's Algorithm 1 says "Find all nodes in G with degree k" while its stage loop ends "until All remaining nodes in G have degree > k" and its Methods define k-shell by removing "nodes whose degree $k \leq 1$... Until there are no nodes in the network with degree $k \leq 1$ ". Strict equality cannot terminate on a three-node path, so cograph follows the at-most reading, which is the only one consistent with the printed termination condition and which reproduces the paper's Tables 2 to 5. Removal inside a stage is simultaneous, matching the printed two-stage two-shell. Because the level starts at one, an isolate falls in the one-shell rather than the zero-shell [centrality](#)(measures = "coreness") reports; isolates carry no edges, so no other node's shell, stage or score is affected.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once and loops are removed. This projection is a cograph convention

outside that domain. Edge weights, mode, cutoff, gravity_mass and path-weight inversion are ignored. Unreachable partners contribute nothing. Isolates and singleton graphs score zero; empty graphs return no scores. Optional maximum normalization applies to the complete result over all nodes. Dense all-pairs distances cost $O(n^3)$ time and $O(n^2)$ memory.

Numerical verification establishes agreement with the published equations and the printed nine-node example, not parity with author software, which was not located, nor any claim about spreading performance.

Value

Named numeric vector 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](https://doi.org/10.1038/s41598021012181).

See Also

[centrality_mcgm](#) and [centrality_mixed_gravity](#) for the other gravity masses.

Examples

```
centrality_dkgm(igraph::make_ring(6))
centrality_dkgm(igraph::make_star(6), dkgm_radius = 1)
```

centrality_dmnc	<i>Density of Maximum Neighborhood Component (DMNC)</i>
-----------------	---

Description

Edges divided by nodes raised to dmnc_epsilon, both taken from the largest connected component of the subgraph induced on a node's neighbors (the focal node excluded).

Usage

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

Arguments

- x Network input (matrix, igraph, network, cograph_network, tna object).
- mode For directed networks: "all" (default), "in", or "out".
- dmnc_epsilon Numeric. Epsilon exponent for DMNC. Default 1.7 as recommended by Lin et al. (2008). centiserve uses 1.67 (four-community assumption). Must be between 1 and 2.
- ... Additional arguments passed to [centrality](#) (e.g., normalized, weighted, directed).

Value

Named numeric vector of DMNC values.

Divergence from centiserve

`centiserve::dmnc()` returns different values, and not only because of its different epsilon default. Its edge count is taken with `induced.subgraph(graph, which(c$membership %in% ...))`, where the membership vector indexes the neighborhood subgraph but is used to subset the original graph. The two index spaces are not the same, so the edges counted are those of an unrelated vertex set. On the Zachary karate club the two disagree on 14 of 34 nodes at a matched epsilon, and reproducing that indexing exactly reproduces `centiserve`'s output. `cograph` counts the edges of the component it actually found.

See Also

[centrality](#) for computing multiple measures at once, [centrality_mnc](#) for the size-only variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_dmnc(adj)
```

`centrality_dynamical_importance`

Dynamical importance by exact vertex deletion

Description

Restrepo, Ott & Hunt's node dynamical importance is the relative drop in adjacency spectral radius on removing that node: $I_i = (\rho(A) - \rho(A_{-i}))/\rho(A)$ (equation 2). This function recomputes the spectral radius after every deletion. The paper's left/right eigenvector product (equation 5) is an approximation and can differ substantially on small networks; it is not used here.

Usage

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

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality . The default <code>normalized = FALSE</code> preserves the published relative loss; <code>TRUE</code> additionally divides positive scores by their maximum.

Details

Supports directed or undirected nonnegative weighted networks. Self-loops are always removed, as in the paper's zero-diagonal definition. Edge weights, `weighted` and `simplify` follow the same adjacency conventions as `centrality_diffusion_centrality`. The measure is invariant to reversing all arcs and ignores mode and path-weight inversion. Disconnected graphs use the spectral radius of the whole graph.

When the original spectral radius is zero (including any directed acyclic graph), the ratio is undefined and all vertices receive NaN. Isolates in a graph with positive spectral radius receive zero. The empty graph returns an empty vector. Strong components are evaluated separately so acyclic parts contribute exactly zero, avoiding numerical eigenvalues of nilpotent blocks. Roundoff in the final ratio is clipped to zero or one.

Repeated eigendecomposition is costly. Select this measure explicitly or use `include = "dynamical_importance"`; it is held back from the default type = "all" tier.

Value

Named numeric vector 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](https://doi.org/10.1103/PhysRevLett.97.094102).

Examples

```
centrality_dynamical_importance(igraph::make_full_graph(4))
```

centrality_dynamics_sensitive

Dynamics-sensitive centrality

Description

Liu et al.'s finite-time dynamics-sensitive (DS) centrality is $S(T) = \sum_{r=0}^{T-1} \beta A [\beta A + (1 - \mu)I]^r \mathbf{1}$, where beta is the spreading rate and mu the recovery rate (equation 5 in the preprint). This is the full recovery-parameter family. For mu=1, it reduces to $\sum_{t=1}^T (\beta A)^t \mathbf{1}$ (equation 7), also the form listed in the Centrality Zoo. For mu=0 it gives the paper's susceptible-infected case.

Usage

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

Arguments

<code>x</code>	Network input accepted by centrality .
<code>ds_beta</code>	Finite spreading rate between 0 and 1, default 0.1.
<code>ds_mu</code>	Finite recovery rate between 0 and 1, default 1.
<code>ds_steps</code>	Nonnegative integer horizon, default 5. Must not exceed <code>.Machine\$integer.max</code> .
<code>...</code>	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive scores are divided by their maximum.

Details

Uses the simple undirected unweighted skeleton, as in the source: either direction creates an edge, parallel edges count once and loops are removed. The projection of other inputs is an explicit co-graph convention. `mode`, edge weights and shortest-path weight inversion do not affect this measure. Isolates score zero. `T=0` or `beta=0` returns zero; `T=1` gives `beta` times degree. The initial seed itself is not added to the score.

This linearized cumulative spreading score allows repeated walks and can exceed the number of nodes. It is not a bounded infection probability or an exact simulation of the nonlinear SIR/SI process. Defaults `beta=0.1`, `mu=1` and `T=5` select a parameter setting studied in the paper; they are not fitted to the input network. Any finite horizon is supported without a spectral convergence condition, subject to numerical precision. Overflow raises an error, even if normalization is requested.

Value

Named numeric vector 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](https://doi.org/10.1038/srep21380).

See Also

[centrality_diffusion_centrality](#).

Examples

```
g <- igraph::make_ring(5)
centrality_dynamics_sensitive(g, ds_beta = 0.1, ds_mu = 1, ds_steps = 5)
centrality_dynamics_sensitive(g, ds_mu = 0)
```

centrality_eccentricity
<i>Eccentricity</i>

Description

Maximum shortest path distance from a node to any other node. For directed networks, `centrality_ineccentricity` and `centrality_outeccentricity` use incoming and outgoing paths.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of eccentricity values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)  
rownames(adj) <- colnames(adj) <- c("A", "B", "C")  
centrality_eccentricity(adj)
```

centrality_effective_size
Effective Size (Burt's)

Description

Network effective size: degree minus redundancy. Measures non-redundant contacts in ego network.

Usage

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

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
 ... Additional arguments passed to [centrality](#).

Value

Named numeric vector of effective size values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_constraint](#) for a related structural holes measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_effective_size(adj)
```

centrality_ehcc *Extended hybrid characteristic centrality*

Description

The extended hybrid characteristic centrality of Liu and Zheng is the closed-neighborhood sum of [centrality_hcc](#): $EHCC(u) = HCC(u) + \sum_{v \in \phi(u)} HCC(v)$, the focal node counted once and each neighbor of the open 1-order neighborhood once. It rewards a node whose neighbors are themselves high in both the extended degree and the E-shell hierarchy, which a node can be without being high itself.

Usage

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

Arguments

x Network input accepted by [centrality](#).
 ... Additional arguments to [centrality](#), including hcc_delta.

Details

Everything recorded on [centrality_hcc](#) carries over unchanged: the source's arg max/arg min typo in step 3 of the E-shell procedure, the original-graph reading of k^{ex} and k_{max}^{ex} against the residual-graph peel, the global and therefore not component-local normalizers, the hcc_delta domain $[0, 1]$, the 0/0 of an edgeless graph written as zero, the simple undirected unweighted skeleton, and the ignored weights, mode, cutoff and inversion. Because HCC lies in $[0, 2]$, EHCC lies in $[0, 2(1 + k_{max})]$, and an isolate scores exactly its own HCC.

Value

Named numeric vector 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](https://doi.org/10.1038/s41598023-303085).

See Also

[centrality_hcc](#) for the summand and [list_centralities](#) for the catalogue.

Examples

```
# On a regular graph every node scores 2, so EHCC is 2 (1 + k).
centrality_ehcc(igraph::make_ring(6))

# The star's center collects every leaf's score as well as its own.
centrality_ehcc(igraph::make_star(6, mode = "undirected"))
```

centrality_eigenvector

Eigenvector Centrality

Description

Influence-based centrality where a node's score depends on the scores of its neighbors. Nodes connected to other high-scoring nodes get higher scores.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of eigenvector centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_pagerank](#) for a random walk variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_eigenvector(adj)
```

centrality_entropy *Entropy Centrality*

Description

Graph-theoretic entropy based on shortest path distribution in the residual graph after removing the node.

Usage

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

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`mode` For directed networks: "all" (default), "in", or "out".
`...` Additional arguments passed to [centrality](#) (e.g., normalized, weighted, directed).

Value

Named numeric vector of entropy centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_entropy_variation(adj)
```

```
centrality_entropy_variation
      Entropy Variation
```

Description

Ai's (2017) vitality measure: the change in the Shannon entropy of a node-level distribution when a node and its links are removed,

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

with f the degree ("entropy_variation_degree", in-, out- or total degree by mode) or the betweenness ("entropy_variation_betweenness"). Natural logarithm, as in the author's code. The difference is signed: a positive value means the remaining network is less even without the node, a negative value that removing it evens the distribution out. Higher = more important.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>of</code>	Which distribution: "degree" (default) or "betweenness".
<code>mode</code>	For the degree variant on directed networks: "all" (default, in + out), "out", or "in".
<code>...</code>	Additional arguments passed to centrality .

Details

The degree variant is computed in closed form. The betweenness variant recomputes betweenness once per node and costs $O(n \cdot nm)$; it ignores edge weights. Self-loops are counted as igraph counts them. When a deletion leaves every f at zero (for instance betweenness on a clique) that entropy is taken as 0.

Validated against the author's own R code path (iCalEnV()) from the paper's repository) to 10^{-15} and against the quantiles of Table 2 of the paper on its 4234-node Snake Idioms network.

Value

Named numeric vector, one value per node, in nats.

References

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

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_entropy_variation(star5)
centrality_entropy_variation(star5, of = "betweenness")
```

centrality_exogenous *Exogenous centrality*

Description

Measures a node's contribution to the base centrality of all other nodes, following Everett and Borgatti (2010): $E(i) = \sum_{j \neq i} [C_G(j) - C_{G-i}(j)]$. The focal node's own base score is excluded. Three base measures are supported, each calculated without normalization before deletion:

reverse_closeness Default. For a graph H with m remaining nodes, $C_H(j) = \sum_{k \neq j} \max(N - d_H(j, k), 0)$, where N is the ORIGINAL input size, including isolates. Unreachable pairs contribute zero. This implements the fixed-size adjustment in section 3.3; it is distinct from ordinary reciprocal-farness closeness.

betweenness Raw shortest-path betweenness with endpoints excluded. Each unordered pair counts once on undirected graphs; directed pairs count separately. Exogenous contributions can be negative when removing a node increases the remaining nodes' betweenness.

degree Simple degree in the chosen base direction. On an undirected graph the exogenous result equals degree. On a directed graph, outgoing base degree produces incoming exogenous degree, and incoming base degree produces outgoing exogenous degree.

Usage

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


Arguments

<code>x</code>	Network input accepted by centrality .
<code>mode</code>	Direction of the base measure: all, out or in. Default all.
<code>exogenous_base</code>	One of "reverse_closeness" (default), "betweenness" or "degree". Exact names are required.
<code>...</code>	Additional arguments to centrality .

Details

Uses the simple binary topology: parallel connections count once, self-loops are removed and weights/path inversion are ignored. Mode "all" projects onto the undirected skeleton; "out" and "in" use directed paths when the input is directed. For undirected input all three modes agree. Empty input returns an empty vector; isolates and singletons score zero. Original size includes other components, so adding an isolate can change reverse-closeness scores of connected nodes even though the isolate's own contribution is zero.

`normalized = TRUE` applies cograph's final division by a positive maximum, retaining negative values; it does not normalize the base measure, nor apply the paper's theoretical normalization. If the maximum is nonpositive, values remain unchanged. Arbitrary normalized base scores, such as unit-length eigenvectors, are not supported.

Numerical verification uses independent NetworkX base scores, explicit path enumeration and analytical graphs. Some numerical entries in the paper's Florentine tables could not be reproduced from NetworkX's graph plus the Pucci isolate; this implementation follows the stated definition and does not claim complete published-table or UCINET parity.

Betweenness and reverse-closeness require repeated all-pairs distances, with worst-case $O(N^4)$ time using the current dense kernels. The measure is therefore in the costly tier even when the degree base is selected.

Value

Named numeric vector 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](https://doi.org/10.1016/j.socnet.2010.06.004).

Examples

```
centrality_exogenous(igraph::make_ring(4), exogenous_base = "betweenness")
```

centrality_expected *Expected Centrality*

Description

Sum of neighbor degrees. Simple but effective influence proxy.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of expected centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_expected(adj)
```

centrality_expected_force
Expected Force centrality

Description

Computes Lawyer's Expected Force after exactly two transmission events without recovery. For each seed, enumerate ordered sequences of two infected-to-susceptible edge transmissions. Each sequence produces a three-node infected cluster with D outgoing edges to susceptible nodes. Normalize these D values across all sequences and take their Shannon entropy using natural logarithms (Lawyer 2015, equation 1).

Usage

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

Arguments

- x Network input accepted by [centrality](#).
- ... Additional arguments to [centrality](#). `normalized = TRUE` divides by the maximum score; all-zero results remain zero.

Details

Different event orders or transmitting parents remain distinct even when they infect the same three nodes. A seed and two adjacent neighbors of an undirected triangle form four sequences, not one. Boundary edges are counted individually even when they reach the same susceptible node. This is not entropy over distinct infected sets or over boundary-degree categories, and is not a probability-weighted epidemic simulation.

Uses the simple unweighted graph, retaining direction. In directed graphs, only outgoing infected-to-susceptible arcs transmit or contribute boundary degree, following the paper's directed extension. Loops and duplicate arcs are removed after generic processing. Weights, mode, inversion and cutoff do not affect the result. The weighted extension and horizons other than two events are outside this implementation.

Zero-degree outcomes use the zero-log-zero entropy limit. If no sequence can perform two transmissions, or every resulting cluster has zero onward force, `cograph` returns zero. The latter is an explicit extension of the paper's undefined all-zero normalization, not author-code parity. Isolates and components of at most three nodes therefore score zero. A single positive-force outcome also has entropy zero. Empty input returns no scores. The measure is local and does not establish epidemic probability, outbreak size or predictive accuracy on the supplied graph.

Native computation groups three-node clusters by boundary degree while preserving their event multiplicities. Worst-case time is $O(n^3)$, memory $O(n^2)$, including dense graph preparation. Scores remain independent between components before maximum normalization.

Value

Named numeric vector 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](https://doi.org/10.1038/srep08665).

See Also

[centrality_modified_expected_force](#) for degree adjustment. [centrality_expected](#) computes a different quantity, the sum of neighbor degrees.

Examples

```
centrality_expected_force(igraph::make_graph("Zachary"))
```

centrality_expected_influence_1
Expected Influence (one-step)

Description

Signed-weight sum of a node's edges (Robinaugh, Millner & McNally 2016). The appropriate centrality for networks with positive *and* negative edges (partial-correlation, glasso, signed correlation networks) where treating negative edges as positive magnitudes can be misleading.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	One of "all", "in", "out" for directed graphs. Default "out".
...	Additional arguments passed to centrality .

Value

Named numeric vector of expected-influence values (signed).

References

Robinaugh DJ, Millner AJ, McNally RJ (2016). Identifying highly influential nodes in the complicated grief network. *Journal of Abnormal Psychology*, 125(6), 747-757.

See Also

[centrality_expected_influence_2](#) for the two-step variant, [centrality_strength](#) for the weighted-degree analogue.

Examples

```
# Signed weight matrix (partial correlations, for example)
W <- matrix(c( 0.0,  0.5, -0.3,  0.2,
              0.5,  0.0,  0.4, -0.1,
              -0.3,  0.4,  0.0,  0.6,
              0.2, -0.1,  0.6,  0.0), 4, 4, byrow = TRUE)
rownames(W) <- colnames(W) <- c("A", "B", "C", "D")
centrality_expected_influence_1(W)
```

centrality_expected_influence_2
Expected Influence (two-step)

Description

Two-step signed-weight sum: a node's own expected influence (EI1) plus the weighted sum of its neighbors' EI1 (Robinaugh, Millner & McNally 2016). Captures both the node's direct influence and the influence it exerts indirectly via highly-connected neighbors.

Usage

```
centrality_expected_influence_2(x, mode = "out", ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	One of "all", "in", "out" for directed graphs. Default "out".
...	Additional arguments passed to centrality .

Value

Named numeric vector of two-step expected-influence values.

References

Robinaugh DJ, Millner AJ, McNally RJ (2016). Identifying highly influential nodes in the complicated grief network. *Journal of Abnormal Psychology*, 125(6), 747-757.

See Also

[centrality_expected_influence_1](#) for the one-step variant.

Examples

```
W <- matrix(c( 0.0,  0.5, -0.3,  0.2,
              0.5,  0.0,  0.4, -0.1,
             -0.3,  0.4,  0.0,  0.6,
              0.2, -0.1,  0.6,  0.0), 4, 4, byrow = TRUE)
rownames(W) <- colnames(W) <- c("A", "B", "C", "D")
centrality_expected_influence_2(W)
```

centrality_extended_coreness

Extended neighborhood coreness

Description

Bae and Kim's extended neighborhood coreness sums the neighborhood coreness of every immediate neighbor: $C_{nc+}(i) = \sum_{j \in N(i)} \sum_{l \in N(j)} k_s(l)$. Equivalently, the score is $A^2 k_s$. Here k_s is the core-number vector of the original simple undirected graph. Core numbers are not recomputed inside each neighborhood.

Usage

```
centrality_extended_coreness(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality . With normalized = TRUE, positive scores are divided by their maximum.

Details

Every length-two walk contributes its endpoint's core number, including returns to the focal node and repeated endpoints reached via different neighbors. This is not a sum over distinct nodes at distance two. Isolates score zero. On a tree, it equals the sum of neighboring degrees; on a d-regular graph it equals d cubed. A larger score means more access to core-rich neighborhoods; numerical equivalence does not imply superior spreading prediction for every network.

Uses the simple undirected unweighted skeleton: either direction creates an edge, parallel edges count once and loops are removed. This projection is a cograph convention for inputs outside the published domain. Weights, mode and shortest-path weight inversion do not affect the score.

Value

Named numeric vector 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.

Examples

```
centrality_extended_coreness(igraph::make_ring(6))
```

centrality_extended_gravity
Extended gravity centrality

Description

Ma et al.'s extended gravity score is the sum of the immediate neighbors' raw gravity scores: $G^+(i) = \sum_{j \in N(i)} G(j)$, where $G(j) = \sum_{l: 0 < d(j,l) \leq r} k_s(j)k_s(l)/d(j,l)^2$. Core numbers and hop distances are calculated on the original simple undirected graph. The radius applies around each neighbor j ; it is not a radius around the focal node i . A contribution can therefore reach $r+1$ hops from i , and paths from a neighbor back to i also contribute.

Usage

```
centrality_extended_gravity(x, gravity_radius = 3, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>gravity_radius</code>	Nonnegative hop-distance cutoff, default 3. NULL or infinity includes the entire reachable component. The optional "auto" setting is a cograph extension: round half the mean finite positive hop distance to the nearest integer (ties to even), with minimum one. It is not a parameter rule from Ma et al.
<code>...</code>	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive final scores are divided by their maximum.

Details

Default radius three is the setting used in the original paper. NULL or infinity includes every reachable partner, excluding the gravity source itself. Radius zero and isolates score zero. The outer neighbor sum has no distance penalty. All inner scores remain raw until the final optional max normalization.

Uses the simple undirected unweighted skeleton, with either direction creating an edge, parallel edges counted once and loops removed. This projection is a cograph convention for other inputs. Edge weights, mode, `gravity_mass` and path-weight inversion do not affect this measure: its masses are always k-shell indices. Computation includes all-pairs hop distances, so it can be expensive for large graphs.

Value

Named numeric vector 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](#).

Examples

```
centrality_extended_gravity(igraph::make_ring(6), gravity_radius = 3)
```

centrality_extended_local_bridging

Extended local bridging centrality

Description

Macker's two-hop localized bridging centrality multiplies betweenness of the focal node in its induced closed two-hop neighborhood by its bridging coefficient. Degrees for that coefficient come from the original graph. The ego network includes every edge between the selected vertices. Its shortest paths can be up to four edges long; this is not global betweenness with a path-length cutoff of two. Betweenness uses unordered pairs, excludes endpoints, and is not normalized by ego-network size.

Usage

```
centrality_extended_local_bridging(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality . <code>normalized = TRUE</code> divides final scores by their maximum; all-zero scores remain zero. Ego betweenness is never scaled by ego size.

Details

Uses the same simple undirected unweighted projection and zero conventions as [centrality_localized_bridging](#). Macker's separate weighted model uses link quality for degree and costs for paths; that model is outside this implementation. Native breadth-first path counts cost $O(\text{sum over ego networks of } n_{\text{ego}} \text{ times } (n_{\text{ego}} + m_{\text{ego}}))$, at worst $O(n \text{ to the fourth power})$, with $O(n \text{ squared})$ memory. This measure is marked costly and must be selected explicitly or through `include`.

Value

Named numeric vector 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](https://doi.org/10.1109/MILCOM.2016.7795393).

Examples

```
centrality_extended_local_bridging(igraph::make_graph("Zachary"))
```

```
centrality_extended_mixed_gravity
```

Extended mixed gravitational centrality

Description

Extended mixed gravitational centrality (EMGC), also called IGC+, sums the raw MGC scores of immediate neighbors: $EMGC_i = \sum_{j \in N(i)} MGC_j$. Each inner MGC score uses its own source node j's core number, partner degrees, and original-graph hop distances. The inner radius is centered on j, so a contribution can reach r+1 hops from i. Paths from j back to i are included. The outer neighbor sum has no distance or mass factor.

Usage

```
centrality_extended_mixed_gravity(x, gravity_radius = 3, ...)
```

Arguments

x	Network input accepted by centrality .
gravity_radius	Nonnegative hop-distance cutoff, default three. NULL or infinity includes every reachable partner. Fractional cutoffs include exactly integer hop distances not exceeding them; values below one give zero. The optional "auto" is a cograph heuristic: round half the mean finite positive distance to the nearest integer (ties to even), with minimum one. It is not the cited radius rule and can change when disconnected components are added.
...	Additional arguments to centrality .

Details

Follows the reproduction in Li and Huang (2022), equation 8, attributed to Wang et al. (2018); the original full equations and software have not been inspected. Uses the same skeleton and radius conventions as [centrality_mixed_gravity](#). Default inner radius three follows the reproduced definition; radius one matches the Zoo's literal inner neighbor sum. Optional maximum normalization occurs only after summing raw neighbor scores. Isolates and radii below one score zero. Empty and singleton graphs give no scores and zero, respectively. Dense O(n cubed) time and O(n squared) memory. Verification of these numerical equations does not establish author-software parity or predictive superiority.

Value

Named numeric vector 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](https://doi.org/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](https://doi.org/10.1038/s41598022140053).

Examples

```
centrality_extended_mixed_gravity(igraph::make_ring(6))
```

```
centrality_flow_betweenness
```

Flow Betweenness Centrality

Description

Max-flow based betweenness centrality.

Usage

```
centrality_flow_betweenness(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of flow betweenness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_betweenness](#) for shortest-path variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_flow_betweenness(adj)
```

centrality_gateway *Gateway Coefficient*

Description

Inter-community brokerage weighted by centrality. Combines participation with degree information. Requires community membership.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
membership	Integer vector of community assignments (one per node).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality .

Value

Named numeric vector of gateway coefficient values (0-1).

See Also

[centrality](#) for computing multiple measures at once, [centrality_participation](#) for the simpler participation coefficient.

Examples

```
adj <- matrix(c(0,1,1,0,0, 1,0,1,0,0, 1,1,0,1,0, 0,0,1,0,1, 0,0,0,1,0), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
centrality_gateway(adj, membership = c(1, 1, 1, 2, 2))
```

centrality_generalized_closeness *Generalized Closeness Centrality*

Description

Sum of α^d over all nodes. Generalization of decay centrality matching tidygraph's implementation.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
decay_parameter	Numeric between 0 and 1 (the alpha parameter). Default 0.5.
...	Additional arguments passed to centrality .

Value

Named numeric vector of generalized closeness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_decay](#) (equivalent formulation).

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_generalized_closeness(adj)
```

centrality_giltschmidt *Gil-Schmidt Power Index*

Description

Sum of $1/d(v,w)$ normalized by $(n-1)$. Variant of closeness using harmonic mean of distances.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of Gil-Schmidt power index values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_harmonic](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_giltschmidt(adj)
```

centrality_global_structure

Global structure model centrality

Description

The Global Structure Model (GSM) of Ullah et al. (2021) is $GSM(i) = \exp(k_s(i)/N) \sum_{j \neq i} k_s(j)/d_{ij}$, where k_s denotes original graph core numbers and d denotes hop distances. It combines a focal coreness factor with distance-discounted coreness of other nodes. N is the total original node count, including isolates.

Usage

```
centrality_global_structure(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality . <code>normalized = TRUE</code> divides final scores by their maximum.

Details

Both GSM and [centrality_hybrid_global_structure](#) use the simple undirected skeleton, ignoring weights, mode, path inversion and distance cutoffs. Loops are removed and parallel connections count once. Only reachable partners contribute; this is an explicit disconnected-graph extension. Isolates and singletons score zero, empty input returns an empty vector. Other components can affect results through the global node count and, for H-GSM, its global mean. These are not independent per-component calculations.

Production uses native coreness and all-pairs distance kernels, with worst-case $O(N^3)$ time and $O(N^2)$ memory. Numerical verification uses independent NetworkX cores/distances and exhaustive small-graph oracles. Agreement with a numerical definition does not establish author-software parity or superior epidemic-spreading predictions.

Value

Named numeric vector 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](https://doi.org/10.1038/s4159802184684x).

Examples

```
centrality_global_structure(igraph::make_ring(4))
```

```
centrality_graph_regularization
```

Graph regularization centrality

Description

Dal Col and Petronetto's graph regularization centrality is $GRC_i = 1/[(I + \gamma L)^{-1}]_{ii}$, where L is the unnormalized weighted graph Laplacian. The i th column of this inverse minimizes $\|s - e_i\|^2 + \gamma s^T L s$. A larger score indicates that smoothing retains less of a unit impulse at its source vertex. This implements the centrality with unit impulses; the author's separate signal option returns smoothed signal values and is not this centrality.

Usage

```
centrality_graph_regularization(x, grc_gamma = 1, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>grc_gamma</code>	Finite nonnegative regularization strength, default one.
<code>...</code>	Additional arguments to centrality .

Details

`grc_gamma` accepts any finite nonnegative number, default one. At zero every score is one. Isolates also score one. Within a component of n vertices scores lie between one and n , approaching n as γ grows without bound. Adding disconnected components does not change existing raw scores. Edge weights and γ act multiplicatively; uniform weight scaling changes scores unless γ is adjusted inversely.

Uses finite nonnegative edge weights when `weighted = TRUE`. Zero weights are absent connections. Unweighted inputs use the simple undirected skeleton. Loops are removed. For weighted directed inputs, opposite arcs are added. The generic `simplify` argument combines parallel edges first; remaining weighted parallel edges are added. These projections are explicit cograph conventions for the published undirected domain. Generic mode, shortest-path weight inversion and cutoff do not affect the result.

The native dense spectral calculation separates each component's constant eigenvector and evaluates the remaining filter in log space. This supports extreme finite γ and uniform weight scales without forming their product. Unresolvable weight ranges or positive spectral condition numbers above $1/(64 \text{ times machine epsilon})$ raise an error. Runtime is $O(n^3)$ and memory $O(n^2)$ per component. Empty graphs return an empty vector.

The author software approximates the same filter with ten Chebyshev terms. This function evaluates the defining inverse to numerical precision; default author-software values need not coincide. Optional `normalized = TRUE` divides scores by their global maximum.

Value

Named numeric vector 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](https://doi.org/10.1016/j.physa.2023.129188).
Dal Col, A. (2023). GRC. Mendeley Data, version 1. [doi:10.17632/ns63f5dj86.1](https://doi.org/10.17632/ns63f5dj86.1).

Examples

```
centrality_graph_regularization(igraph::make_ring(4), grc_gamma = 0.5)
```

centrality_gravity	<i>Gravity centrality</i>
--------------------	---------------------------

Description

$G(i) = \sum_j m_i m_j / d_{ij}^2$, optionally truncated at gravity_radius. The published members of the family differ only in the mass and the reach:

Usage

```
centrality_gravity(  
  x,  
  mode = "all",  
  gravity_mass = "kshell",  
  gravity_radius = 3,  
  ...  
)
```

Arguments

- x Network input: matrix, igraph, network, cograph_network, or tna object.
- mode Direction: "all", "out" or "in".
- gravity_mass "kshell" (default), "degree", or "legacy".
- gravity_radius Largest distance to include: a number, "auto" for half the mean distance, or NULL for the whole graph. Default 3.
- ... Additional arguments passed to [centrality](#).

Details

Gravity centrality (Ma, Ma, Zhang & Wang 2016) k-shell mass, radius 3 – the default.

Gravity model (Li, Ren, Ma, Liu, Zhang & Zhou 2019, eq. 1) gravity_mass = "degree", gravity_radius = NULL.

Local gravity model (same paper, eq. 2) gravity_mass = "degree", gravity_radius = "auto", which uses their empirical half-mean-distance heuristic (eq. 5). cograph rounds to the nearest integer (ties to even), with minimum 1, using finite positive distances on disconnected graphs. These rounding and disconnected-graph rules are cograph conventions.

Value

Named numeric vector, one value per node.

Change in 2.4.8

Before 2.4.8 this measure computed $\sum_j k_j s_j / d_{ij}^2$: the product of degree and k-shell on the partner, no mass at all on the focal node, and no truncation. That is not the formula of Li et al. (2019) that its help page cited, and dropping the focal mass changes the ranking rather than the scale. The default is now Ma et al. (2016). gravity_mass = "legacy" with gravity_radius = NULL reproduces the earlier values exactly.

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_coreness](#), [centrality_kreach](#), [centrality](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_gravity(adj)
centrality_gravity(adj, gravity_mass = "degree", gravity_radius = NULL)
```

centrality_harary	<i>Harary Centrality</i>
-------------------	--------------------------

Description

Sum of $1/d^2$ over all reachable node pairs. Robust to disconnected graphs.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of Harary centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_harary(adj)
```

centrality_harmonic	<i>Harmonic Centrality</i>
---------------------	----------------------------

Description

Sum of inverse shortest path distances to all other nodes. Unlike closeness, harmonic centrality handles disconnected graphs naturally (unreachable nodes contribute 0 instead of making the measure undefined).

Usage

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

```
centrality_inharmonic(x, ...)
```

```
centrality_outharmonic(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of harmonic centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_closeness](#) for the traditional variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_harmonic(adj)
```

centrality_hcc

Hybrid characteristic centrality

Description

Liu and Zheng's hybrid characteristic centrality adds a local and a global characteristic on the same scale: $HCC(u) = k^{ex}(u)/k_{max}^{ex} + pos(u)/pos_{max}$. The local half is the *extended degree* $k^{ex}(u) = \delta k(u) + (1 - \delta) \sum_{v \in \phi(u)} k(v)$, the node's own degree blended with its neighbors'; the global half is the *E-shell* position index, the round in which a repeated minimum-extended-degree peel removes the node. Both terms are divided by their largest value, so each lies in $[0, 1]$ and the raw score lies in $[0, 2]$.

Usage

```
centrality_hcc(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality , including hcc_delta.

Details

The E-shell hierarchy decomposition is not a k-shell decomposition, although the Centrality Zoo describes it as "a variant of k-shell decomposition". There is no outer loop over a shell index and no repeat-until-stable inner loop: each round removes exactly the set of remaining nodes attaining the current minimum extended degree, recomputes the extended degrees on what is left, and tags the removed nodes with the round number. The position indexes therefore run $1, \dots, pos_{max}$ with every value attained, rather than being shell numbers, and the two procedures disagree on the source's own figure 1.

The source's printed algorithm contains a typo, and cograph implements the correction its own tables require. Step 3 of the E-shell procedure prints $S_p = \arg \max_{u \in G_p} \{k^{ex}(u)\}$ while the same sentence calls S_p "the set of minimum nodes", the preceding paragraph says "the nodes with minimum extended degree are found and deleted", and the paper's table 2 heads its column "Minimum extended degree" with the increasing values 2, 2.5, 3, 4.5, 5, 6. The minimum reading reproduces every printed row; the literal maximum reading is a different measure.

The peel recomputes but equation (4) does not. Step 6 updates the extended degrees on the residual graph after every removal, which is what the printed table 2 minima require. The $k^{ex}(u)$ of equation (4), and the k_{max}^{ex} it is divided by, are nevertheless the **original**-graph values: the paper's own worked line $HCC(a) = 4.5/11 + 4/6$ uses the original 4.5 and the original maximum 11, and its node d settles the question, since its original 9.5 gives the printed 1.86 while its residual 6 at removal time would give 1.55.

Raw scores are not component-local. k_{max}^{ex} and pos_{max} are single global constants, so adding a disconnected component – an isolate included – can change every score, and not merely by a common factor, because the two terms rescale independently. The source does not discuss disconnected graphs.

Degenerate cases are cograph decisions, not the source's. An isolate has extended degree zero, which for $\delta \in [0, 1]$ is the global minimum, so it always leaves in the first round with $pos = 1$. On an edgeless graph every extended degree is zero and $k_{max}^{ex} = 0$, making the first term $0/0$; it is written as **zero**, which leaves the E-shell term alone. One round then removes everything, so every node of an edgeless graph – a singleton included – scores exactly 1. Empty graphs return no scores.

`hcc_delta` defaults to the source's 0.5 and is restricted to the source's stated domain $[0, 1]$, where $\delta = 1$ recovers the classical degree and $\delta = 0$ drops the node's own degree entirely. Values outside that interval are refused with a `cograph_bad_parameter` error rather than extended: they make the extended degree negative on some graphs, and then equation (4) divides by a nonpositive maximum, which the source never contemplates.

Uses the simple undirected unweighted skeleton, the source's stated domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and path-weight inversion are ignored, and directed input is symmetrized rather than read as a directed case, which the source does not define. The source states no further normalization; `normalized = TRUE` max-scales the finished vector as elsewhere in [centrality](#), on top of the two divisions equation (4) already performs. Cost is one dense matrix-vector product per peeling round, so $O(n^3)$ in the worst case rather than the $O(n + m)$ a sparse min-heap would give.

Numerical verification establishes agreement with the definition and with the values the source prints for its figure 1, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](https://doi.org/10.1038/s41598023-303085).

See Also

[centrality_ehcc](#) for the neighborhood sum of this score, [centrality_dkgm](#) for another shell-and-degree hybrid, and [list_centralities](#) for the catalogue.

Examples

```
# Every node of a regular graph has the same extended degree, so one
# round removes the whole graph and every node scores 1 + 1 = 2.
centrality_hcc(igraph::make_ring(6))

# A star peels its leaves first and its center second.
centrality_hcc(igraph::make_star(6, mode = "undirected"))

# delta = 1 is the classical degree in the extended-degree slot.
centrality_hcc(igraph::make_star(6, mode = "undirected"), hcc_delta = 1)
```

centrality_heatmap	<i>Heatmap, Flow Coefficient, Local Entropy, Weighted h-index, Redundancy</i>
--------------------	---

Description

Five local measures.

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 (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
...	Additional arguments passed to centrality .

Details

heatmap (**Duron 2020**) Farness minus the mean farness of the neighbors, $C(v) = f(v) - \frac{1}{k_v} \sum_{u \in N(v)} f(u)$, with f the sum of hop distances to reachable nodes. **Lower is more central**. Isolates score NaN. Reproduces Table 1 of the paper.

flow_coefficient (**Honey et al. 2007**) Among ordered pairs of distinct neighbors, the fraction joined by a two-step path through the node but not by a direct link, as implemented in the Brain Connectivity Toolbox. On an undirected graph it equals one minus the clustering coefficient; it carries new information only on directed graphs. Nodes with fewer than two neighbors score 0.

local_entropy (**Nie et al. 2016**) $-\sum_{j \in N(i)} k_j \ln k_j$, as printed by the sources. Always non-positive and more negative for larger, denser neighborhoods, so **lower is more central**; isolates score 0, the maximum. The original article is closed access; the formula is that of the Zoo and of Omar and Plapper's 2021 survey, which agree.

weighted_h_index (**Gao et al. 2019**) h-index of the multiset in which each neighbor j contributes the topological weight $k_i k_j$ repeated k_j times. Edge weights on the input play no role.

redundancy (**Burt 1992; Borgatti 1997**) Mean degree of the node's neighbors within its ego network, $2t_i/k_i$; equal to degree minus effective size. Higher = fewer structural holes. Reproduces Borgatti's worked example.

heatmap, local_entropy and weighted_h_index follow mode; the others ignore direction. Edge weights are ignored.

Value

Named numeric vector, one value per node.

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](#).

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_heatmap(star5)
centrality_weighted_h_index(star5)
centrality_redundancy(star5)
```

centrality_hubbell	<i>Hubbell Centrality</i>
--------------------	---------------------------

Description

Hubbell (1965) input-output centrality: $C = (I - wW)^{-1}\mathbf{1}$, where W is the (weighted) adjacency matrix and w is a weight factor that must satisfy $w \cdot \rho(W) < 1$ for the system to be solvable.

Usage

```
centrality_hubbell(x, hubbell_weight = 0.5, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>hubbell_weight</code>	Attenuation factor w . Default 0.5. If $w \cdot \rho(W) \geq 1$, the function returns NA with a warning.
<code>...</code>	Additional arguments passed to centrality .

Details

Bit-exact match against `centiserve::hubbell` when edge weights are passed explicitly (cograph mirrors centiserve's full-inverse LAPACK call path).

Value

Named numeric vector of Hubbell centrality values (or NA if the system is not solvable).

Note on centiserve equivalence

`centiserve::hubbell(g, weights = NULL)` silently resets all edge weights to 1, ignoring the graph's weight attribute. To reproduce cograph's values with centiserve on a weighted graph, pass `weights = igraph::E(g)$weight` explicitly.

References

Hubbell, C. H. (1965). An input-output approach to clique identification. *Sociometry*, 28(4), 377-399.

See Also

[centrality](#), [centrality_katz](#).

Examples

```
# Small weighted path graph; spectral radius permits weightfactor = 0.5
adj <- matrix(0, 4, 4)
adj[1,2] <- adj[2,1] <- adj[2,3] <- adj[3,2] <- adj[3,4] <- adj[4,3] <- 0.3
rownames(adj) <- colnames(adj) <- LETTERS[1:4]
centrality_hubbell(adj, hubbell_weight = 0.5)
```

centrality_hybrid_global_structure

Hybrid global structure model centrality

Description

Mukhtar et al.'s H-GSM (2023) uses $s_i = \exp(k_s(i)k_i/N)$, $a = \lceil \log_2(N^{-1} \sum_i s_i) \rceil$, and $H-GSM(i) = s_i \sum_{j \neq i} s_j / d_{ij}^a$. k_i is simple degree, $k_s(i)$ is original coreness, and d is hop distance. The ceiling exponent is computed from the mean self-influence over ALL original nodes, including isolates whose self-influence is one. The factor s_i alone is not the final centrality score.

Usage

```
centrality_hybrid_global_structure(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality .

Details

Topology and disconnected-graph conventions are shared with [centrality_global_structure](#). The adaptive exponent is used exactly as specified, including its discontinuities at powers of two; it is not smoothed or replaced by a fixed exponent.

Self-influence, its mean and final sums are evaluated in logarithmic form. Raw scores exceeding double precision raise an error. With `normalized = TRUE`, final scores are computed directly as exponentials of log-score differences, so normalized results remain available even when raw scores overflow. Extremely small normalized ratios may underflow to zero. Normalization is applied to the complete score, not separately to self-influence or neighbor contributions.

Value

Named numeric vector 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.

Examples

```
centrality_hybrid_global_structure(igraph::make_ring(4))
```

centrality_iec	<i>Immediate Effects Centrality</i>
----------------	-------------------------------------

Description

Friedkin's immediate effects centrality scores a node by how quickly the rest of the network's influence reaches it. Actors whose effects travel over long sequences of interpersonal influence are more dependent on intervening actors than those whose effects travel over short ones, so the measure is the reciprocal of the mean length of the influence sequences that end at a node. Writing W for the row-stochastic influence matrix, c for its left eigenvector at eigenvalue one, $Z = (I - W + \mathbf{1}c')^{-1}$ for the fundamental matrix, Z_{dg} for Z with its off-diagonal entries set to zero and E for the all-ones matrix, the mean lengths are $M = (I - Z + EZ_{dg}) \text{diag}(1/c)$ and the score is $c_{IEC}(j) = (n - 1) / \sum_{i \neq j} m_{ij}$. M is the mean first passage time matrix of the chain, so the sum runs *down* column j and a high score marks a node the network reaches fast.

Usage

```
centrality_iec(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality .

Details

The influence matrix carries a unit self-loop, and the self-loop is load-bearing. The source builds W by setting the diagonal of the adjacency matrix to one and dividing each row by its sum, $w_{ij} = a_{ij} / \sum_j a_{ij}$ with $a_{ii} = 1$, a construction it attributes to French (1956) and states twice on page 1494, once in the body and once in the note to Table 1. Its footnote 10 says why the diagonal is there: a strong network with $w_{ii} > 0$ must be regular, meaning aperiodic, and its footnote 9 gives the two-cycle counterexample that a zero diagonal admits. An implementation that drops the self-loop is not computing this measure on a different scale, it is computing a different measure.

This is not cograph's centrality_markov, and the difference is not a rescaling. The two are both built from mean first passage times and are easy to confuse – cograph's own candidate ledger confused them for several rounds – but they differ twice over. markov normalizes A without adding the diagonal, and it divides the column sum by n , counting the excluded diagonal entry, where equation (20) divides by $n - 1$. The second difference is a constant factor $n/(n - 1)$ and cannot reorder anything; the first can and does. On the five-node star markov gives 1.25, 0.161, 0.161, 0.161, 0.161 where iec gives 0.5, 0.08, 0.08, 0.08, 0.08, and the two rank the nodes differently on 2 of the 21 connected five-node graphs. Both are kept: markov is the older behavior that existing results depend on, iec is Friedkin's published measure.

Reducible input is refused, not extended. Equation (11) needs an irreducible chain. Without one the eigenvector of equation (9) has a dimension per closed class, so c is not determined, and $\text{diag}(1/c)$ is undefined wherever c vanishes. The danger is that the closed form does not announce the failure: for i and j in different blocks $z_{ij} = 0$, and equation (11) then returns the entirely finite $m_{ij} = z_{jj}/c_j$ in place of an infinite mean first passage time. Rather than publish a finite wrong number, cograph tests the chain first and returns NA at every node with a cograph_undefined_measure warning. In practice the test is connectedness of an undirected graph and strong connectedness of a directed one, since the mandated self-loops settle aperiodicity for free. Friedkin restricts his own analysis to regular networks and never defines the measure outside them. centrality_rsp_betweenness answers on disconnected input because *its* source states a rule for an unreachable pair; this one states none, and a component-wise reading would additionally have to invent whether the $n - 1$ of equation (20) counts the component or the network.

A singleton is NA and an empty graph returns no scores. Equation (20) divides by $n - 1$, which is zero when $n = 1$; the same NA and the same warning follow. An isolate never appears on its own, because a graph containing one is reducible and is already NA everywhere.

Direction is kept; weights, loops and parallel edges are not. W is a matrix of directed influence, row i being what actor i attends to, so a directed input is used as it stands and the measure needs a strongly connected one. There is no in/out/all variant to choose between, so mode, cutoff and invert_weights are ignored. Weights are dropped, deliberately: $a_{ii} = 1$ is calibrated against $a_{ij} = 1$, so multiplying every weight by a constant would silently re-weight each actor's self-reliance against the network, and the source demonstrates only the binary case. Loops in the input are absorbed by the mandated unit diagonal and parallel edges collapse, since $a_{ij} = 1$ "wherever a line exists between two points". The source states no normalization, so normalized = TRUE max-scales the finished vector as elsewhere in centrality.

The source prints a complete numerical fixture. Table 1, pages 1492-1494, gives this measure to three decimals for every node of all 21 connected non-isomorphic five-node graphs. All 105 printed values are reproduced by this implementation; see the batch 49 published audit in the package's verification directory.

Value

Named numeric vector in input node order, NA at every node when the influence chain is reducible or the graph 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](#) for the older, and different, mean-first-passage measure, [centrality_random_walk](#) for another chain-based score, and [list_centralities](#) for the catalogue.

Examples

```
# On a complete graph W = J/n, so Z = I, every mean first passage time is
# n, and the score is (n - 1) / (n (n - 1)) = 1/n. Friedkin's Table 1
# prints .200 for the five-node case.
centrality_iec(igraph::make_full_graph(5))

# The five-node star is row 1 of that table: .500 at the center and .080
# at each leaf.
centrality_iec(igraph::make_star(5, mode = "undirected"))

# A disconnected graph has no answer: the influence chain is reducible,
# so every node is NA and a warning says why.
two <- matrix(0, 4, 4)
two[1, 2] <- two[2, 1] <- two[3, 4] <- two[4, 3] <- 1
tryCatch(centrality_iec(two), warning = conditionMessage)
```

centrality_iira

*Improved iterative resource allocation (IIRA)***Description**

IIRA is [centrality_ira](#) with the receiver's share scaled by how much of a spreading process that receiver could actually carry: $a_{ij} = [1 - (1 - \beta)^{k_i}] \theta_i (\sum_{u \in \Gamma(j)} \theta_u)^{-1}$, where k_i is the degree of i and β the spreading rate. The recursion and the initial condition $I(0) = (1, \dots, 1)$ are unchanged; there is no α exponent, and the denominator keeps the plain masses.

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, the k-shell index the source's worked example uses) or "degree". Shared with centrality_ira .

iira_beta	Spreading rate β , a single number in $(0, 1]$; default 0.2, the source's worked-example value. The source sweeps β in its experiments and recommends no other default.
iira_steps	Number of iterations t , a single nonnegative whole number; default 50, the source's worked-example value. Zero returns $I(0)$.
...	Additional arguments to centrality .

Details

The scores are tiny and only their order means anything. The factor $\psi_i = 1 - (1 - \beta)^{k_i}$ is strictly below one, so every column of A sums to less than one, the spectral radius is below one, and $I(t) \rightarrow 0$ geometrically. The source runs exactly $t = 50$ steps and prints an $I(50)$ of order 10^{-20} ; `cograph` returns that raw vector, so the printed example is reproducible, and `normalized = TRUE` max-scales it into $[0, 1]$ for reading. Never compare raw IIRA scores across connected components: each component decays at its own rate, so after `iira_steps` steps they sit on different exponential scales. A large `iira_steps` underflows to zero.

The Centrality Zoo entry is not this formula. Section 2.185 prints $p_{ij} = (1 - (1 - \beta)^{d_i}) a_{ij} c_i / \sum_k a_{ik} c_k$, which pairs the numerator's index with the denominator's own neighborhood; the source pairs them with opposite sets. As printed, the Zoo's row sums are $\psi_i c_i d_i / \sum_{k \in N(i)} c_k$, so its matrix is stochastic in neither direction although the entry calls it stochastic, and it does not reproduce the source's printed matrix or its printed $I(50)$. `cograph` implements the source.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and path-weight inversion are ignored. An isolate has an empty neighbor sum and $\psi = 0$, so it scores zero from the first step; that is the value of the source's empty sum, not an accidental zero. `iira_steps = 0` returns the initial $I(0)$, a vector of ones. Empty graphs return no scores. Cost is one dense n^2 matrix plus `iira_steps` matrix-vector products.

The version of record was not read: what was read is the author preprint arXiv:1505.03214v1, whose method section, worked example and figures carry the definition reproduced here. Numerical verification establishes agreement with those equations and with every value printed in the preprint's figure 2 example, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](#) for the measure this improves, and [list_centralities](#) for the catalogue.

Examples

```
# The source's figure 2, whose printed I(50) is
# 8.19e-20, 4.32e-20, 4.32e-20, 6.7e-21, 6.7e-21
fig2 <- igraph::make_graph(c(1, 2, 1, 3, 2, 3, 1, 4, 1, 5),
                           directed = FALSE)

centrality_iira(fig2)

# Only the order carries meaning, so max-scale for reading
centrality_iira(fig2, normalized = TRUE)
```

centrality_improved_closeness

Improved closeness centrality

Description

Luan et al.'s improved closeness is $ICC(i) = (n - 1) / \sum_{j \neq i} d_{ij} / \sigma_{ij}^\alpha$, where d is the hop distance and σ counts shortest paths. Multiple shortest paths reduce the effective distance to a partner. At alpha zero this is ordinary normalized closeness on a connected graph; on a tree it is independent of alpha because each pair has one shortest path. Scores need not be bounded by one.

Usage

```
centrality_improved_closeness(x, icc_alpha = 0.2, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>icc_alpha</code>	Multiplicity exponent between zero and one, default 0.2.
<code>...</code>	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive final scores are divided by their maximum. The published $n-1$ factor is present in raw scores already.

Details

Uses the simple undirected unweighted skeleton: either direction creates an edge, parallel edges count once and self-loops are removed. Weights, mode and path-weight inversion do not affect the result. These are explicit cograph projections to the published domain.

In a disconnected graph, every node has an unreachable partner and therefore scores zero under the global infinite-distance convention. Singletons score zero by an explicit cograph convention for the otherwise undefined zero-over-zero expression. For within-component scores, supply each component separately. Empty input returns an empty vector.

Breadth-first traversal counts shortest paths in logarithmic form, avoiding overflow when the number of paths exceeds double precision. Extremely small effective-distance terms can underflow to zero, but direct-neighbor terms remain one and keep the denominator positive. Computation costs $O(n \text{ times } (n+m))$ with an additional dense adjacency representation. Default alpha 0.2 is a setting studied in the source, not an estimate or a guarantee of optimal spreading predictions.

Value

Named numeric vector 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/s1142402101117.

Examples

```
centrality_improved_closeness(igraph::make_ring(4), icc_alpha = 0.2)
```

```
centrality_improved_global_structure
```

Improved global structure model centrality

Description

The IGSM definition reproduced in Mukhtar et al. (2023), equation 5, is $IGSM(i) = \exp(k_i/N) \sum_{j \neq i} k_j / d_{ij}^a$, with $a = \lceil \log_2(\bar{k}) \rceil$. The original method is attributed to Zhu and Wang (2022); the exact equation used here was checked in the later primary experimental paper, not its original full text. IGSM uses simple degrees rather than GSM's core numbers, and its distance exponent depends on global mean degree, including isolates.

Usage

```
centrality_improved_global_structure(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality .

Details

Topology, normalization and disconnected-graph conventions follow [centrality_global_structure](#). For a positive mean degree below one, the exponent may be zero or negative; it is not clamped. With a negative exponent, more distant reachable partners contribute more, an explicit consequence of extending the equation to sparse disconnected inputs. Unreachable partners still contribute zero. Edgeless graphs score zero by an explicit extension because the logarithm of zero in the exponent is otherwise undefined.

This implements IGSM itself, without an additional nearest-neighbor aggregation for the extended IGSM variant.

Value

Named numeric vector 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](https://doi.org/10.1088/16741056/ac380d).

Examples

```
centrality_improved_global_structure(igraph::make_ring(4))
```

centrality_information

Information Centrality (Stephenson-Zelen)

Description

Information centrality (Stephenson & Zelen 1989) measures a node's importance in terms of the "information" contained in all paths (not only shortest) passing through it. Defined via the inverse of a Laplacian-like matrix, yielding per-node $IC_i = 1/(C_{ii} + (\text{tr}(C) - 2R_i)/n)$ where $C = A^{-1}$ and R_i is the row sum of C .

Usage

```
centrality_information(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::infocent` on connected undirected graphs (cograph mirrors sna's exact construction and call sequence).

Value

Named numeric vector of information centrality values.

References

Stephenson, K., & Zelen, M. (1989). Rethinking centrality: Methods and examples. *Social Networks*, 11(1), 1-37.

See Also

[centrality](#), [centrality_current_flow_closeness](#).

Examples

```
adj <- matrix(c(0,1,1,0, 1,0,1,1, 1,1,0,1, 0,1,1,0), 4, 4)
rownames(adj) <- colnames(adj) <- LETTERS[1:4]
centrality_information(adj)
```

centrality_integration

Integration Centrality

Description

Distance-based influence: sum of $1 - (d-1)/\max(d)$ over all nodes.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of integration centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_integration(adj)
```

centrality_ira *Iterative resource allocation (IRA)*

Description

Every node starts with one unit of resource and hands it to its neighbors in proportion to the *receiver's* centrality, repeatedly, until the amounts stop moving. The share node j sends to a neighbor i is $a_{ij} = \theta_i^\alpha / \sum_{u \in \Gamma(j)} \theta_u^\alpha$, the recursion is $I(t+1) = AI(t)$ from $I(0) = (1, \dots, 1)$, and the steady state I ranks the spreaders. Because every non-isolate column of A sums to one, the total resource is conserved: $\sum_i I_i(t) = n$ at every step on a graph with no isolates, and each connected component keeps its own vertex count.

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, the k-shell index the source's worked example uses) or "degree". The source also mentions closeness and betweenness.
ira_alpha	Exponent α on the mass, a single finite number; default one, the only value the source uses.
ira_tol	Stopping tolerance ε on the largest absolute change between successive iterates; default 1e-6, the source's own value.
ira_max_iter	Iteration bound, a single whole number of at least one; default 1000. Reaching it raises <code>cograph_no_converge</code> .
...	Additional arguments to centrality .

Details

The equilibrium has a closed form. Writing $s_i = \sum_{u \in \Gamma(i)} \theta_u^\alpha$, the limit is $I_i \propto \theta_i^\alpha s_i$ within each component, scaled so the component's scores sum to its size. On the source's own figure 1(a) that reproduces the printed $[15/8, 5/4, 5/4, 5/16, 5/16]$ exactly. `cograph` nevertheless iterates, because the iteration is what the source defines and what its table reports, and because the closed form is a limit that need not exist; see the next paragraph.

The iteration does not always converge, and `cograph` says so. A is the transition matrix of a reversible walk, so on a bipartite component it has an eigenvalue of exactly -1 . The coefficient of

that eigenvector in $I(0) = (1, \dots, 1)$ is the difference in size between the component's two vertex classes, so the iteration settles into a period-two cycle, never meets `ira_tol`, and returns a value that depends on the parity of the last step. The three-star alternates for ever between $(3, 1/3, 1/3, 1/3)$ and $(1, 1, 1, 1)$, while the four-path, whose classes are equal, converges to $(2/3, 4/3, 4/3, 2/3)$. Neither the source nor the Centrality Zoo mentions this. `cograph` runs the source's own rule, stops at `ira_max_iter`, raises a `cograph_no_converge` warning naming the largest remaining change, and returns I at `ira_max_iter`. It does not silently report that iterate as an equilibrium, and it does not substitute the average of the two alternating iterates, which would converge but is not the source's rule. Every graph in the source's own figure 1 carries a triangle and converges.

The Centrality Zoo (section 2.204) states the transpose, $p_{ij} = a_{ij}c_j^\alpha / \sum_k a_{ik}c_k^\alpha$, and asks for the principal left eigenvector of P . That is the same object up to scale on a graph where the limit exists, but it is not the source's finite iteration: it sidesteps the parity problem instead of reporting it, and it carries no $\sum_i I_i = n$ scale.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and path-weight inversion are ignored. An isolate is in nobody's neighborhood, so it receives nothing and its own unit is not passed on: it scores zero from the first step, which is the value of the source's empty sum and not an accidental zero, and it is the reason $\sum_i I_i = n$ is stated only for graphs with no isolates. Empty graphs return no scores. Cost is one dense n^2 matrix plus one matrix-vector product per iteration.

Numerical verification establishes agreement with the source equations and with every value printed in the source's table 1, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](https://doi.org/10.1209/02955075/106/48005).

See Also

[centrality_iira](#) for the improved variant, and [list_centralities](#) for the catalogue.

Examples

```
# The source's figure 1(a): a triangle with two pendants on one corner.
# The printed steady state is 15/8, 5/4, 5/4, 5/16, 5/16.
fig1a <- igraph::make_graph(c(1, 2, 1, 3, 2, 3, 1, 4, 1, 5),
                           directed = FALSE)

centrality_ira(fig1a)

# The source's other mass, and a nonlinear exponent
centrality_ira(fig1a, ira_mass = "degree", ira_alpha = 2)
```

centrality_katz	<i>Katz Centrality</i>
-----------------	------------------------

Description

Katz (1953) status index: $C = (I - \alpha A^T)^{-1} \mathbf{1}$. Each node's score sums attenuated walks of every length back to it, with attenuation α applied per step. Rankings are identical to Bonacich's alpha centrality with a uniform exogenous vector.

Usage

```
centrality_katz(x, katz_alpha = 0.1, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
katz_alpha	Attenuation factor. Must satisfy $\alpha < 1/\rho(A)$ where $\rho(A)$ is the spectral radius. Default 0.1 matches centiserve and NetworkX conventions.
...	Additional arguments passed to centrality .

Details

Equivalence is verified bit-exact against `centiserve::katzcent` (cograph mirrors centiserve's exact LAPACK call sequence) and at machine epsilon against `igraph::alpha_centrality(exo = 1)` and `networkx.katz_centrality_numpy`.

Value

Named numeric vector of Katz centrality values.

References

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39-43.

See Also

[centrality](#), [centrality_eigenvector](#), [centrality_pagerank](#).

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_katz(adj)
```

centrality_ked	<i>KED method centrality</i>
----------------	------------------------------

Description

The KED method of Chen, Xiao, Zeng and Zhang combines how many local paths leave a node with how diverse they are: $KED(i) = k_i (1 + H_i) \exp(K_i/N)$, where $K_i = \sum_{j \in N(i)} k_j$ is the sum of the neighbors' degrees, $H_i = (\sum_{j \in N(i)} -p_j \log p_j) / \log k_i$ with $p_j = k_j / K_i$ is the normalized entropy of the neighbor-degree distribution, and N is the number of nodes in the whole graph. The source calls K_i the local path number and H_i the path diversity: two nodes of equal degree with equally many second neighbors are separated by how evenly their neighbors carry those paths.

Usage

```
centrality_ked(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality .

Details

H_i is a ratio of two logarithms in the same base – equation (2) divides the entropy by the entropy of the uniform distribution on k_i outcomes – so the base cancels and no choice of base is being made. H_i lies in $[0, 1]$ and is exactly one when the neighbor degrees are all equal, which is the source's $1 \leq E_i \leq 2$.

The measure takes no parameters. Equation (6) is a bare product; the exponents α and β the Centrality Zoo attributes to Chen et al. appear nowhere in the paper, and none is offered here.

Two degenerate cases are cograph decisions, not the source's. A node with one neighbor has $p = 1$, so its entropy is zero, and its normalizer $\log k_i$ is zero too: H_i is $0/0$ and is written as **zero**, giving $E_i = 1$. That is the value approached from $k_i = 2$ as one neighbor's share vanishes, and the one that gives a node with a single path the least path diversity; the alternative reading of $0/0$ as "the entropy equals its own maximum, so $H = 1$ " would double every leaf's score. An isolate has both sums empty; H_i is written as zero there as well, and the score is zero whatever finite E_i is chosen, because k_i multiplies the product. Empty graphs return no scores.

Raw scores are not comparable across graphs of different order. N in D_i is the vertex count of the whole network, as the source's own table 1 defines it, so adding a disconnected component – an isolate included – changes every score, and unlike a plain rescaling it can also change the ranking, because $\exp(K_i/N)$ shrinks the large K_i more than the small.

The source's stated range $1 \leq D_i \leq e$ is not general. It holds exactly when $K_i \leq N$, which is true of the sparse toy networks of its figure 1 and false on dense graphs: every node of K_5 has $K_i = 16$ against $N = 5$, so $D_i = e^{3.2}$. cograph implements the formula, not the range claim. Scores can therefore be large; $K_i/N \leq (n-1)^2/n$, so nothing overflows below about 710 vertices even on a complete graph, and an overflow beyond that raises an error rather than returning Inf.

This is not the Centrality Zoo's formula. Zoo section 2.215 writes $c_{KED}(i) = k_i E_i^\alpha D_i^\beta$ with $E_i = (\sum_j -p_j \log p_j) / \log k_i$ and $D_i = \exp(K_i / \max_l K_l)$: it drops the 1+ from E_i and divides by the largest cluster degree instead of by N . On the source's own figure 1 that reading gives 13.5914 and 6.5672 where the paper prints 25.9187 and 19.2212, which cograph reproduces. The $\max_l K_l$ denominator is a plausible misreading, since it makes the paper's stated $1 \leq D_i \leq e$ hold, but it reproduces neither printed number. cograph implements the paper and offers no Zoo variant.

Uses the simple undirected unweighted skeleton, the source's undirected domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and path-weight inversion are ignored. The source also defines a directed variant (its equation 3, replacing the neighborhood by the out-neighborhood and k_i by k_i^{out}); that variant is not implemented, so a directed input is symmetrized rather than being read as the paper's directed case. The source states no normalization; normalized = TRUE max-scales the finished vector as elsewhere in [centrality](#). Cost is two sparse matrix-vector products, $O(n + m)$.

Numerical verification establishes agreement with the two scores the source prints for its figure 1, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](https://doi.org/10.1209/02955075/104/68006).

See Also

[centrality_lnc](#) and [centrality_neighbor_distance](#) for other neighbor-degree sums, [centrality_entropy](#) for a plain neighborhood entropy, and [list_centralities](#) for the catalogue.

Examples

```
# Every node of a ring has two neighbors of degree two, so the
# neighbor degrees are even, H is one and the score is 4 exp(4 / n)
centrality_ked(igraph::make_ring(8))

# A star: the center's neighbors are all leaves, so H is one again,
# and the center scores exactly 2 (n - 1) times a leaf, here 10
centrality_ked(igraph::make_star(6, mode = "undirected"))
```

centrality_kreach	<i>Geodesic K-Path Centrality</i>
-------------------	-----------------------------------

Description

Count of nodes reachable within shortest path distance k. Measures how many nodes a given node can reach quickly.

Usage

```
centrality_kreach(x, mode = "all", k = 3, ...)
```

Arguments

- x Network input (matrix, igraph, network, cograph_network, tna object).
- mode For directed networks: "all" (default), "in", or "out".
- k Maximum path length. Default 3.
- ... Additional arguments passed to [centrality](#) (e.g., weighted, directed, invert_weights).

Value

Named numeric vector of k-reach centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_kreach(adj, k = 2)
```

centrality_lac	<i>Local Average Connectivity (LAC)</i>
----------------	---

Description

Average degree of neighbors within the neighborhood subgraph. Measures how interconnected a node's neighbors are. Proposed by Li et al. (2011) for identifying essential proteins in PPI networks.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of LAC values.

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](#) for computing multiple measures at once, [centrality_dmnc](#) for another neighborhood density measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_lac(adj)
```

centrality_laplacian *Laplacian Centrality*

Description

Energy drop from the graph Laplacian when a node is removed (Qi et al. 2012). Measures a node's importance to the overall network energy.

Usage

```
centrality_laplacian(x, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>...</code>	Additional arguments passed to centrality (e.g., weighted, directed).

Value

Named numeric vector of Laplacian centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_laplacian(adj)
```

centrality_leaderrank *LeaderRank Centrality*

Description

PageRank variant with a ground node connected to all nodes. Requires a directed graph.

Usage

```
centrality_leaderrank(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object). Must be directed.
...	Additional arguments passed to centrality .

Value

Named numeric vector of LeaderRank values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_pagerank](#) for standard PageRank.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_leaderrank(adj)
```

centrality_length_scaled_betweenness

Betweenness and closeness variants that carry a tuning parameter

Description

Four measures that reweight, rescope or re-tune a measure `centrality` already computes. Each is a thin wrapper on `centrality()`.

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>...</code>	Additional arguments passed to <code>centrality</code> .
<code>betweenness_delta</code>	Decay exponent for <code>centrality_delta_betweenness</code> . Default 1.
<code>mode</code>	Direction: "all", "out" or "in".
<code>closeness_delta</code>	Distance exponent for <code>centrality_delta_closeness</code> . Default 1.

Details

`length_scaled_betweenness` (**Borgatti & Everett 2006; Brandes 2008, Algorithm 5**) Betweenness with each separated pair weighted by $1/d(s, t)$, so brokering between nearby nodes counts for more than brokering across the graph.

`delta_betweenness` (**Agneessens, Borgatti & Everett 2017**) Betweenness with the pair weight $(d(s, t) - 1)^{-\delta}$ (`betweenness_delta`, default 1). At $\delta = 0$ it is ordinary betweenness; raising it concentrates the score on locally brokered pairs.

`ego_betweenness` (**Everett & Borgatti 2005**) Betweenness computed inside the node's own ego network rather than the whole graph. A node with fewer than two neighbors scores 0. It is close to, but not a function of, `effective_size`.

`delta_closeness` (**Agneessens, Borgatti & Everett 2017, eq. 2**) $\sum_j d_{ij}^{-\delta} / (n-1)$ (`closeness_delta`, default 1). One exponent spans the closeness family: $\delta = 1$ is harmonic over $n - 1$, $\delta = 2$ is harary over $n - 1$, a large δ approaches degree, and $\delta = 0$ counts the reachable set.

Bounded-distance betweenness, which the Centrality Zoo lists as "k-betweenness", needs no separate measure: it is `centrality(x, measures = "betweenness", cutoff = k)`.

Value

Named numeric vector, one value per node.

References

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_gravity](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_length_scaled_betweenness(adj)
centrality_delta_betweenness(adj, betweenness_delta = 2)
centrality_ego_betweenness(adj)
centrality_delta_closeness(adj, closeness_delta = 2)
```

centrality_leverage	<i>Leverage Centrality</i>
---------------------	----------------------------

Description

Measures a node's influence over its neighbors based on relative degree differences. Positive values indicate the node has more connections than its average neighbor.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of leverage centrality values (range -1 to 1).

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_leverage(adj)
```

centrality_lhc

*Lhc Index***Description**

Wang, Yang, Liu and Ma's Lhc index is a semi-local hybrid: it reads a node's *neighbor information* from degree and its *topological location* from the share of the network's triangles that sit on it, then spreads both over a small ball and collects the result one step out. The influence of a node is $C(v) = \sum_{u \in \Phi(v)} k_u (1 + TP(u)) / d^2(uv)$, a sum over the ball $\Phi(v)$ of radius `lhc_radius` in which each member contributes its degree, inflated by its triangle share, discounted by the square of its distance; and the index itself is $Lhc(v) = \sum_{w \in \tau(v)} C(w)$, the influence summed over the open neighborhood $\tau(v) = N(v)$. The triangle share is $TP(u) = NTS(u) / TNTS$, with $NTS(u)$ the number of triangles containing u and $TNTS = \sum_u NTS(u)$.

Usage

```
centrality_lhc(x, ...)
```

Arguments

`x` Network input accepted by [centrality](#).
`...` Additional arguments to [centrality](#), including `lhc_radius`.

Details

The denominator is $TNTS$, not the number of triangles, and the paper settles it rather than the Zoo. Immediately after defining $TNTS$ the source writes that "the total number of triangle structure exists in the network are $\frac{1}{3} * TNTS$ ", so $TNTS = 3\Delta$ for Δ distinct triangles and TP sums to exactly one over the nodes – it really is a share. Entry 2.221 of the Centrality Zoo transcribes the structure of both equations correctly but names the denominator " Δ , the total number of triangular structures in the network", which read literally is three times too small. The two readings are not related by a monotone transform in general, and they differ substantially: on the Krackhardt kite the paper's reading scores node 1 at 100.15 where the Zoo's literal wording gives 125.45. cograph follows the paper.

`lhc_radius` is the source's own parameter, exposed with the source's default. The paper writes it d , states on page 4 that "the distance ranged d is set to be 2, namely, only the nearest neighbors and the next-nearest neighbors are taken into consideration", and then sweeps it in section 3 over eleven real networks, reporting that "the optimal value of d is about 2-3" and that the correlation

stabilizes beyond 3. It is therefore a genuine modeling knob rather than an implementation detail, and it is exposed with the paper's 2 as the default. At `lhc_radius = 1` the ball collapses to the neighbors and $C(v)$ becomes $\sum_{u \in N(v)} k_u(1 + TP(u))$; a radius at or above the graph's diameter takes in everything reachable and the score stops moving. The domain is a whole number of at least one; anything else is refused with a `cograph_bad_parameter` error.

Both neighborhoods are open, and a node contributes to its own score. $\Phi(v)$ is $1 \leq d(u, v) \leq \text{lhc_radius}$: the focal node is outside it, because $d^2(vv) = 0$ would divide by zero, and unreachable nodes fall outside the radius so no infinity arises. $\tau(v)$ is the open neighborhood. It follows – the paper does not remark on it, but its equations say so – that v does enter its own $Lhc(v)$, since v lies in $\Phi(w)$ at distance 1 for every neighbor w .

Triangle-free graphs are a cograph decision, taken explicitly. Every tree, star, path, even cycle and bipartite graph has $TNTS = 0$, and $TP(u)$ is then $0/0$ everywhere. The source never mentions the case. Since $TNTS$ is a sum of nonnegative counts, it vanishes exactly when every numerator $NTS(u)$ vanishes too, so there is no share to distribute and no node with a claim on one: TP is written as **zero**, and the index reduces to the pure degree-over-squared-distance sum, which is the neighbor and location half of the hybrid with the triangle half contributing nothing. The test is made on $TNTS$ before any division, so no $0/0$ is evaluated; NA or an error would refuse every tree, which the source's own construction handles perfectly well.

Raw scores are not component-local. $TNTS$ is a global sum, so attaching a disconnected component that carries a triangle rescales every TP and moves every score. Attaching a component with no triangle – an isolate included – changes nothing, since it changes no degree, no triangle and no finite distance inside the existing components. An isolate itself scores zero because $\tau(v)$ is empty and equation (2) is an empty sum; a singleton graph and every node of an edgeless graph score zero for the same reason, and an empty graph returns no scores.

Direction, weights, loops and parallel edges are dropped to the simple undirected skeleton the source defines on: k_u is a count, $d(uv)$ a hop count and $NTS(u)$ a combinatorial quantity, and the paper's eleven networks are simple and undirected. There is no in/out/all variant to select, so the measure sits in the no-mode family, and `cutoff` and `invert_weights` are ignored as well. The source states no normalization, so `normalized = TRUE` max-scales the finished vector as elsewhere in [centrality](#).

The source prints no numerical example. There is no toy graph with a table of scores anywhere in the paper – its Table 1 lists network statistics and its figures are aggregate SIR and Kendall plots – so there is no published per-node fixture to reproduce. Verification rests instead on independent reference implementations and on hand-derived closed forms for stars, complete graphs, rings and paths.

Value

Named numeric vector 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](https://doi.org/10.1371/journal.pone.0251208).

See Also

[centrality_hcc](#) and [centrality_ked](#) for other degree-and-position hybrids, [centrality_neighbor_distance](#) for another distance-discounted neighborhood sum, and [list_centralities](#) for the catalogue.

Examples

```
# The path 1-2-3 is triangle-free, so the triangle share drops out and
# the scores are the hand-derived 2, 4.5, 2.
centrality_lhc(igraph::make_graph(c(1, 2, 2, 3), directed = FALSE))

# On a complete graph every node scores (n-1)^3 (n+1) / n; for n = 5
# that is 76.8.
centrality_lhc(igraph::make_full_graph(5))

# Widening the ball can only raise the score, and it stops moving once
# the radius reaches the diameter.
ring <- igraph::make_ring(9)
centrality_lhc(ring, lhc_radius = 1)
centrality_lhc(ring)
centrality_lhc(ring, lhc_radius = 4)
```

centrality_lin

*Lin Centrality***Description**

Reachable nodes squared divided by sum of distances. Well-defined for disconnected graphs.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of Lin centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_closeness](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_lin(adj)
```

centrality_linerank	<i>LineRank centrality</i>
---------------------	----------------------------

Description

Computes PageRank probabilities on the graph whose vertices represent input edges, then aggregates them at the original endpoints. On directed inputs, edge *e* can lead to *f* when the target of *e* is the source of *f*. On undirected inputs, distinct edge states are adjacent when they share an endpoint, following Kosa et al.'s clarification. This uses one state per undirected edge. A pair sharing both endpoints is adjacent once.

Usage

```
centrality_linerank(
  x,
  damping = 0.85,
  linerank_aggregation = "probability",
  ...
)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>damping</code>	Edge-walk continuation probability in [0,1), default 0.85.
<code>linerank_aggregation</code>	Either probability (default) or weight.
<code>...</code>	Additional arguments to centrality .

Details

Line-graph transition weights are products of original edge weights. Row normalization cancels the starting edge weight, so products need not be formed. Uniform teleportation uses probability $1 - \text{damping}$; a dangling edge state also redistributes uniformly. The latter is an explicit cograph PageRank convention because the source does not pin dangling behavior. Damping accepts [0,1), default 0.85; zero is a limit extension.

Default `linerank_aggregation = "probability"` sums stationary edge probabilities, following the definition's prose and the later study. Raw scores then sum to two on a graph with edges. "weight" additionally multiplies each probability by its original edge weight, matching the weighted incidence aggregation in Kang et al.'s Algorithm 2. These conventions differ for weighted inputs

and are not interchangeable. The original pseudocode also has inconsistent row/column normalization; this implementation follows its random-walk definition, corroborated by the later paper, rather than claiming literal pseudocode equivalence.

Retains direction, loops and remaining parallel edges as distinct states. A directed loop can transition to itself. Undirected line graphs exclude self transitions. Both aggregation choices count endpoint incidences, so an original loop contributes twice at its node. These loop conventions are explicit extensions. Generic loops and simplify apply first. Finite nonnegative weights are supported; zero-weight edges are absent. `weighted = FALSE` uses unit edge weights. Generic mode, shortest-path inversion and cutoff do not affect the result. Isolates score zero, edgeless inputs return zeros, and empty inputs return no scores.

The native dense line-graph solve costs $O(m^3)$ time and $O(m^2)$ memory for m retained edges; this is not the authors' distributed large-graph implementation. The measure must be requested explicitly. Unresolvable transition ranges, unstable systems and overflowing raw weighted aggregation raise errors. Maximum normalization supports raw weight overflow by scaling weights first; tiny ratios can underflow.

Value

Named numeric vector 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.

Examples

```
centrality_linerank(igraph::make_ring(4))
```

centrality_inc

Local neighbor contribution centrality

Description

The local neighbor contribution (LNC) of Dai, Wang, Sheng, Sun, Khawaja, Ullah, Dejene and Duan multiplies what a node contributes on its own by what its neighborhood contributes to it: $LNC(i) = d_i^3 (1 - 1/d_i)^{d_i-1} (\sum_{j \in N(i)} d_j) / (n - 1)$, with $0^0 = 1$. The first two factors are the source's *own contribution* $ownCon(i) = d_i(1 - 1/d_i)^{d_i-1}$, the chance that a node picking one neighbor uniformly at random reaches a given one and misses the rest, scaled by its degree; the rest is the *neighbor contribution* $neiCon(i) = d_i^2 \sum_{j \in N(i)} d_j / (n - 1)$, the source's cluster degree weighted by its neighbors' degree centralities.

Usage

```
centrality_lnc(x, ...)
```

Arguments

x Network input accepted by [centrality](#).
 ... Additional arguments to [centrality](#).

Details

The measure takes no parameters. The source calls this out as a feature, "Parameter-Free: LNC does not rely on prior knowledge and parameter adjustments", so none is offered.

Raw scores are not comparable across graphs of different order. The $1/(n-1)$ comes from the degree centrality of equation (1), where n is the vertex count of the whole network, not of the node's component. Adding a disconnected component therefore multiplies every score by $(n-1)/(n'-1)$, leaving the ranking alone and the raw values not.

The source's printed equations do not literally give its printed numbers, and cograph follows the numbers. Equations (4) and (5) both sum a term over $j = 1, \dots, k$, and k is described three incompatible ways: the prose calls it the number of nearest and next nearest neighbors, Algorithm 1 line 12 sets it to the degree, and equation (5) taken literally carries one factor of d_i too many. The printed intermediates $D(v_5) = 12$, $ownCon(v_5) = 1.6875$ and $neiCon(v_5) = 19.2$, together with all eleven Table 1 influences, are reproduced by exactly one pair of factors, the one above: k acts as d_i in (5) and as d_i^2 in (4). The equally literal split that moves one d_i from the neighbor factor to the own factor gives the same product, so the measure itself is unambiguous.

This is not the Centrality Zoo's formula. Zoo section 2.238 writes the own contribution as $d_i |N^{(\leq 2)}(i)| \sum_{j \in N^{(\leq 2)}(i)} (1/d_j)(1 - 1/d_j)^{|N^{(\leq 2)}(i)|-1}$, replacing the focal node's own contribution probability $P(v_i)$ by each neighbor's $P(v_j)$ and the binomial count d_i by the size of the two-hop neighborhood; its neighbor factor is right in form but uses that same two-hop size where the printed numbers need d_i^2 . On the source's own Figure 1 the Zoo reading reproduces none of the eleven printed values and inverts the paper's headline ranking, scoring v_8 32.23 above v_5 28.90 where the paper prints 32.4 for v_5 and 29.7 for v_8 , and lifting the degree-two nodes v_6, v_7 above the degree-three v_9 . cograph implements the paper. No Zoo variant is offered.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and path-weight inversion are ignored. Isolates score zero, and so does the single node of a singleton graph: the source has no value there, since $P(v_i) = 1/0$ and the $n-1$ denominator vanishes, and zero is a cograph extension chosen because d_i^3 and the empty neighbor-degree sum are both zero. Empty graphs return no scores. The source states no normalization; normalized = TRUE max-scales the finished vector as elsewhere in [centrality](#). Nothing overflows: the cubed degree is bounded by n^3 , the neighbor-degree sum by twice the edge count, and the binomial factor lies in $[1/4, 1]$. Cost is one sparse matrix-vector product, $O(n+m)$.

Numerical verification establishes agreement with the source's printed Table 1 and printed intermediates, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](#) and [centrality_neighbor_distance](#) for other neighborhood sums, and [list_centralities](#) for the catalogue.

Examples

```
# Every node of a ring has degree two and a neighbor-degree sum of four
centrality_lnc(igraph::make_ring(6))

# A star: the center carries the whole neighborhood
centrality_lnc(igraph::make_star(5, mode = "undirected"))
```

centrality_load	<i>Load Centrality</i>
-----------------	------------------------

Description

Fraction of all shortest paths passing through a node, similar to betweenness but weighting paths by 1/count (Goh et al. 2001).

Usage

```
centrality_load(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality (e.g., weighted, directed).

Value

Named numeric vector of load centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_betweenness](#) for the standard variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_load(adj)
```

centrality_lobby	<i>Lobby Index (H-Index of Neighborhood)</i>
------------------	--

Description

Largest k such that the node's closed neighborhood contains at least k nodes with degree $\geq k$.
 Network analogue of the h-index.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named integer vector of lobby index values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_lobby(adj)
```

centrality_localized_bridging

Localized bridging centrality from ego betweenness

Description

Nanda and Kotz's localized bridging centrality is the product of a node's unnormalized betweenness in its induced one-hop ego network and its bridging coefficient. The coefficient is reciprocal focal degree divided by the sum of reciprocal neighbor degrees, all measured in the original graph. It is not computed from degrees truncated to the ego network.

Usage

```
centrality_localized_bridging(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality . <code>normalized = TRUE</code> divides final scores by their maximum; all-zero scores remain zero. Ego betweenness is never scaled by ego size.

Details

Each unordered pair of other ego-network vertices contributes the fraction of its shortest paths that pass through the focal vertex. Endpoints are excluded. Uses a simple unweighted undirected skeleton: either arc direction creates an edge, loops are removed and parallel edges count once. Weights, mode, inversion and cutoff are ignored. This projection is an explicit cograph convention, not a directed or weighted generalization of LBC.

Isolates and leaves score zero; the isolate value extends the undefined bridging coefficient by zero. Complete graphs score zero. Disconnected components are evaluated independently before optional maximum scaling. Empty graphs return no scores. The one-hop calculation uses the Everett-Borgatti common-neighbor shortcut in each ego network, with worst-case $O(n^4)$ time and $O(n^2)$ memory for dense matrix multiplication across all nodes.

Value

Named numeric vector 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](https://doi.org/10.1007/9781461408574_7).

See Also

[centrality_extended_local_bridging](#) for two-hop ego networks. [centrality_local_bridging](#) retains the distinct legacy score, inverse degree times bridging coefficient.

Examples

```
centrality_localized_bridging(igraph::make_graph("Zachary"))
```

centrality_local_bridging

Local Bridging Centrality

Description

(1/degree) times bridging coefficient. Local measure of inter-community connectivity. This legacy score differs from Nanda and Kotz's ego-betweenness product; use [centrality_localized_bridging](#) for their LBC definition.

Usage

```
centrality_local_bridging(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of local bridging values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_bridging](#) for the betweenness-weighted variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_local_bridging(adj)
```

centrality_local_dimension

Local Dimension

Description

Growth exponent of the ball around a node (Silva & Costa 2013; Pu et al. 2014). Let $B_i(r)$ be the number of nodes within r hops of i , the node itself included. The local dimension is the slope of $\ln B_i(r)$ on $\ln r$ over $r = 1, \dots, d_{\max}(i)$:

$$D_i = \frac{d \ln B_i(r)}{d \ln r}.$$

A node that reaches most of the network in a few hops has a small exponent, so **lower values mark more influential nodes**. When a node has a single radius (it reaches every other node in one hop) the regression is undefined and the discretized derivative $r n_i(r)/B_i(r)$ at $r = 1$ is reported, where $n_i(r)$ counts the nodes at distance exactly r .

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
...	Additional arguments passed to centrality .

Details

The implementation reproduces the worked example in Wen & Jiang (2019), which reports 0.9231 for ring sizes 4, 5, 4, 4. Distances are hop counts; edge weights are ignored.

Value

Named numeric vector, one value per node. NaN for a node that reaches no other node.

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](#) for the entropy-weighted variant, [centrality_distance_entropy](#).

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_local_dimension(star5)
```

centrality_local_dimension_fixed

Fixed-Radius, Fuzzy and Volume Local Dimensions

Description

Three further members of the local-dimension family, all computed from hop counts (edge weights are ignored) with the center node counted in its own ball, as in [centrality_local_dimension](#).

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 (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
ld_radius	Radius r for local_dimension_fixed, in hops. A single number of at least 1; default 2. Anything else raises a cograph_bad_parameter error.
...	Additional arguments passed to centrality .

Details

local_dimension_fixed (Silva & Costa 2013) The discretized estimator $D_i(r) = r n_i(r) / B_i(r)$ at one radius ld_radius (default 2), where $n_i(r)$ is the ring at distance r and $B_i(r)$ the ball within it. A structural descriptor rather than an importance ranking; nodes with eccentricity below the radius score 0. The paper defines a curve in r and fixes r per figure; the Zoo lists this fixed-radius form separately from Pu et al.'s regression form.

fuzzy_local_dimension (Wen & Jiang 2019) Fuzzy ball $N_i(r) = \sum_{d_{ij} \leq r} e^{-d_{ij}^2/r^2} / |\{j : d_{ij} \leq r\}|$ for $r = 1, \dots, d_{\max}(i)$; the measure is the slope of $\log N_i(r)$ on $\log r$. Larger = more influential. Reproduces Table 1 of the paper (Krackhardt kite) and its karate-club top ten in order.

local_volume_dimension (Li & Deng 2021) Volume $V_i(l) = \sum_{d_{ij} \leq l} k_j$, $l = 1, \dots, ecc(i)$; the measure is the slope of $\ln V_i(l)$ on $\ln l$. Smaller = more important. The article is closed access; the definition follows the authors' own later preprint and the Zoo entry, and no published per-node values exist to check against.

The two regression measures return NaN for a node with fewer than two radii.

Value

Named numeric vector, one value per node.

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](#).

Examples

```
path5 <- matrix(0, 5, 5)
path5[cbind(1:4, 2:5)] <- 1; path5 <- path5 + t(path5)
rownames(path5) <- colnames(path5) <- LETTERS[1:5]
centrality_local_dimension_fixed(path5)
centrality_fuzzy_local_dimension(path5)
centrality_local_volume_dimension(path5)
```

centrality_local_efficiency

Local efficiency, s-core, fragmentation, k-path census and EPC

Description

Five node measures that other centrality packages expose and centrality() did not. Each is a thin wrapper on [centrality](#).

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: matrix, igraph, network, cograph_network, or tna object.
mode	Direction: "all", "out" or "in".
...	Additional arguments passed to centrality .
kpath_len	Maximum path length for centrality_kpath. Default 3.
epc_threshold	Edge removal probability. Default 0.5.
epc_runs	Number of percolation realizations. Default 1000.
epc_seed	Random seed. Default NULL, which leaves the caller's stream alone and makes the estimate vary between calls.

Details

local_efficiency (Latora & Marchiori 2001) The global efficiency of the subgraph induced on the node's neighbors, the node itself removed: the mean of $1/d_{jl}$ over ordered pairs of neighbors, with distances measured inside that subgraph. Nodes with fewer than two neighbors score 0. High values mark a node whose neighborhood survives its loss. Matches `igraph::local_efficiency()` and `brainGraph::efficiency(type = "local")`.

s_core (Eidsaa & Almaas 2013) The weighted k-core: the largest strength threshold s whose maximal subgraph of nodes with strength at least s still contains the node. Unit weights give the k-core number exactly. Uses edge weights.

fragmentation (Borgatti 2006) Distance-weighted fragmentation of the network after deleting the node: $1 - \sum 1/d_{ij}/((n-1)(n-2))$ over the ordered pairs that remain. Higher means a more disruptive removal. Matches `keyplayer::fragment()` on unweighted input.

kpath (Sade 1989) The number of simple paths of length at most `kpath_len` (default 3) that the node lies on, endpoints included; length 1 alone reproduces degree. Matches the per-vertex column sums of `sna::kpath.census()`. Enumeration is exhaustive, so cost grows with branching factor to the power `kpath_len`.

epc (Lin et al. 2008) Edge percolated component: each edge survives with probability $1 - \text{epc_threshold}$, and the score is the mean size of the node's component over `epc_runs` realizations, as a share of the network. `cytoHubba` and `centiserve::epc()` divide by the node count alone, so their number is `epc_runs` times this one; the ranking is the same. A Monte Carlo estimate – pass `epc_seed` for a reproducible value.

`local_efficiency`, `fragmentation` and `kpath` follow `mode`; `s_core` and `epc` read the undirected skeleton.

Value

Named numeric vector, one value per node.

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_weighted_kshell](#), [centrality_geodesic_kpath](#), [network_local_efficiency](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_local_efficiency(adj)
centrality_s_core(adj)
centrality_fragmentation(adj)
centrality_kpath(adj, kpath_len = 2)
centrality_epc(adj, epc_runs = 50, epc_seed = 1)
```

centrality_local_information_dimension

Local Information Dimensionality

Description

Entropy-weighted local dimension (Wen & Deng 2020). With $p_i(l) = B_i(l)/N$ the share of the network inside the box of l hops around i (node included), the box information is $I_i(l) = -p_i(l) \ln p_i(l)$ and

$$D_i^I = -\frac{dI_i(l)}{d \ln l},$$

estimated as minus the least-squares slope of $I_i(l)$ on $\ln l$ for $l = 1, \dots, \lceil d_{\max}(i)/2 \rceil$. **Higher values mark more influential nodes.** When only one box size is available the discretized derivative of the source paper, $l(1 + \ln p_i(l)) n_i(l)/N$, is reported.

Usage

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


Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
<code>...</code>	Additional arguments passed to centrality .

Details

Distances are hop counts; edge weights are ignored.

Value

Named numeric vector, one value per node. NaN for a node that reaches no other node.

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](#).

Examples

```
path5 <- matrix(0, 5, 5)
path5[cbind(1:4, 2:5)] <- 1; path5 <- path5 + t(path5)
rownames(path5) <- colnames(path5) <- LETTERS[1:5]
centrality_local_information_dimension(path5)
```

centrality_malatya *Malatya centrality*

Description

The static Malatya score of a node is the sum of its degree divided by each neighbor's degree: $M(i) = \sum_{j \in N(i)} d_i/d_j$. Computes the score on the original graph. On nonisolated vertices it is exactly the reciprocal of [centrality_bridging_coefficient](#); this relationship follows from their definitions, not rank correlation.

Usage

```
centrality_malatya(x, ...)
```

Arguments

x Network input accepted by [centrality](#).
 ... Additional arguments to [centrality](#). With normalized = TRUE, positive scores are divided by their maximum.

Details

Uses the simple undirected unweighted skeleton: either direction creates an edge, parallel edges count once and self-loops are removed. This is an explicit projection of other inputs to the source's domain. The empty neighbor sum assigns isolates zero. On a regular graph the score equals degree. High scores favor nodes with many neighbors of low degree.

Value

Named numeric vector 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](#).

Examples

```
centrality_malatya(igraph::make_star(5, mode = "undirected"))
```

centrality_map_equation

Map equation centrality with explicit coding and flow conventions

Description

Measures the reduction in codelength when a node is silenced, comparing the original codebook used without that node's codeword to a redesigned codebook. It does not remove the node or recompute the network partition. The score in bits is $-(s-p) \log_2((s-p)/s)$, where p is the node visit rate and s is the rate of use of its module's codebook.

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 gives one module. Unnamed vectors follow input order. Named vectors must match all input node names exactly and are reordered to input order.
map_flow	"unrecorded" (default) for unrecorded link teleportation, or "recorded" for recorded uniform node teleportation.
map_convention	"paper" (default) includes the exit symbol; "infomap" reproduces the visit-only author implementation/table.
...	Additional arguments to centrality , including damping, weighted, simplify, and normalized.

Details

With `map_convention = "paper"`, `s` includes module node visits and module exits, as explicitly defined in Blocker et al. (2022), equations 2 and 9-11. With `"infomap"`, `s` includes node visits only, reproducing Infomap 2.15.1's modular centrality and the paper's Table 1. These two conventions differ when a module has exit flow. The published table does not reproduce the equation's exit-inclusive convention. Both conventions give nonnegative scores; the continuous boundary value is zero if `p` or `s-p` is zero. The Zoo summary uses a different codelength subtraction.

The default unrecorded link-teleportation model teleports proportionally to out-strength, then records only link-following steps and normalizes their total flow to one. On undirected inputs this gives visit rates proportional to strength, independent of damping. Recorded node teleportation uses uniform destinations and records all moves, including teleportation. Both models teleport away from dangling nodes. Damping is the probability of following a link, default 0.85; it must be less than one. Recorded teleportation remains directed even for reciprocal input arcs.

Weights are nonnegative interaction strengths; zero weights are absent. Direction is retained, and undirected edges become reciprocal arcs. Loops are removed. Parallel edges follow centrality's simplify rule; remaining parallel weights sum. Mode, inversion and cutoff are ignored. With no positive edges, unrecorded flow and scores are zero by cograph convention; recorded flow is uniform. Empty and singleton graphs return no scores and zero respectively. Unrecorded isolates score zero; recorded isolates may have positive scores because their teleportation visits are recorded.

The partition is held fixed. NULL means one module containing every node, the paper's one-level case. For a hierarchical partition, supply globally unique leaf-module labels: silencing affects only that leaf codebook, so higher levels cancel in the score difference. This function does not run community detection or claim that a supplied partition is optimal.

Dense flow calculation takes $O(n^3)$ time and $O(n^2)$ memory; unrecorded undirected flow takes $O(n^2)$. Extreme weight ranges or numerically singular flow solves raise errors. Optional maximum scaling changes the raw bit units; tiny relative scores may underflow.

Value

Named numeric vector 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](https://doi.org/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](https://doi.org/10.1103/PhysRevE.85.056107).

Examples

```
g <- igraph::make_graph("Zachary")
centrality_map_equation(g)
centrality_map_equation(g, membership = rep(1:2, each = 17),
                        map_convention = "infomap")
```

centrality_markov	<i>Markov Centrality</i>
-------------------	--------------------------

Description

Inverse of column means of the mean first passage time matrix. Requires a connected graph.

Usage

```
centrality_markov(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of Markov centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_markov(adj)
```

centrality_mcc	<i>Maximal clique centrality</i>
----------------	----------------------------------

Description

For every maximal clique C containing a vertex, add $(|C| - 1)!$. Only maximal cliques count: a clique contained in a larger clique is excluded. This is Chin et al.'s MCC, not a count of all cliques.

Usage

```
centrality_mcc(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive scores are divided by their maximum.

Details

Uses the simple undirected, unweighted skeleton: either direction creates an edge, parallel edges count once, and self-loops are removed. Singleton cliques are excluded, so isolates score zero. This is an explicit cograph convention consistent with the paper's degree reduction when neighbors have no edges between them. Reading the printed sum literally with singleton cliques would instead assign isolates $0! = 1$.

Maximal clique enumeration has exponential worst-case cost. MCC is held back from `centrality(type = "all")`; select it explicitly or use `include = "mcc"`. Scores use double precision; overflow raises an error, including any clique with more than 171 vertices. Normalization happens after raw calculation and does not bypass this limit.

Value

Named numeric vector 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](#), [list_centralities](#).

Examples

```
centrality_mcc(igraph::make_full_graph(5))
```

centrality_mcgmm

*Multi-characteristics gravity model***Description**

Li and Huang's MCGM combines degree k , core number s and eigenvector centrality x in node masses. Write K , S and X for these features divided by their respective global maxima. Equations 17 and 18 define $\alpha = \max\{\text{median}(K), \text{median}(X)\} / \text{median}(S)$, $m_i = K_i + \alpha S_i + X_i$, and $MCGM_i = \sum_{j: 0 < d(i,j) \leq R} m_i m_j / d(i,j)^2$. The default radius two is the paper's recommended practical setting. All features refer to the original graph, not each node's neighborhood.

Usage

```
centrality_mcgmm(x, mcgm_radius = 2, mcgm_alpha = NULL, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>mcgm_radius</code>	Nonnegative hop-distance cutoff, default two. NULL or infinity includes every reachable partner. Fractional cutoffs include exactly the integer hop distances not exceeding them.
<code>mcgm_alpha</code>	NULL uses the published median-based coefficient. A finite nonnegative scalar explicitly overrides it.
<code>...</code>	Additional arguments to centrality .

Details

The source domain is simple undirected unweighted graphs. Other inputs use their simple undirected skeleton: either arc creates one edge, parallel edges count once and loops are removed. Weights, mode, cutoff, gravity_mass, gravity_radius and path-weight inversion are ignored. These input projections are cograph conventions.

On connected graphs with edges, X is the unique positive Perron vector, scaled to maximum one. For disconnected graphs the paper does not specify an eigenvector selection. This implementation projects the all-ones vector onto the global dominant eigenspace and then scales to maximum one. Equivalently, it selects the limit of identity-shifted power iteration initialized uniformly. Components below the largest spectral radius have eigenvector feature zero; tied components share the projection. Component roots within 64 times machine epsilon times n times $\max(1, \text{spectral radius})$ are treated as tied. All feature maxima and medians remain global. Adding a disconnected component can change scores.

When edges exist but median coreness is zero, the source's automatic alpha is undefined and an error requests an explicit `mcgm_alpha`. This override is an extension of the published adaptive rule; setting it to one recovers equation 16. It is never silently inferred from a different subset of nodes. Isolates score zero when the mass rule is defined. Edgeless graphs and radii below one return zero by an explicit empty-interaction convention, including a singleton; empty graphs return no scores. NULL or infinite radius includes all reachable partners.

Raw scores preserve equation 18's scale. Optional maximum normalization occurs after all gravity contributions and can handle very large explicit alpha values whose raw scores overflow. Dense spectral calculations and all-pairs distances require $O(n^3)$ time and $O(n^2)$ memory. Unresolved positive eigenvectors or overflowing raw scores raise errors. The published nine-node numerical example is reproduced at its printed precision. This establishes numerical agreement, not a universal guarantee of spreading prediction or parity with unreleased author software.

Value

Named numeric vector 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](https://doi.org/10.1038/s41598022140053).

Examples

```
centrality_mcgmg(igraph::make_ring(6))
centrality_mcgmg(igraph::make_star(6), mcgmg_radius = 3)
```

centrality_mixed_gravity

Mixed gravitational centrality

Description

Mixed gravitational centrality (MGC), also called improved gravitational centrality (IGC), uses the focal node's core number as its mass and the partner node's degree as its mass: $MGC_i = k_s(i) \sum_{j: 0 < d(i,j) \leq r} k(j) / d(i,j)^2$. All degrees, core numbers and hop distances are measured on the original simple undirected graph. The masses are asymmetric even though distances are symmetric. This differs from using core numbers on both ends or degree on both ends of each interaction.

Usage

```
centrality_mixed_gravity(x, gravity_radius = 3, ...)
```

Arguments

x	Network input accepted by centrality .
gravity_radius	Nonnegative hop-distance cutoff, default three. NULL or infinity includes every reachable partner. Fractional cutoffs include exactly integer hop distances not exceeding them; values below one give zero. The optional "auto" is a cograph heuristic: round half the mean finite positive distance to the nearest integer (ties to even), with minimum one. It is not the cited radius rule and can change when disconnected components are added.
...	Additional arguments to centrality .

Details

The implementation follows the explicit reproduction of Wang et al.'s method in Li and Huang (2022), equations 5-8, with default radius three. The original 2018 full equations and author software have not been inspected. The Zoo summary writes an immediate-neighbor inner sum; gravity_radius = 1 reproduces that literal interpretation. Numerical verification establishes agreement with the cited reproduced definition, not parity with unavailable original software or a guarantee of spreading performance.

Uses the simple undirected unweighted skeleton: either arc creates an edge, parallel edges count once and loops are removed. This projection is a cograph convention outside the source domain. Edge weights, mode, cutoff, gravity_mass and path-weight inversion are ignored. Isolates and singleton graphs score zero; empty graphs return no scores. Unreachable partners contribute zero. With a fixed radius, adding a disconnected component leaves existing raw scores unchanged. Optional maximum normalization applies to the complete result over all nodes. Dense all-pairs distances cost $O(n^3)$ time and $O(n^2)$ memory.

Value

Named numeric vector 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](#).

Examples

```
centrality_mixed_gravity(igraph::make_ring(6))
centrality_mixed_gravity(igraph::make_star(6), gravity_radius = 1)
```

centrality_mnc	<i>Maximum Neighborhood Component (MNC)</i>
----------------	---

Description

Size of the largest connected component in the node's neighborhood subgraph.

Usage

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


Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named integer vector of MNC values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_dmnc](#) for the density variant.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_mnc(adj)
```

centrality_modified_expected_force

Modified Expected Force centrality

Description

Multiplies the two-event Expected Force by the logarithm of alpha times seed degree (Lawyer 2015, equation 2). Alpha defaults to two, as in the paper, and must be finite and strictly greater than one. Directed input uses outgoing degree, consistent with the outgoing transmission process. An isolate scores zero without evaluating the logarithm of zero. All graph, event-counting and zero-force conventions of [centrality_expected_force](#) apply. Native log addition avoids overflow when alpha times degree cannot be represented.

Usage

```
centrality_modified_expected_force(x, exf_alpha = 2, ...)
```

Arguments

x	Network input accepted by centrality .
exf_alpha	Degree rescaling factor, default two, finite and greater than one. The paper motivates small values; larger finite values are permitted by the formula without a predictive-performance claim.
...	Additional arguments to centrality . normalized = TRUE divides by the maximum score; all-zero results remain zero.

Value

Named numeric vector 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](https://doi.org/10.1038/srep08665).

Examples

```
centrality_modified_expected_force(igraph::make_graph("Zachary"))
```

centrality_modularity_vitality
Modularity Vitality

Description

Contribution of a node to the modularity of a fixed partition (Magelinski, Bartulovic & Carley 2021):

$$V_Q(i) = Q(G, C) - Q(G - i, C \setminus \{i\}),$$

the drop in Newman modularity when node i is deleted and the remaining nodes keep their communities. Positive values mark community hubs (removing them weakens the modular structure); negative values mark bridges (removing them sharpens it). Weighted graphs use edge weights; directed graphs use the Leicht-Newman directed modularity, as igraph does.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>membership</code>	Community labels, one per node (integer, factor, or character). Required; without it the function warns and returns NA. Obtain one from detect_communities .
<code>...</code>	Additional arguments passed to centrality .

Details

All n vitalities are computed in closed form from one matrix product, without recomputing modularity n times.

Value

Named numeric vector, one value per node. NaN where deleting the node leaves a graph with no edges.

Conditions

Raises an error of class `cograph_bad_membership` when membership is not one non-missing label per node.

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](#), [detect_communities](#).

Examples

```
# Two triangles joined by one bridge edge (C -- D)
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_modularity_vitality(adj, membership = c(1, 1, 1, 2, 2, 2))
```

centrality_ncvoterank *NCVoteRank*

Description

Kumar and Panda's (2020) neighborhood-coreness VoteRank. As in VoteRank, every node votes for its neighbors with its voting ability, the top scorer is elected, and the abilities around it are weakened; here each voter's ability is additionally weighted by its neighborhood coreness,

$$s_u = \sum_{v \in N(u)} va_v [\theta + (1 - \theta) nc_v], \quad nc_v = \frac{\sum_{w \in N(v)} ks(w)}{\max_j \sum_{w \in N(j)} ks(w)},$$

with ks the k-shell index (Bae & Kim 2014) and $\theta = 0.5$. After an election the winner's ability drops to 0, its neighbors lose $1/\langle k \rangle$ and the nodes two steps away lose $1/(2\langle k \rangle)$. Elections continue until every node is placed, as in [centrality_voterank](#); the first elected scores 1, the last $1/n$.

Usage

```
centrality_ncvoterank(x, ncvote_theta = 0.5, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>ncvote_theta</code>	Weight θ of the plain vote. Default 0.5.
<code>...</code>	Additional arguments passed to centrality .

Details

Provenance. The original Physica A article could not be obtained; this definition follows the Centrality Zoo encyclopedia (Shvydun 2025) and three independent restatements (Yu et al. 2020, Li et al. 2022, Zhu et al. 2023), which agree on the voter-side coreness weighting. The scaling of the coreness term by its maximum follows Yu et al., who state the coreness is normalized without giving the form. With $\theta = 1$ and no two-hop weakening the procedure is exactly VoteRank, which is reproduced against `networkx.voterank`. Defined for undirected graphs; direction, weights and loops are ignored.

Value

Named numeric vector in $(0, 1]$, one score per node.

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](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_ncvoterank(adj)
```

centrality_neighborhood_connectivity
Neighborhood Connectivity

Description

Mean degree of a node's neighbors (Maslov & Sneppen 2002), the "average neighbor degree" reported by Cytoscape:

$$C_{NC}(i) = \frac{1}{k_i} \sum_{j \in N(i)} k_j.$$

High values mark nodes attached to hubs. Isolates score 0. Under mode = "out" the out-neighbors' out-degrees are averaged, under "in" the in-neighbors' in-degrees.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
...	Additional arguments passed to centrality .

Value

Named numeric vector, one value per node.

References

Maslov, S., & Sneppen, K. (2002). Specificity and stability in topology of protein networks. *Science*, 296(5569), 910-913.

See Also

[centrality_degree](#), and `igraph::knn()` for the Barrat weighted generalization.

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_neighborhood_connectivity(star5)
```

centrality_neighbor_distance

Neighborhood centrality, and its neighbor distance special case

Description

Neighborhood centrality adds to a node's own benchmark centrality the benchmark centrality of the nodes its walks reach, discounted once per step: $C_i^n(\theta) = \theta_i + a \sum_{j \in \Gamma_i} \theta_j + a^2 \sum_{l \in \Gamma_j \setminus i} \theta_l + \dots + a^n \sum_{s \in \Gamma_{s-1} \setminus x} \theta_s$. The sums are nested and each level excludes only the node the walk just came from, so the k -th term sums θ over the endpoints of the **non-backtracking walks of length k** that start at i , once per walk. A walk may revisit a node it passed earlier, including i itself; only immediate backtracking is barred. The Zoo calls the setting `nd_mass = "degree"`, `nd_order = 2`, `nd_decay = 0.2` the *neighbor distance centrality*, and that is the default here; it is the configuration the source recommends.

Usage

```
centrality_neighbor_distance(
    x,
    nd_order = 2,
    nd_decay = 0.2,
    nd_mass = "degree",
    ...
)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>nd_order</code>	Number of steps n , a single nonnegative whole number; default two, the source's recommended setting. The source studies one to four steps. Zero returns the benchmark centrality.
<code>nd_decay</code>	Per-step decay a , a single finite number; default 0.2, the source's own value. The source's domain is $[0, 1]$.
<code>nd_mass</code>	Benchmark centrality θ : "degree" (default) or "coreness". These are the two the source uses.
<code>...</code>	Additional arguments to centrality .

Details

This is not the same as summing over distance shells. The Centrality Zoo (section 2.279, equation 2.1) paraphrases the measure with sums over $N^{(k)}(i)$, "the set of k -hop neighbors", which visits each node at most once per level and never revisits a closer one. The two readings agree on trees and disagree on any graph carrying a triangle or a cycle of length at most $2n$, and the difference is a per-node offset, not a rescaling. On the triangle-plus-pendant A-B, A-C, B-C, A-D with the defaults, the walk sums of the source give 4.16, 3.24, 3.24, 1.76 while distance shells would give 4.00, 3.04, 3.04, 1.76. cograph implements the source equation. No shell variant is offered: the shell form appears only in a secondary paraphrase, which also attributes the measure to a different paper whose text does not contain it.

The source states no normalization, so raw scores grow with `nd_decay` and `nd_order`; `normalized = TRUE` max-scales the finished vector and is a cograph convention. `nd_decay` is $a \in [0, 1]$ in the source, which sweeps 0.1 to 0.5; cograph accepts any finite value, and a negative or larger one leaves the source's domain. `nd_order = 0` drops every sum and returns θ itself, which is what the source says $a = 0$ does.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once, and loops are removed, since a loop would make "the node the walk just came from" ambiguous. Edge weights, mode, cutoff and path-weight inversion are ignored. Isolates have every sum empty and score θ_i , which is zero for both benchmarks; walks never leave a component, so the raw score of a node is unchanged by adding a disconnected component. Empty graphs return no scores. Core numbers follow [centrality](#)'s "coreness", so an isolate sits in the zero-shell. Cost is `nd_order` dense matrix-vector products, $O(n^2)$ each. Walk counts grow geometrically in `nd_order`, so a large order overflows to infinity; the source considers one to four steps.

Numerical verification establishes agreement with the source equation as printed in the author preprint, not parity with author software, which does not exist, and not any claim about spreading performance.

Value

Named numeric vector 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](https://doi.org/10.1016/j.physa.2016.02.028).

See Also

[centrality_semi-local](#) and [centrality_extended_coreness](#) for other neighborhood sums, and [list_centralities](#) for the catalogue.

Examples

```
# Neighbor distance centrality: degree benchmark, two steps, a = 0.2
centrality_neighbor_distance(igraph::make_ring(6))

# The source's other benchmark, and a wider neighborhood
centrality_neighbor_distance(igraph::make_star(7, mode = "undirected"),
                             nd_order = 3, nd_mass = "coreness")
```

centrality_ninl	<i>Node and Neighbor Layer Information centrality</i>
-----------------	---

Description

Zhu and Wang's NINL initializes each node with the sum of original-graph degrees in its closed radius- r neighborhood. The paper sets r to the ceiling of the graph's average shortest-path length. Each iteration then replaces every node's score by the sum of its neighbors' previous scores: $NINL-p = A^p NINL-0$. The paper uses $p = 3$; zero iterations returns the initial degree volume. Repeated vertices and edges in these walks count.

Usage

```
centrality_ninl(x, ninl_order = 3, ninl_radius = NULL, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>ninl_order</code>	Nonnegative integer iteration count, default 3. At most $2^{53} - 1$, the consecutive-integer precision of doubles.
<code>ninl_radius</code>	NULL for the source-defined automatic radius, or a nonnegative integer hop radius, or Inf for all reachable nodes. Radius zero uses the focal node's degree alone.
<code>...</code>	Additional arguments to centrality . <code>normalized = TRUE</code> divides by the maximum score; all-zero scores remain zero. Normalization is optional and is not part of the raw definition in the original paper.

Details

Uses simple undirected unweighted topology: either arc creates an edge; loops and parallel edges are removed. Weights, mode, inversion and cutoff are ignored. This does not claim a directed or weighted NINL definition.

The mean path length includes all distinct vertex pairs. For disconnected graphs it is infinite, so the automatic radius includes every reachable node in each component. This is an explicit cograph extension of the paper's connected example; unreachable nodes never enter the degree sum. Isolates score zero and empty graphs return no scores. A supplied radius is an explicit generalization of the paper's automatic-radius rule.

Stepwise propagation evaluates the requested finite iteration count, without assuming convergence to eigenvector centrality. Exact repeated floating-point states of period one or two allow the remaining iterations to be skipped while preserving parity. No tolerance-based convergence cutoff is used. Normalized scores can alternate on bipartite graphs. Dense distance calculation and propagation take $O(n^3 + pn^2)$ time and $O(n^2)$ memory; very large orders can be slow if no exact repeated state occurs. Raw overflow raises an error. With maximum normalization, global rescaling after every step avoids overflow; extremely small relative scores can still underflow in double precision.

Value

Named numeric vector 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](https://doi.org/10.3390/sym13091570).

Examples

```
centrality_ninl(igraph::make_graph("Zachary"))
centrality_ninl(igraph::make_star(5, mode = "undirected"), ninl_order = 2)
```

centrality_node_contraction

Node Contraction Centrality (IMC and IIMC)

Description

Tan, Wu and Deng's (2006) node-contraction importance, as restated by Wang et al. (2011). The agglomeration (cohesion) of a graph is $\partial(G) = 1/(N\bar{L})$, with $\bar{L}$ the mean shortest-path length over ordered pairs; contracting a node merges it with all its neighbors into one node, and

$$IMC(v) = 1 - \partial(G)/\partial(G_v).$$

The improved form (node_contraction_improved) adds the same score of the node's edges computed on the line graph: $IIMC(v) = \alpha IMC(v) + \beta \sum_{e \ni v} IMC_{L(G)}(e)$, with $\alpha/\beta = 5$ (contraction_rho) and $\alpha + \beta = 1$, the normalization that reproduces the paper's Table 1. Higher = more important. Both reproduce Table 1 of Wang et al. (2011). The Zoo entry describes the contracted graph as the graph with the node removed; the sources define it by contraction, which is what is implemented.

Usage

```
centrality_node_contraction(x, ...)
```

```
centrality_node_contraction_improved(x, contraction_rho = 5, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .
contraction_rho	Ratio α/β for the improved form. Default 5.

Details

On a disconnected graph the mean path length is taken over the mutually reachable ordered pairs (a cograph choice; the sources assume connected graphs). Direction, weights and loops are ignored. Cost is one all-pairs computation per node, so $O(n^2(n+m))$; the improved form does the same on the line graph, $O(m^2(m+m'))$.

Value

Named numeric vector, one value per node.

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](#).

Examples

```
path5 <- matrix(0, 5, 5)
path5[cbind(1:4, 2:5)] <- 1; path5 <- path5 + t(path5)
rownames(path5) <- colnames(path5) <- LETTERS[1:5]
centrality_node_contraction(path5)
centrality_node_contraction_improved(path5)
```

centrality_pagerank	<i>PageRank Centrality</i>
---------------------	----------------------------

Description

Random walk centrality measuring node importance. Simulates a random walker that follows edges with probability damping and jumps to a random node with probability 1 - damping.

Usage

```
centrality_pagerank(x, damping = 0.85, personalized = NULL, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
damping	Damping factor (probability of following an edge). Default 0.85.
personalized	Named numeric vector for personalized PageRank. Values should sum to 1. Default NULL (uniform).
...	Additional arguments passed to centrality (e.g., weighted, directed).

Value

Named numeric vector of PageRank values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_eigenvector](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_pagerank(adj)
centrality_pagerank(adj, damping = 0.9)
```

centrality_pairwisedis

*Pairwise Disconnectivity (Potapov et al. 2008)***Description**

For a directed network, `pairwisedis(v)` is the fraction of ordered reachable pairs (s, t) that become unreachable when node v is removed:

$$PD(v) = (|P(G)| - |P(G - v)|) / |P(G)|$$

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` Directed network input (matrix, igraph, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#).

Details

Bit-exact match against `centiserve::pairwisedis` on directed graphs. Requires the input to be directed; returns NA with a warning on undirected inputs.

Value

Named numeric vector of pairwise disconnectivity values in $[0, 1]$.

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](https://doi.org/10.1186/147121059227).

See Also

[centrality](#), [robustness](#).

Examples

```
adj <- matrix(c(0,1,0, 0,0,1, 1,0,0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_pairwisedis(adj)
```

centrality_participation

Participation Coefficient

Description

Measures diversity of inter-community connections. Nodes connecting to many communities have high participation. Requires community membership.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
membership	Integer vector of community assignments (one per node).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality .

Value

Named numeric vector of participation coefficient values (0-1).

See Also

[centrality](#) for computing multiple measures at once, [centrality_within_module_z](#) for within-community connectivity.

Examples

```
adj <- matrix(c(0,1,1,0,0, 1,0,1,0,0, 1,1,0,1,0, 0,0,1,0,1, 0,0,0,1,0), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
centrality_participation(adj, membership = c(1, 1, 1, 2, 2))
```

centrality_percolation

Percolation Centrality

Description

Importance for spreading processes using node states. Each node has a state (0-1) representing how activated it is. When all states are equal, equivalent to betweenness.

Usage

```
centrality_percolation(x, states = NULL, ...)
```

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
states Named numeric vector of node states (0-1). Default NULL (all nodes get state 1).
... Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of percolation centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_betweenness](#) which this generalizes.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_percolation(adj)
centrality_percolation(adj, states = c(A = 0.8, B = 0.2, C = 0.5))
```

centrality_power

Bonacich Power Centrality

Description

Measures influence based on connections to other influential nodes. The power parameter controls whether connections to well-connected nodes increase or decrease centrality.

Usage

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

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
mode For directed networks: "all" (default), "in", or "out".
... Additional arguments passed to [centrality](#) (e.g., normalized, weighted, directed).

Value

Named numeric vector of power centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_eigenvector](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_power(adj)
```

centrality_prestige_domain

Domain Prestige

Description

Directed-graph prestige measure: for each node v , the number of other nodes that can reach v via a directed path.

$$\text{domain}(v) = |\{u \neq v : u \rightarrow^* v\}|$$

Usage

```
centrality_prestige_domain(x, ...)
```

Arguments

<code>x</code>	Directed network input (matrix, igraph, cograph_network, tna object).
<code>...</code>	Additional arguments passed to centrality .

Details

Bit-exact match against `sna::prestige(cmode = "domain")`. Directed-only; returns NA with a warning on undirected input.

Value

Named numeric vector of domain prestige values in $\{0, 1, \dots, N - 1\}$.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

[centrality](#), [centrality_reaching_local](#) for the dual "out-reachability" measure, [centrality_pairwisedis](#) for a related reachability-based directed measure.

Examples

```
# Directed 3-cycle: every node reaches every other node
adj <- matrix(c(0,1,0, 0,0,1, 1,0,0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_prestige_domain(adj)
```

```
centrality_prestige_domain_proximity
      Domain Proximity Prestige
```

Description

Distance-weighted variant of domain prestige. For each directed node v :

$$PD(v) = R_v^2 / (D_v \cdot (n - 1))$$

where R_v is the number of other nodes that reach v , and D_v is the sum of geodesic distances from those reachers to v . A node that is reachable quickly from many others scores high; unreachable nodes score 0.

Usage

```
centrality_prestige_domain_proximity(x, ...)
```

Arguments

x Directed network input (matrix, igraph, cograph_network, tna object).
... Additional arguments passed to [centrality](#).

Details

Bit-exact match against `sna::prestige(cmode = "domain.proximity")` on strongly connected directed graphs. Directed-only; returns NA with a warning on undirected input.

Value

Named numeric vector of domain proximity prestige values in $[0, 1]$.

Divergence from sna on disconnected graphs

`sna`'s formula computes $(\text{counts} > 0) * \text{gdist}$ element-wise and then sums to get the denominator. For any pair where $\text{gdist} = \text{Inf}$ (unreachable), R evaluates $\text{FALSE} * \text{Inf} = \text{NaN}$, so the entire denominator becomes NaN and `sna` zeros every node via `p[is.nan(p)] <- 0`. `cograph` masks with `is.finite()` before summing, producing mathematically correct values on any directed graph, including those with disconnected components.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

[centrality](#), [centrality_prestige_domain](#) for the unweighted count, [centrality_reaching_local](#) for the dual out-reachability measure.

Examples

```
# Directed 3-cycle: each node is reached by both others at distance 1 and 2
adj <- matrix(c(0,1,0, 0,0,1, 1,0,0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_prestige_domain_proximity(adj)
```

centrality_proximal_betweenness

Proximal betweenness centrality

Description

Fractions of shortest paths on which a node is the first or last intermediate vertex, following Brandes (2008), section 3.2, Algorithm 3. Paths have unit edge lengths. Each reachable ordered source-destination pair contributes equally, divided among all its shortest paths.

Usage

```
centrality_proximal_betweenness(x, proximal_variant = "source", ...)
```

Arguments

x	Network input accepted by centrality .
proximal_variant	One of "source" (default), "target", "sum", or "union".
...	Additional arguments to centrality . normalized = TRUE divides by the maximum score; all-zero results remain zero.

Details

The original terminology calls the last intermediate vertex the proximal source (a proxy interacting directly with the destination), and the first intermediate vertex the proximal target. The source variant is the default. Endpoints are excluded, so paths with fewer than two edges contribute nothing. The sum variant counts both roles; the union variant counts a vertex only once when a two-edge path places it in both roles. These are the two combination options in the paper.

Raw scores sum over ordered pairs, including on undirected graphs, following the displayed definition and Algorithm 3. They are not halved. Source and target scores agree on undirected graphs;

sum is twice either score, whereas union removes the two-edge overlap. This convention is distinct from the usual unordered-pair scaling of undirected betweenness.

Uses the simple unweighted graph, retaining edge direction. Loops are removed and repeated edges count once after generic input processing. Weights, mode, inversion and cutoff do not affect this measure. Weighted shortest paths and edge-distinct multigraph paths are outside this implementation's verified domain. Unreachable pairs, isolates and complete graphs contribute zero; empty graphs return no scores.

Native breadth-first searches and dependency accumulation take $O(n(n+m))$ time after the current $O(n^2)$ dense graph preparation. Path counts use double precision; a nonfinite count raises an error instead of returning invalid fractions. Counts above the exact-integer range can be rounded, so numerical equivalence is tolerance-based.

Value

Named numeric vector 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](https://doi.org/10.1016/j.socnet.2007.11.001).

Examples

```
centrality_proximal_betweenness(igraph::make_graph("Zachary"))
centrality_proximal_betweenness(igraph::make_ring(5),
                                proximal_variant = "union")
```

centrality_radiality *Radiality Centrality*

Description

Centrality based on sum of (diameter + 1 - distance) normalized by $n-1$. Nodes closer to others (on average) have higher radiality.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of radiality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_closeness](#) for a related measure.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_radiality(adj)
```

centrality_random_walk

Random Walk Centrality

Description

Inverse sum of random walk distances. Requires a connected graph.

Usage

```
centrality_random_walk(x, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>...</code>	Additional arguments passed to centrality .

Value

Named numeric vector of random walk centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_random_walk(adj)
```

centrality_random_walk_decay

Random walk decay centrality

Description

Was, Rahwan and Skibski's random walk decay centrality sums discounted first-arrival probabilities: $RWD_v = \sum_u b_u E_u[a^{T_v}; T_v < \infty]$, where T_v is the first time the walk reaches v and a is `rwd_decay`. The walk follows outgoing edges in proportion to their nonnegative weights. Sinks lead to a terminal state outside the graph; there is no restart or redistribution of their probability. The start counts as an arrival at time zero, so each node contributes its own starting weight b to its score. Later returns to the same target make no additional contribution.

Usage

```
centrality_random_walk_decay(x, rwd_decay = 0.5, rwd_node_weights = NULL, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>rwd_decay</code>	Finite discount factor in $[0,1)$, default 0.5.
<code>rwd_node_weights</code>	Nonnegative finite starting weights, one per input node. NULL means ones. Unnamed vectors follow input node order; named vectors must match every node name exactly and are reordered.
<code>...</code>	Additional arguments to centrality .

Details

The paper defines a in $(0,1)$. The default 0.5 is an explicit cograph choice; zero is supported as the continuous limit, returning b . `rwd_node_weights = NULL` sets all starting weights to one. These weights are not normalized into a probability distribution in the final score. All-zero starting weights return zero by linear extension. Isolates score their own starting weight; empty input returns no scores. Disconnected components are independent before optional normalization.

Retains input direction and loops. Undirected edges become opposite transitions; an undirected self-loop is one stay transition. Remaining parallel edges contribute their combined weight, or their multiplicity when `weighted = FALSE`. Generic `simplify` is applied first; use `simplify = FALSE` to preserve unweighted parallel multiplicity. Use `loops = FALSE` to remove loops explicitly. Node weights and edge weights are distinct. `weighted = FALSE` ignores edge weights but retains supplied node weights. Generic mode, shortest-path inversion and cutoff do not affect this measure.

Removing any target's outgoing edges cannot change its own raw score: those edges can only be traversed after first arrival. Other nodes' scores may change. Global maximum normalization need not preserve this property. The published Example 3 has internally inconsistent numerical values; the implementation follows Definition 1, equation 6. Independent first-arrival calculations and the separate Example 4 and 5 tables verify it.

Native absorbing systems are solved separately for each target, using only vertices that can reach it. Worst-case runtime is $O(n^4)$ with $O(n^2)$ memory, so this measure must be requested explicitly. Row scaling avoids overflow of total outgoing weights. Unresolvable transition ranges, unstable solves and raw score overflow raise errors. If a first-arrival probability underflows, a forward-mass solve and log-space incoming flux recover its contribution where representable. Unrepresentably small final contributions can still underflow to zero. `normalized = TRUE` supports overflowing raw mass sums by scaling starting weights first; tiny normalized contributions may underflow.

Value

Named numeric vector 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](https://doi.org/10.1609/aaai.v33i01.33012197).

Examples

```
centrality_random_walk_decay(igraph::make_ring(4), rwd_decay = 0.8)
```

centrality_reaching_local

Local Reaching Centrality (Mones, Vicsek & Vicsek 2012)

Description

Local reaching centrality measures how much of the network is reachable from a node.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality .

Details

- Directed unweighted: $LRC(v) = |\{u : u \neq v, v \rightarrow u\}| / (N - 1)$.
- Undirected unweighted: average of $1/d(v, u)$ over all $u \neq v$, divided by $N - 1$. Numerically equal to `igraph::harmonic_centrality(normalized = TRUE)`.
- Weighted: NetworkX convention, where edge weights are interpreted as strengths and path length is $\sum_e (\text{total_weight} / w_e)$. Per-path score is the mean of original edge weights along the shortest path.

Bit-exact match against `networkx.local_reaching_centrality` across all three branches. Bit-exact match against `igraph::harmonic_centrality(normalized = TRUE)` for the undirected unweighted branch. See [reaching_global](#) for the graph-level hierarchy measure derived from per-node LRC.

Value

Named numeric vector of local reaching centrality values.

References

Mones, E., Vicsek, L., & Vicsek, T. (2012). Hierarchy measure for complex networks. *PLoS ONE*, 7(3), e33799.

See Also

[centrality](#), [centrality_harmonic](#), [reaching_global](#).

Examples

```
# Directed path A -> B -> C
adj <- matrix(c(0,1,0, 0,0,1, 0,0,0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_reaching_local(adj, mode = "out")
```

centrality_relative_entropy

Relative-Entropy Integrated Evaluation

Description

Integrates several centrality indexes into one score without asking the user to weight them. Each index is first turned into a discrete distribution over the nodes, and the integrated score is the distribution that has the smallest total relative entropy to all of them. Chen, Wang and Luo (2016) show that the minimizer has a closed form, equation (11): $w_i = \prod_{j=1}^m u_{ji}^{1/m} / \sum_i \prod_{j=1}^m u_{ji}^{1/m}$, the normalized geometric mean of the m index distributions. The result sums to one, so it reads as a share of importance rather than a raw score.

Usage

```
centrality_relative_entropy(
  x,
  re_indexes = c("degree", "closeness", "betweenness", "constraint"),
  re_negative = NULL,
  ...
)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>re_indexes</code>	Character vector of constituent indexes, in any order, without repeats. Default is the source's four-index distinctiveness set; see the Constituent indexes section for the full vocabulary.
<code>re_negative</code>	Character vector naming which of <code>re_indexes</code> are negative, that is, mapped by equation (9). Default NULL uses the source's own declarations, which make <code>constraint</code> and <code>largest_component</code> negative and everything else positive. Pass character(<code>0</code>) to treat every requested index as positive.
<code>...</code>	Additional arguments to centrality .

Details

A *positive* index, where a larger value marks the more important node, becomes a distribution through equation (8), $C'(i) = C(i) / \sum_j C(j)$. A *negative* index, where a smaller value marks the more important node, becomes one through equation (9), $C'(i) = (1 - C(i) / \sum_j C(j)) / \sum_k (1 - C(k) / \sum_j C(j))$. Both maps are invariant to rescaling a positive index, so only the shape of an index matters, never its units.

The geometric mean is unforgiving: a node that scores exactly zero on any one index scores exactly zero overall. That is the source's own printed behavior – its Table 2 gives zero to the three Kite nodes with zero betweenness – and `cograph` reproduces it rather than smoothing it away.

Value

Named numeric vector in input node order, summing to one.

Constituent indexes

`re_indexes` accepts the six indexes the source both defines and declares a direction for. Their default directions are the source's own.

degree Number of neighbors (section 3.2). Positive.

closeness Equation (3), $1 / \sum_j l_{ij}$, the reciprocal of the raw distance sum with no $|V| - 1$ factor. Positive.

betweenness Equation (4), summed over ordered pairs $j \neq i \neq k$, so twice the usual unnormalized undirected betweenness. Positive.

constraint Equation (6), the network constraint coefficient, with the outer sum over every other node rather than over the neighbors alone. Negative.

n_components Number of connected components left after deleting the node (section 4.2). Positive.

largest_component Size of the largest component left after deleting the node (section 4.2). Negative.

The default is the four-index "distinctiveness" set of the source's Kite study. Passing all six reproduces its six-index column, and passing only `n_components` and `largest_component` its two-index "destructiveness" column. Equation (2) clustering and equation (5) eigenvector are defined in the source but never used and never declared positive or negative, and equation (7) average path length is infinite as soon as deleting a node disconnects the graph, so none of them is offered.

Conventions and undefined cases

Equation (6) is not Burt's constraint. Its outer sum runs over all of V , so a node two steps away contributes through the indirect term alone; on the source's Kite this gives node 1 the printed 1.25 where `igraph::constraint()` gives 1. The printed outer limit $j = 1 \dots |V|$ would also include $j = i$ and raise that node to 1.5, so `cograph` excludes $j = i$: it is the only reading that reproduces the printed table.

Equation (3) sums distances over all of V , which is infinite on a disconnected graph and would leave the index identically zero. `cograph` sums over the reachable partners instead. This agrees with equation (3) exactly on a connected graph, which is the graph class the source works in, and is a `cograph` extension outside it. An isolate reaches nobody, so `cograph` gives it closeness zero, and an isolate invests nowhere, so `cograph` reads its constraint investment row as zeros; both are `cograph` conventions.

There is no defensible value when a requested index is zero at every node – betweenness on a complete graph, degree on an edgeless one – because equation (8) then divides by zero, and none when equation (9)'s denominator $|V| - 1$ vanishes on a single node, or when every node is zero on some index and equation (11) divides by zero. All three raise a `cograph_undefined_index` error naming the index; none returns zeros. Naming the measure yourself always raises. When a tier such as `centrality(x, type = "all")` asked for it instead, that condition becomes a `cograph_undefined_measure` warning and an NA column, so one undefined measure does not take the whole tier down – this is what happens on a complete graph, where the betweenness index of the default set vanishes.

Uses the simple undirected unweighted skeleton, which is the source domain: either arc creates one edge, parallel edges count once and loops are removed. Edge weights, mode, cutoff and `invert_weights` are ignored. Empty graphs return no scores. The base of the logarithm in equation (10) cancels out of equation (11), so the closed form and this implementation are base-free. Raw output already sums to one; `normalized = TRUE` divides by the maximum, as elsewhere in [centrality](#), so the largest share becomes one and the vector no longer sums to one.

Numerical verification establishes agreement with the published equations and the printed Kite tables, not parity with author software, which the source does not offer, nor any claim about spreading performance.

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](https://doi.org/10.21629/JSEE.2016.06.10).

See Also

[centrality_bridging](#) for the nearest existing cograph measure by rank correlation, and [list_centralities](#) for every measure's orientation.

Examples

```
# The source's own Kite study: four distinctiveness indexes.
centrality_relative_entropy(igraph::make_graph("Krackhardt kite"))

# Only the two destructiveness indexes of its section 4.2.
centrality_relative_entropy(
  igraph::make_graph("Krackhardt kite"),
  re_indexes = c("n_components", "largest_component")
)

# Any subset works, and any index can be re-declared negative.
centrality_relative_entropy(
  igraph::make_tree(7, children = 2, mode = "undirected"),
  re_indexes = c("degree", "closeness"), re_negative = "closeness"
)
```

centrality_residual_closeness

Residual Closeness Centrality

Description

Sum of $1/2^d$ for all nodes, including self. Robust to disconnected graphs.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of residual closeness values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_dangalchev](#) (alias).

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_residual_closeness(adj)
```

centrality_resistance_curvature

Node resistance curvature

Description

Devriendt and Lambiotte's node resistance curvature is $p_i = 1 - \frac{1}{2} \sum_{j \sim i} w_{ij} R_{ij}$, where weights are electrical conductances and R is effective resistance. Equivalently, it is one minus half the expected degree in a random spanning tree whose probability is proportional to the product of its edge conductances. The expectation is taken separately within each connected component.

Usage

```
centrality_resistance_curvature(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality . With <code>normalized = TRUE</code> , scores are divided by their positive maximum; negative values remain negative and the component-sum identity no longer holds. Default raw scores retain the published interpretation.

Details

Low, possibly negative, curvature characterizes tree-like junctions; larger curvature characterizes locally redundant connections. It is a geometric descriptor, not a universal ranking of influence. On a tree the score is one minus half the degree; on an unweighted clique or cycle of n vertices every node scores $1/n$. Isolates score one, following the empty sum. Raw scores sum to the number of connected components.

Uses finite nonnegative edge weights as conductances when `weighted = TRUE`; zero weights are absent connections. Without weights, uses the simple undirected skeleton. Self-loops are always removed. For weighted directed inputs, opposite arcs are added to form undirected conductances. The `simplify` argument combines parallel edges first; remaining weighted parallel edges are added. These input projections are cograph conventions for the source's undirected domain. `mode` and `shortest-path` weight inversion do not affect the result.

Exact dense electrical systems are solved component by component, using Cholesky factors of grounded Laplacians. Squared triangular-solve norms avoid subtracting nearly equal pseudoinverse entries. Uniform rescaling of conductances within a component leaves curvature unchanged. Extreme weight ranges can still produce numerical singularity or overflow, in which case an error is raised. Dense factorization and edge solves cost up to $O(n^3 + n^2 m)$ per component; the measure is excluded from the default all tier and must be requested explicitly.

Value

Named numeric vector 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.

Examples

```
centrality_resistance_curvature(igraph::make_star(5, mode = "undirected"))
```

centrality_rsp_betweenness

Randomized Shortest Paths Betweenness Centrality

Description

Kivimäki, Lebicot, Saramäki and Sauerens interpolate between shortest-path betweenness and a random-walk quantity with a single knob. They place a Boltzmann distribution over the absorbing walks from s to t , tilted by an inverse temperature β away from the unbiased random walk and towards low-cost walks, and score a node by the expected number of visits it receives summed over every ordered source-target pair: $bet_i = \sum_{s,t} (z_{si}/z_{st} - z_{ti}/z_{tt})z_{it}$, where $Z = (I - W)^{-1}$ is the fundamental matrix of the killed random walk $W = (D^{-1}A) \circ \exp(-\beta C)$. Large `rsp_beta` concentrates the distribution on shortest paths; `rsp_beta` towards zero relaxes it to the plain random walk, where the source states the score becomes proportional to degree on an undirected graph.

Usage

```
centrality_rsp_betweenness(x, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>...</code>	Additional arguments to centrality , including <code>rsp_beta</code> and <code>rsp_cost</code> .

Details

The published closed form is only defined on a strongly connected graph, and the source says what to do otherwise. Equation (15) divides by every entry of Z , and Algorithm 1 takes "a directed strongly connected graph" as its input, but the text below equation (9) settles the general case directly: the derivation "holds only if there exists a path from s to t . Otherwise, naturally, $\bar{n}_{ij}(s, t) = 0$." `cograph` applies that zero rule to the whole term of an unreachable pair and evaluates the closed form masked by reachability, which reproduces equation (15) to machine precision whenever the graph is strongly connected and extends it consistently when it is not. The mask has to reach *both* halves of the term, since both come from the same $\bar{n}_i(s, t)$; `NetworkToolbox::rspbc()` masks

only the reciprocal and leaves the $n \text{ Diag}(Z^{\dagger})$ half counting every source, so the two part company on a disconnected graph and agree exactly on a strongly connected one.

The consequence is that scores are component-local. A node's score depends only on the pairs it can stand between, so two disjoint triangles score exactly what one triangle scores, and adding a disconnected component – an isolate included – leaves every existing score untouched. That is a cograph decision, taken because the source resolves the unreachable pair rather than because the source discusses disconnected graphs, which it does not.

Zero out-degree is a derived zero, not an imputed one. D^{-1} is undefined at out-degree zero. cograph writes that row of P^{ref} as zero, which is the paper's own killed random walk read at a node where the walker dies at once; Z then has $z_{ii} = 1$ and the arithmetic gives exactly $1 - 1 = 0$. An isolate, a singleton graph and every node of an edgeless graph therefore score zero because the formula says so. NetworkToolbox::rspbc() raises an error on such input, and centrality_current_flow_betweenness returns NA on disconnected input; this measure is able to answer where those cannot, because $(I - W)$ stays nonsingular whatever the connectivity.

rsp_beta defaults to 0.01, which is not the source's number. The paper fixes no default and treats β as a modeling choice; 0.01 is the value recommended by NetworkToolbox::rspbc(), adopted here so that the two implementations are directly comparable out of the box. It sits near the high-temperature end, so the default reading is close to the random-walk limit and far from shortest-path betweenness – raise it, to 1 or beyond, to move towards shortest paths. The domain is $\beta > 0$; zero and negative values are refused with a cograph_bad_parameter error rather than extended, since $\beta \leq 0$ is outside the Boltzmann model and can make W leave the substochastic regime the inverse depends on.

rsp_cost chooses how a weight becomes a cost, because the source leaves C free. Algorithm 1 takes the cost matrix as an input and never derives it from the weights. "inverse", the default, sets $C = 1/w$, reading a weight as an affinity so a heavier edge is cheaper; this is cograph's usual convention for a weight and the one NetworkToolbox::rspbc() hard-codes. "weight" sets $C = w$, reading a weight as a distance. The two coincide on a binary graph, where both give unit cost per arc, so the choice only bites on genuinely weighted input. Negative or non-finite weights are refused with a cograph_bad_input error: Algorithm 1 requires a non-negative cost matrix, and a negative cost makes $\exp(-\beta C) > 1$ and the Neumann series diverge.

Direction is read from the graph, not from mode: P^{ref} normalizes by out-strength and Z counts directed walks, so a directed input is scored as directed and a reversed input generally scores differently. There is no in/out/all variant to select, so the measure sits in the no-mode family. Loops are dropped and cutoff and invert_weights are ignored; the source discusses none of the three. The source states no normalization, so normalized = TRUE max-scales the finished vector as elsewhere in centrality.

Marked costly: one dense $n \times n$ inverse, which the source itself calls the computational bottleneck at $O(n^3)$ time and $O(n^2)$ memory, "because of which the method is currently not practical with very large networks" (page 7). It is held back from centrality(type = "all") and computed whenever named directly.

Numerical verification establishes agreement with the definition and with NetworkToolbox::rspbc() on strongly connected input after undoing that function's rounding and shifting, which are its own post-processing and are nowhere in the paper. The paper prints no table of node scores on a small graph, so there is no published per-node example to reproduce; what is checked against the paper instead is the printed limit claim on page 9, that the score becomes proportional to degree as $\beta \rightarrow 0^+$ on an undirected graph.

Value

Named numeric vector 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_current_flow_betweenness](#) and [centrality_random_walk](#) for the random-walk end of the same spectrum, [centrality_betweenness](#) for the shortest-path end, and [list_centralities](#) for the catalogue.

Examples

```
# A single edge scores exactly 1 at both nodes, for every rsp_beta.
centrality_rsp_betweenness(igraph::make_full_graph(2))

# A directed cycle scores n (n - 1) / 2 everywhere, independently of
# rsp_beta: every ordered pair is joined by exactly one directed path.
centrality_rsp_betweenness(igraph::make_ring(5, directed = TRUE))

# Raising rsp_beta moves the reading from the random walk towards
# shortest paths, and can reorder the nodes.
kite <- igraph::make_graph(c(1,2, 1,3, 1,4, 1,6, 2,4, 2,5, 2,7, 3,4, 3,6,
                           4,5, 4,6, 4,7, 5,7, 6,7, 6,8, 7,8, 8,9, 9,10),
                          directed = FALSE)
centrality_rsp_betweenness(kite)
centrality_rsp_betweenness(kite, rsp_beta = 1)
```

centrality_rumor

Rumor Centrality

Description

Shah and Zaman's (2010, 2011) maximum-likelihood score for the source of a rumor that has spread under the susceptible-infected model to every node. On a tree,

$$R(v) = \frac{N!}{\prod_u T_u^v},$$

where T_u^v is the number of nodes in the subtree rooted at u when the tree is rooted at v : the number of spreading orders that could have started at v . On a general graph the paper evaluates R on the breadth-first tree rooted at each node (its eq. 24). Higher values mark nodes that are more plausible origins, which in practice are nodes near the center of the network.

Usage

```
centrality_rumor(x, ...)
```

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#).

Details

The value is returned as $\log R(v)$ (natural log) because $N!$ overflows beyond 170 nodes; rankings and differences are unchanged. N is the size of the node's component, so a disconnected graph is scored component by component and an isolate scores 0. The breadth-first tree attaches each node to the earliest discovered node of the previous layer, scanning neighbors in label order; the paper does not fix a tie rule, and this one reproduces its Figure 3. Direction and edge weights are ignored.

Validated on trees against a brute-force count of spreading orders and against the worked examples in the paper.

Value

Named numeric vector, $\log R$ per node.

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](#) for computing multiple measures at once.

Examples

```
path5 <- matrix(0, 5, 5)
path5[cbind(1:4, 2:5)] <- 1; path5 <- path5 + t(path5)
rownames(path5) <- colnames(path5) <- LETTERS[1:5]
exp(centrality_rumor(path5))    # spreading orders from each node
```

centrality_salsa	<i>SALSA Authority Centrality</i>
------------------	-----------------------------------

Description

Stochastic Approach for Link-Structure Analysis. Returns authority scores. Requires a directed graph.

Usage

```
centrality_salsa(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object). Must be directed.
...	Additional arguments passed to centrality .

Value

Named numeric vector of SALSA authority scores.

See Also

[centrality](#) for computing multiple measures at once, [centrality_authority](#) for HITS authority.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_salsa(adj)
```

centrality_semilocal	<i>Semi-Local Centrality</i>
----------------------	------------------------------

Description

Triple-nested neighborhood computation measuring 4-hop local influence.

Usage

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

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>mode</code>	For directed networks: "all" (default), "in", or "out".
<code>...</code>	Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of semi-local centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_semilocal(adj)
```

centrality_shapley_game1

Shapley Value Centrality (Games 1, 2 and 3)

Description

Game-theoretic centrality of Michalak, Aadithya, Szczepanski, Ravindran and Jennings (2013): the Shapley value of each node in a coalition game whose worth $v(C)$ is the number of nodes a coalition C "covers". Each game has a closed form, so the values are exact and cost linear time.

Usage

```
centrality_shapley_game1(x, ...)

centrality_shapley_game2(x, shapley_k = 2, ...)

centrality_shapley_game3(x, shapley_cutoff = 2, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>...</code>	Additional arguments passed to centrality .
<code>shapley_k</code>	Neighbor threshold k for game 2. Default 2.
<code>shapley_cutoff</code>	Hop cutoff for game 3. Default 2.

Details

Game 1 (shapley_game1) $v(C)$ = nodes in C or adjacent to it. $SV(v) = \sum_{u \in \{v\} \cup N(v)} 1/(1 + k_u)$.

Game 2 (shapley_game2) $v(C)$ = nodes in C or with at least k neighbors in C . $SV(v) = \min(1, k/(1 + k_v)) + \sum_{u \in N(v)} \max(0, (k_u - k + 1)/(k_u(1 + k_u)))$. With $k = 1$ this is game 1. Threshold via shapley_k (default 2).

Game 3 (shapley_game3) $v(C)$ = nodes within shapley_cutoff hops of C (default 2). $SV(v) = \sum_{u \in \{v\} \cup N_d(v)} 1/(1 + |N_d(u)|)$, where $N_d(u)$ is the set of nodes within d hops of u . With cutoff 1 this is game 1.

Values in every game sum to the number of nodes (efficiency). Higher values mark nodes whose presence adds more coverage to a typical coalition. Degrees exclude self-loops, as in the paper. On a directed graph the coverage runs along out-edges and the denominators use in-degrees (the paper's stated extension); distances for game 3 are hop counts, so edge weights are ignored.

Validated against exact Shapley values obtained by enumerating every coalition on random graphs of up to eight nodes, including graphs with isolates, self-loops and several components.

Value

Named numeric vector, one Shapley value per node.

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](#) for computing multiple measures at once.

Examples

```
star5 <- matrix(0, 5, 5)
star5[1, 2:5] <- 1; star5[2:5, 1] <- 1
rownames(star5) <- colnames(star5) <- LETTERS[1:5]
centrality_shapley_game1(star5)
centrality_shapley_game2(star5, shapley_k = 2)
centrality_shapley_game3(star5, shapley_cutoff = 1)
```


Description

Xu et al.'s SpectralRank is the positive right eigenvector belonging to the largest real eigenvalue of the augmented adjacency $B = \begin{pmatrix} A+P & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix}$. A ground node connects bidirectionally to every original node with unit edge weight. The diagonal P is zero for ordinary SpectralRank; a nonnegative prior gives the paper's weighted SpectralRank family.

Usage

```
centrality_spectralrank(x, sr_prior = 0, ...)
```

Arguments

x	Network input accepted by centrality .
sr_prior	Nonnegative finite scalar or one value per original node. Default zero selects SpectralRank; one selects a uniform unit prior.
...	Additional arguments to centrality .

Details

Raw scores are scaled by the maximum over ALL nodes, including the ground node, which is then omitted from the output. Its score is not redistributed. Consequently the largest returned score can be below one. Optional `normalized = TRUE` additionally divides by the maximum over original nodes, changing this source-defined scale.

The paper uses binary adjacency and outgoing neighbors: an edge *i* to *j* contributes *j*'s score to *i*. The function preserves this orientation; transpose the graph to use incoming neighbors. Finite nonnegative edge weights extend the same matrix definition; they are interaction weights, separate from the diagonal-prior meaning of weighted SpectralRank. Unit ground edges stay fixed, so scaling original edge weights generally changes scores. For tiny asymmetric matrix weights, set `directed = TRUE` or use a directed igraph object because the shared parser otherwise uses approximate symmetry detection.

Loops are removed and remaining parallel edges sum after the generic simplify rule; unweighted remaining edges count once each. Zero weights are absent. Mode, path-weight inversion and cutoff are ignored. Named vector priors are matched to node names. Scalar priors broadcast; the ground prior is always zero. Supply externally computed degree, H-index or coreness scores as a vector to select those prior families.

All nodes, including isolates, receive positive spectral scores because of the ground links. Without edges or priors, each of *n* original nodes scores $1/\sqrt{n}$. For an edgeless graph the paper's unshifted power iteration oscillates, although the Perron eigenvector is unique. This function explicitly uses that eigenvector definition, without claiming convergence of the published iteration. A singleton scores one; an empty graph returns no scores. Adding disconnected nodes generally changes other scores because all share the ground node.

For nonzero priors the implementation follows section III-A2's $B = \tilde{A} + P$. Algorithm 1 constructs that matrix but its update line prints $\tilde{A}$, omitting P; this inconsistency is retained in the verification audit. No author-software parity is claimed.

Dense eigendecomposition takes $O(n^3)$ time and $O(n^2)$ memory. Extreme weight/prior ranges or unresolved positive eigenpairs raise errors. The score defines a spectral ranking, not a spreading probability or a general guarantee of predictive performance.

Value

Named numeric vector 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.

Examples

```
centrality_spectralrank(igraph::make_ring(5))
centrality_spectralrank(igraph::make_star(5), sr_prior = 1)
```

centrality_strength	<i>Strength Centrality (Weighted Degree)</i>
---------------------	--

Description

Sum of edge weights connected to each node. For directed networks, centrality_instrength sums incoming weights and centrality_outstrength sums outgoing weights.

Usage

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

centrality_instrength(x, ...)

centrality_outstrength(x, ...)
```

Arguments

- x Network input (matrix, igraph, network, cograph_network, tna object).
- mode For directed networks: "all" (default), "in", or "out".
- ... Additional arguments passed to centrality (e.g., normalized, weighted, directed).

Value

Named numeric vector of strength values.

See Also

centrality for computing multiple measures at once, centrality_degree for the unweighted version.

Examples

```
mat <- matrix(c(0, .5, .3, .5, 0, .8, .3, .8, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
centrality_strength(mat)
```

centrality_stress	<i>Stress Centrality</i>
-------------------	--------------------------

Description

Number of shortest paths passing through each node. Unlike betweenness, does not normalize by the total number of shortest paths.

Usage

```
centrality_stress(x, ...)
```

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	Additional arguments passed to centrality .

Value

Named numeric vector of stress centrality values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_betweenness](#) for the normalized variant.

Examples

```
adj <- matrix(c(0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
centrality_stress(adj)
```

centrality_subgraph *Subgraph Centrality*

Description

Participation in closed loops (walks), weighting shorter loops more heavily. Based on the diagonal of the matrix exponential of the adjacency matrix.

Usage

```
centrality_subgraph(x, ...)
```

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of subgraph centrality values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_subgraph(adj)
```

centrality_s_shell *s-shell Index*

Description

Liu, Tang, Do and Hui's (2017) strength-based generalization of k-shell for identifying spreaders. Each link is given an asymmetric weight from the topology alone,

$$w_{ij} = 1 + (k_i k_j^{out})^a,$$

where k_j^{out} is the number of j 's neighbors that lie outside i 's closed neighborhood (links that lead a spreading process to new territory), and each node's strength is $s_i = \sum_{j \in N(i)} w_{ij}$. The graph is then peeled like a k-shell but by strength: the minimum remaining strength is the threshold, everything at or below it is removed (neighbors lose the corresponding w_{ji}), removals cascade until the threshold holds, and the removed nodes receive the next shell index. Higher index = more central. With $a = 0$ the shells are the dense ranks of the k-core numbers.

Usage

```
centrality_s_shell(x, s_shell_a = 0.5, ...)
```

Arguments

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>s_shell_a</code>	Exponent a of the link weights. A single non-negative number; default 0.5. Anything else raises a <code>cograph_bad_parameter</code> error.
<code>...</code>	Additional arguments passed to centrality .

Details

The index is an ordinal counter (1 = outermost shell), not a strength value, so it is not comparable across graphs. Isolates form shell 1 on their own, shifting every other shell up by one, as the paper's rule implies. Direction, edge weights and self-loops are ignored. The paper's robust default is $a = 0.5$.

Validated against the shell peeled at each threshold being exactly the complement of the maximal subgraph in which every node keeps strength above the threshold (brute force over all vertex subsets), and against k-core dense ranks at $a = 0$.

Value

Named integer vector of shell indices, one per node.

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](#) for the k-shell index.

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 2, 1, 3, 4, 5), c(2, 3, 3, 4, 5, 6))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_s_shell(adj)
```

centrality_topological_coefficient
Topological Coefficient

Description

Fraction of shared second-order neighbors, measuring topological overlap between a node and its neighbors.

Usage

```
centrality_topological_coefficient(x, ...)
```

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
 ... Additional arguments passed to [centrality](#).

Value

Named numeric vector of topological coefficient values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_topological_coefficient(adj)
```

centrality_transitivity
Local Transitivity (Clustering Coefficient)

Description

Proportion of triangles around each node relative to the number of possible triangles. Measures how tightly clustered a node's neighborhood is.

Usage

```
centrality_transitivity(x, transitivity_type = "local", isolates = "nan", ...)
```

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).

`transitivity_type` Type of transitivity: "local" (default), "global", "undirected", "localundirected", "barrat" (weighted), "weighted", or "onna". "onna" computes the Onnela / Holme weighted clustering coefficient on the symmetrized matrix and matches `tna::centralities(., "Clustering")` byte-for-byte. Auto-set to "onna" when `tna_network = TRUE` (passed via ...) and the user did not pass an explicit value.

`isolates` How to handle isolate nodes: "nan" (default) or "zero".

... Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of transitivity values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_transitivity(adj)
```

centrality_truss	<i>Truss, mixed-degree decomposition and local social-capital measures</i>
------------------	--

Description

Five measures with explicit definitions and numerical reference checks. All use the simple, unweighted, undirected skeleton: either direction creates an edge, parallel edges count once and self-loops are removed. This projection is a cograph input convention; no directed or weighted generalization of the published measures is claimed. All isolates score 0.

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 .
...	Additional arguments to centrality . With normalized = TRUE, positive scores are divided by their maximum.
mdd_lambda	Exhausted-degree weight between 0 and 1, default 0.7.

Details

truss Maximum truss number of an incident edge (Malliaros et al. 2016). A k-truss requires at least k-2 triangles per edge within the surviving subgraph, matching NetworkX. An edge outside any triangle has truss number 2; a complete graph on k vertices has node truss number k. Some sources instead label by the triangle threshold, producing values two smaller.

mdd Mixed-degree decomposition (Zeng & Zhang 2013): repeatedly peel by residual degree plus mdd_lambda times exhausted degree. Nodes falling below the current shell threshold join that shell before the threshold advances. Zero recovers the k-core number; one recovers degree. Intermediate thresholds are real-valued. Default 0.7, as in the paper's worked example.

bridging_coefficient Hwang et al.'s reciprocal-degree ratio: $(1/d_i) / \sum_{j \in N(i)} 1/d_j$. This is the coefficient itself, before multiplication by betweenness.

godfather Jackson's Godfather index: the number of unordered pairs of neighbors with no edge between them. Equals $d_i(d_i - 1)/2$ minus the number of triangles containing i.

support Jackson's supported relationships: the number of neighbors sharing at least one common neighbor with i. An edge is counted once even if it belongs to multiple triangles.

LocalRank (Chen et al. 2012), also listed in the Centrality Zoo, is already available as [centrality_semilocal](#) on an undirected, unweighted graph; it needs no additional numerical function.

Value

Named numeric vector 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

[list_centralities](#), [centrality_coreness](#), [centrality_bridging](#).

Examples

```
adj <- matrix(1, 4, 4)
diag(adj) <- 0
centrality_truss(adj)
centrality_mdd(adj, mdd_lambda = 0.7)
centrality_support(adj)
```

centrality_trust_pagerank
Trust-PageRank

Description

Sheng, Zhu, Wang, Wang and Hou replace PageRank's uniform split of a node's score among its neighbors with a *trust-value* that mixes how similar two nodes are with how large the receiving node's degree is. The similarity is SimRank restricted to the lines of the graph, the degree ratio is a node's degree over the total degree of its partner's neighborhood, and the two are blended and fed to a damped power iteration:

$$Rs_{ij} = \frac{s(i, j)}{\sum_{k \in N_j} s(j, k)}, \quad Rd_{ij} = \frac{d_i}{\sum_{k \in N_j} d_k},$$

$$T(i, j) = (1 - k)Rs_{ij} + kRd_{ij}, \quad TPR_i^t = \frac{1 - \alpha}{n} + \alpha \sum_{j \in N_i} T(i, j)TPR_j^{t-1},$$

with the similarity itself the fixed point of $s(a, a) = 1$ and $s(a, b) = (C/(|N_a||N_b|)) \sum_{l \in N_a} \sum_{m \in N_b} s(l, m)$.

Usage

```
centrality_trust_pagerank(x, ...)
```

Arguments

x	Network input accepted by centrality .
...	Additional arguments to centrality , including tpr_alpha, tpr_k, tpr_decay, tpr_tol and tpr_max_iter.

Details

The Centrality Zoo cites the wrong paper for this measure. Its entry 2.381 attributes Trust-PageRank to Sheng et al.'s *Physica A* 541:123262, which defines the unrelated global-and-local-structure index. The measure printed there is equations (2), (4), (5), (6) and (7) of the *Algorithms* paper cited below, which is fully open access. A reader following the Zoo's reference will land on a different measure.

Both ratios are normalized over the receiving node's neighborhood, so the trust matrix is column-stochastic and equation (7) is an ordinary damped PageRank. Because s is symmetric, $\sum_{i \in N_j} Rs_{ij} = 1$ and $\sum_{i \in N_j} Rd_{ij} = 1$ whatever k is, so $\sum_{i \in N_j} T(i, j) = 1$ and the iteration has a

unique fixed point at which the scores sum to one. The source never fixes an iteration count and does not need to: the count is a convergence tolerance, exposed here as `tpr_tol` and `tpr_max_iter`, and a recursion still moving at the bound raises `cograph_no_converge` rather than returning a silently unconverged estimate. **Both tests are relative rather than absolute:** the similarities on one graph span many orders of magnitude, because the mass reaching a line decays geometrically with its distance from the nearest triangle, and on a long chain at $C = 0.2$ the largest similarity is 2.4×10^{-2} while the smallest positive one is 2.7×10^{-20} . An absolute test would stop while those small entries were still an order of magnitude out, and equation (2) divides two of them by each other. An isolate emits nothing, so on a graph with isolates the scores sum to less than one.

The similarity recursion runs on the lines of the graph only, and this is what makes it converge. Algorithm 1’s line 4 quantifies over *connected* pairs and Table 3 marks every non-adjacent cell with a dash, so the similarity map holds an entry for each line and for the diagonal, and a non-adjacent pair entering the double sum contributes zero rather than the 0.1 that initializes the lines. The diagonal $s(l, l) = 1$ is then the only inhomogeneous term, and it reaches a line (a, b) exactly through the common neighbors of a and b – that is, through the triangles the line carries. Each row of the linear part sums to at most $1 - p/(d_a d_b)$ for a line on p triangles, so the recursion contracts on any block that carries a triangle even at the source’s $C = 1$. Pinning non-adjacent pairs at 0.1 instead reproduces neither published fixture; see the batch 51 published audit in the package’s verification directory.

On a component that has lines but no triangle the measure has no value, and that whole component is returned as NA. With no triangle the recursion is homogeneous, its least nonnegative fixed point is $s \equiv 0$, and equation (2) divides zero by zero. An undefined column makes equation (7) undefined for everything that solves against it, which is why the NA covers the component rather than the one node. The class is not a corner case: every path, tree, star, even cycle and complete bipartite graph is in it, and so is the Petersen graph. An isolate is *not*: it is never a denominator in equation (2), and equation (7) gives it the bare $(1 - \alpha)/n$. `cograph` refuses to name a value on the rest. The obvious fallback, $Rs_{ij} := 1/d_j$, was considered and rejected: it is not forced by the vanishing numerators the way the zero of `centrality_dil` and `centrality_lhc` is, since the ratios need only sum to one over N_j and nothing in the source chooses between the ways of doing that; adopting it would silently turn the measure into a degree-ratio PageRank over the whole triangle-free class while still calling it Trust-PageRank. This follows `centrality_iec`, which returns NA rather than the finite number its closed form would otherwise print, and deliberately does not follow `centrality_dil`. A second reading – start the recursion at the source’s 0.1 rather than at zero – would define the sub-class on which that start is itself a fixed point (K_2 , P_3 , stars, C_4 , complete bipartite graphs), where it yields $Rs_{ij} = 1/d_j$ independently of the constant’s size. It was rejected because the value is then an artifact of the initialization being uniform rather than of the graph, because it leaves the rest of the triangle-free class undefined anyway, and because separating it from an exponentially decaying zero needs a numerical threshold where `cograph` can instead settle the question structurally, by asking which lines can reach a triangle at all.

Direction and weights are dropped, because the source excludes them. Page 3 sets the paper in an undirected network with $a(i, j) = 1$, and every quantity in the five equations is a count or a ratio of counts. A directed, weighted or multigraph input is projected onto its simple undirected skeleton – arcs symmetrized, weights and parallel edges collapsed to a single line, loops dropped – as every other undirected-domain measure in `centrality` does, so `mode`, `cutoff` and `invert_weights` are ignored. The source states no normalization, so `normalized = TRUE` max-scales the finished vector as elsewhere.

The source’s claim that C does not matter is false for the converged recursion, and C is ex-

posed rather than hidden. Page 5 argues that "the value of C does not affect the results, since only the ratio of similarity is calculated". That holds for a homogeneous recursion, where C is an overall scale, but not for this one: the diagonal makes it affine, so C enters the resolvent as well as the scale. Measured on the Zachary karate club, moving C from 1 to 0.5 moves R_s by up to 0.141 and the scores by up to 9.1×10^{-4} . `tpr_decay` defaults to the source's 1.

Both published fixtures are reproduced. Table 3 on page 6 prints seven similarities of the five-node network of Fig. 3, and Table 5 on page 10 prints the Trust-PageRank top ten of the Krackhardt kite and of the Zachary karate club. All seven similarities round to their printed two decimals and all ten karate positions are recovered in order; the kite is recovered up to three exact ties forced by its own automorphism. See the batch 51 published audit.

Value

Named numeric vector in input node order, one score per node, summing to one on a graph with no isolate and no undefined component. NA at every node of a component that has lines but no triangle, accompanied by a `cograph_undefined_measure` warning. The domain does not depend on `tpr_k`: the similarity ratio is part of the trust-value at every mixing weight, and `cograph` does not switch a measure's domain on the knife-edge value `tpr_k = 1`.

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](https://doi.org/10.3390/a13110280).

See Also

[centrality_pagerank](#) for the uniform split this measure replaces, [centrality_dil](#) and [centrality_lhc](#) for other triangle-aware scores, [centrality_iec](#) for the other measure that returns NA outside its domain, and [list_centralities](#) for the catalogue.

Examples

```
# The Krackhardt kite, one of the source's two published fixtures. Its
# automorphism forces three exact ties, so the paper's printed order
# 7, 4, 5, 9, 10, 3, 6, 8, 2, 1 is recovered up to those ties.
kite <- igraph::make_graph(
  c(6, 10, 6, 5, 6, 7, 10, 5, 10, 7, 10, 9, 5, 7, 5, 4, 5, 3,
    7, 9, 7, 4, 7, 8, 9, 4, 9, 8, 4, 8, 4, 3, 3, 2, 2, 1),
  directed = FALSE)
centrality_trust_pagerank(kite)

# A complete graph is vertex-transitive, so every node scores 1 / n.
centrality_trust_pagerank(igraph::make_full_graph(5))

# The similarity has nothing to work with on a triangle-free graph, so
# the measure declines to score a ring rather than inventing a split.
suppressWarnings(centrality_trust_pagerank(igraph::make_ring(6)))
```

centrality_two_way_rw *Two-Way Random Walk Betweenness*

Description

Curado, Rodriguez, Tortosa and Vicent's (2022) counting measure. For every unordered pair (i, j) the two-step transfer $P_{itj} = w_{it}w_{tj}/(d_i d_j)$ (zero when any two of the three coincide) is combined into $T_{ij}[t, k] = P_{itj}P_{jki}$, the diagonal is dropped, and the single largest entry credits one count to t and one to k . A node's score is its total count over all pairs. Higher = more central; nodes never on a winning two-way route score 0, so sparse tails are not ranked. Reproduces the paper's toy example exactly, including every printed fraction.

Usage

```
centrality_two_way_rw(x, ...)
```

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).
`...` Additional arguments passed to [centrality](#).

Details

The paper's P_{itj} is not a random-walk probability (its denominator is $d_i d_j$, not $d_i d_t$); it is implemented as printed. Ties in the maximum go to the first entry in row-major order. Edge weights are used; direction and loops are ignored. Cost is $O(n^4)$: fine to a few hundred nodes, slow beyond.

Value

Named numeric vector of counts, one per node.

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](#) for Newman's random-walk betweenness.

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_two_way_rw(adj)
```

centrality_volume	<i>Volume centrality</i>
-------------------	--------------------------

Description

Sum of the original-graph degrees of all vertices within `volume_radius` hops, including the focal vertex. This is the localized volume measure of Wehmuth & Ziviani (DANCE/DACCER). Degrees include edges leaving the neighborhood; they are not recomputed inside the induced subgraph. Radius zero returns degree. Infinite radius returns twice the number of edges in the focal connected component.

Usage

```
centrality_volume(x, volume_radius = 2, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>volume_radius</code>	Nonnegative integer hop radius, or Inf. Default 2, the local radius investigated by Wehmuth & Ziviani.
<code>...</code>	Additional arguments to centrality . With <code>normalized = TRUE</code> , positive scores are divided by their maximum.

Details

Uses the simple undirected, unweighted skeleton: either direction creates an edge, parallel edges count once, and self-loops are removed. This is an explicit input projection, not a weighted or directed generalization. Isolates score zero.

Value

Named numeric vector 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](#).

Examples

```
centrality_volume(igraph::make_ring(6), volume_radius = 1)
centrality_volume(igraph::make_ring(6), volume_radius = 0)
```

centrality_voterank	<i>VoteRank Centrality</i>
---------------------	----------------------------

Description

Identifies influential spreaders via an iterative voting mechanism. Returns normalized rank (1 = most influential). Based on Zhang et al. (2016).

Usage

```
centrality_voterank(x, ...)
```

Arguments

- x Network input (matrix, igraph, network, cograph_network, tna object).
- ... Additional arguments passed to [centrality](#) (e.g., weighted, directed).

Value

Named numeric vector of VoteRank values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_voterank(adj)
```

centrality_weighted_kshell	<i>Weighted k-shell, Renewed Coreness and Geodesic k-path</i>
----------------------------	---

Description

weighted_kshell (**Garas, Schweitzer & Havlin 2012**) k-shell decomposition on the generalized degree $k' = (k^\alpha s^\beta)^{1/(\alpha+\beta)}$ (wks_alpha, wks_beta, both 1), after the paper's weight normalization (divide by the mean, then by the minimum, round to the nearest integer). Integer thresholds label the shells, so unit weights give the k-core number and isolates score 0. Reproduces the paper's Figure 1 example and its Table 2 core size on the netscience network. Uses edge weights.

renewed_coreness (Liu, Tang, Zhou & Do 2015) Each link gets the diffusion importance $D_{ij} = (|N(j) \setminus N[i]| + |N(i) \setminus N[j]|)/2$; links below `renewed_threshold` (paper: 2) are removed and the k-core number of the residual graph is the renewed coreness. A clique with no outside links collapses to 0. Reproduces the paper's Figure 1 and all twelve percentages of its supplementary Table S1; the Zoo's transcription with open neighborhoods is off by one.

geodesic_kpath (Borgatti & Everett 2006) The number of shortest paths of length at most `kpath_k` (default 3) that start at the node, counted with multiplicity. Note that `centiserve::geokpath` counts nodes within k instead, which is the paper's vertex-disjoint variant and equals m-reach.

`geodesic_kpath` follows `mode`; the other two ignore direction.

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

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object).
<code>wks_alpha, wks_beta</code>	Exponents of degree and strength in the weighted k-shell. Default 1 and 1.
<code>...</code>	Additional arguments passed to centrality .
<code>renewed_threshold</code>	Diffusion-importance threshold. Default 2.
<code>mode</code>	For directed networks: "all" (default), "out" (distances along out-edges), or "in".
<code>kpath_k</code>	Maximum path length. Default 3.

Value

Named numeric vector, one value per node.

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](https://doi.org/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_kreach](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_weighted_kshell(adj)
centrality_renewed_coreness(adj)
centrality_geodesic_kpath(adj, kpath_k = 2)
```

centrality_weighted_leaderrank

Weighted LeaderRank centrality

Description

Li et al.'s weighted LeaderRank adds a ground node g . Each original directed edge and each edge from an original node to g has weight one. The edge from g to node i has weight $(k_i^{in})^\alpha$, using original in-degree before ground edges are added. Scores follow the stationary distribution of the row-normalized augmented matrix.

Usage

```
centrality_weighted_leaderrank(x, wlr_alpha = 1, ...)
```

Arguments

<code>x</code>	Network input accepted by centrality .
<code>wlr_alpha</code>	Finite in-degree exponent, default one, a setting studied in the source rather than a universal optimum.
<code>...</code>	Additional arguments to centrality . <code>normalized = TRUE</code> divides final scores by their maximum.

Details

Raw scores retain total mass $N+1$ across the augmented graph, following the all-nodes-one initialization in the original paper, section 2. The ground score is omitted from the returned vector without redistribution. The Zoo instead initializes the ground at zero, yielding raw scores smaller by $N/(N+1)$; final max-normalized scores agree. The existing [centrality_leaderrank](#) uses a different redistribution/scale convention, so raw equality at alpha zero is not asserted.

Directed arcs are retained; an undirected edge is treated as two opposite arcs, an explicit extension. Input weights are ignored: `weighted` refers to the algorithm's ground-edge weights. Loops are removed and parallel arcs count once. Mode, path inversion and cutoff do not change the result.

Alpha can be any finite number. Negative values require strictly positive original in-degree at every node. At alpha zero all ground-edge weights are one, including for zero-in-degree nodes. With positive alpha, these nodes receive no ground resource and have zero stationary score; if every in-degree is zero the ground row is undefined and all scores are NaN. Empty input returns an empty

vector. These boundary conventions are explicit; no pseudocount is added to the published in-degree weights.

A native linear solve eliminates the ground variable and obtains the unique stationary distribution even when ordinary iteration is periodic. This uses $O(N^3)$ time and $O(N^2)$ memory. Ground transition probabilities are calculated with shifted logarithms, avoiding overflow for large exponents; extremely small probabilities may underflow to zero.

Value

Named numeric vector in input node order.

References

Li, Q., Zhou, T., Lu, L., & Chen, D. (2014). Identifying influential spreaders by weighted Leader-Rank. *Physica A*, 404, 47-55. doi:[10.1016/j.physa.2014.02.041](https://doi.org/10.1016/j.physa.2014.02.041).

Examples

```
centrality_weighted_leaderrank(igraph::make_ring(4, directed = TRUE))
```

centrality_wiener	<i>Wiener Index Centrality</i>
-------------------	--------------------------------

Description

Total sum of shortest path distances from a node to all others. Higher values indicate less central (more peripheral) nodes.

Usage

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

Arguments

- x Network input (matrix, igraph, network, cograph_network, tna object).
- mode For directed networks: "all" (default), "in", or "out".
- ... Additional arguments passed to [centrality](#) (e.g., normalized, weighted, directed).

Value

Named numeric vector of Wiener index values.

See Also

[centrality](#) for computing multiple measures at once.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
centrality_wiener(adj)
```

centrality_within_module_z

Within-Module Degree Z-Score

Description

Z-score of intra-community connectivity. High values indicate hubs within their own community. Requires community membership.

Usage

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

Arguments

x	Network input (matrix, igraph, network, cograph_network, tna object).
membership	Integer vector of community assignments (one per node).
mode	For directed networks: "all" (default), "in", or "out".
...	Additional arguments passed to centrality .

Value

Named numeric vector of within-module z-score values.

See Also

[centrality](#) for computing multiple measures at once, [centrality_participation](#) for between-community diversity.

Examples

```
adj <- matrix(c(0,1,1,0,0, 1,0,1,0,0, 1,1,0,1,0, 0,0,1,0,1, 0,0,0,1,0), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
centrality_within_module_z(adj, membership = c(1, 1, 1, 2, 2))
```

centrality_wvoterank *WVoteRank, EnRenew and VoteRank++*

Description

Three further spreader-selection procedures in the VoteRank family. All three elect one node per round until every node is placed and return the election order as a score, 1 for the first elected down to $1/n$; ties go to the lowest node index. Direction and self-loops are ignored.

Usage

```
centrality_wvoterank(x, ...)

centrality_enrenew(x, enrenew_depth = 2, ...)

centrality_voterank_plus(x, voterank_lambda = 0.1, ...)
```

Arguments

x Network input (matrix, igraph, network, cograph_network, tna object).
... Additional arguments passed to [centrality](#).
enrenew_depth Renewal radius l for enrenew. Default 2.
voterank_lambda Suppression factor λ for voterank_plus. Default 0.1.

Details

wvoterank (Sun, Chen, He & Ch'ng 2019) VoteRank for weighted graphs: $s_v = \sqrt{k_v \sum_{u \in N(v)} v a_u w_{vu}}$.

After an election the winner's ability is 0 and its neighbors lose $1/\langle w \rangle$, where $\langle w \rangle$ is the average strength (the paper's Figure 1 pins strength, not degree). Uses edge weights; with unit weights it is VoteRank with a square-root score. Reproduces all sixty numbers of the paper's Figure 1.

enrenew (Guo, Yang, Guo, Pan & Chen 2020) Entropy-based selection: $E_v = \sum_{u \in N(v)} -p_{uv} \ln p_{uv}$ with $p_{uv} = k_u / \sum_{l \in N(v)} k_l$; after electing the largest E , every entropy term flowing outward to depth $d \leq l$ is scaled by $1 - 1/(2^{d-1} \ln \langle k \rangle)$, with $l = \text{enrenew_depth}$ (default 2). Reproduces the paper's Figure 1. The authors' released code differs from the paper in several ways; the paper is implemented. Note the factor turns negative when $\langle k \rangle < e$.

voterank_plus (Liu, Li, Fang & Yao 2021) Initial ability $\ln(1 + k_i/k_{\max})$, degree-proportional vote shares over unelected neighbors, score $\sqrt{k_i \sum_j v a_j w_{j \rightarrow i}}$, and after an election abilities are multiplied by λ one step away and $\sqrt{\lambda}$ two steps away (voterank_lambda, default 0.1). The article is closed access; the implementation matches the authors' released code exactly, including its exclusion of elected nodes from the vote-share denominator.

Value

Named numeric vector in $(0, 1]$, one score per node.

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](#).

Examples

```
adj <- matrix(0, 6, 6)
adj[cbind(c(1, 1, 2, 4, 4, 5, 3), c(2, 3, 3, 5, 6, 6, 4))] <- 1
adj <- adj + t(adj)
rownames(adj) <- colnames(adj) <- LETTERS[1:6]
centrality_wvoterank(adj)
centrality_enrenew(adj)
centrality_voterank_plus(adj)
```

centrality_x_degree	<i>X-degree centrality</i>
---------------------	----------------------------

Description

Computes Torres et al.'s X-degree (equation 3.15):

$$Xdeg(i) = \left(\sum_{j \in N(i)} (d_j - 1) \right)^2 - \sum_{j \in N(i)} (d_j - 1)^2.$$

Degrees are measured in the original simple undirected graph. The score counts oriented nonbacktracking walks of four edges whose middle vertex is *i*. Walks can revisit a vertex provided they do not immediately reverse an edge. It is also the sum of entries of the paper's matrix DFE, where D, F and E are blocks of the nonbacktracking matrix around *i*.

Usage

```
centrality_x_degree(x, ...)
```

Arguments

- | | |
|------------------|---|
| <code>x</code> | Network input accepted by centrality . |
| <code>...</code> | Additional arguments to centrality . <code>normalized = TRUE</code> divides scores by their maximum; an all-zero result stays zero. |

Details

Uses the simple undirected skeleton: direction, weights, mode, inversion and cutoff do not affect results. Loops are removed and parallel edges count once. This projection is a cograph convention extending the published simple, unweighted, undirected domain. Isolates and leaves score zero; every vertex of a star also scores zero. Empty graphs return no scores. Disconnected components are independent before maximum normalization. These cases follow directly from the local formula.

Native arithmetic accumulates nonnegative pair products instead of subtracting two squares. Aggregation takes $O(n+m)$ time after neighbor construction; the current dense skeleton conversion uses $O(n^2)$ time and memory. This is a score on the supplied graph, not the paper's iterative node-removal immunization algorithm. Agreement with the author function and matrix definition does not establish immunization efficacy, exact eigendrop prediction or an unconditional spectral upper bound.

Value

Named numeric vector 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.

Examples

```
centrality_x_degree(igraph::make_graph("Zachary"))
```

centralization	<i>Centralization index</i>
----------------	-----------------------------

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: the summed gap between the most central node and every other node, divided by the theoretical maximum for the measure, so 0 marks a perfectly even network and 1 a perfect star. Nodes whose score is NA or NaN are dropped from the sum. Returns 0 when the network has two or fewer nodes.

Examples

```
star <- matrix(0, 5, 5)
star[1, 2:5] <- 1; star[2:5, 1] <- 1
cograph::centralization(star, "degree")
```

cluster_quality	<i>Cluster Quality Metrics</i>
-----------------	--------------------------------

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

```
mat <- matrix(runif(100), 10, 10)
diag(mat) <- 0
clusters <- c(1,1,1,2,2,2,3,3,3,3)

q <- cluster_quality(mat, clusters)
q$per_cluster # Per-cluster metrics
q$global      # Modularity, coverage
mat <- matrix(runif(100), 10, 10)
diag(mat) <- 0
cqual(mat, c(1,1,1,2,2,2,3,3,3,3))
```

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

<code>x</code>	Network input: adjacency matrix, igraph object, or <code>cograph_network</code> .
<code>communities</code>	A communities object (from communities or igraph) or a membership vector (integer vector where <code>communities[i]</code> is the community of node <code>i</code>).
<code>n_random</code>	Number of random networks to generate for the null distribution. Default 100.
<code>method</code>	Null model type: "configuration" Preserves degree sequence (default). More stringent test. "gnm" Erdos-Renyi model with same number of edges. Tests against random baseline.
<code>null</code>	Which null question to answer. Default "detect": "detect" Null is the modularity of the <i>best partition found by community detection</i> on each null graph. Answers "is the observed partition stronger than what community detection would recover on similar random graphs?" — the historical behavior. "fixed" Null is the modularity of the supplied <code>communities</code> membership <i>evaluated on each null graph</i> . Answers "does the supplied partition itself explain more structure than it would on similar random graphs?" — the conservative test.
<code>seed</code>	Random seed for reproducibility. Default NULL.

Details

Two null models are supported. The default, `null = "detect"`, generates `n_random` random networks, runs community detection (Louvain, with fast-greedy fallback) on each, and records the resulting modularity. Low p-value means the observed partition beats what detection would return on similar random graphs. `null = "fixed"` instead evaluates the user-supplied membership on each null graph, so low p-value means the partition itself is stronger than it would be on similar random graphs — a tighter question that isolates the partition's quality from any detector's behavior.

A significant result (low p-value) indicates that the community structure is stronger than expected by chance for networks with similar properties.

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: $(\text{observed} - \text{null_mean}) / \text{null_sd}$

p_value One-sided p-value (probability of observing equal or higher modularity by chance)

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

See [cluster_significance](#).

References

Reichardt, J., & Bornholdt, S. (2006). Statistical mechanics of community detection. *Physical Review E*, 74, 016110.

See Also

[communities](#), [cluster_quality](#)

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
sig <- cluster_significance(g, comm, n_random = 20, seed = 123)
print(sig)

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")
  comm <- community_louvain(g)
  csig(g, 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", "gephi", or "custom"; a coordinate matrix/data frame; a CographLayout; or an igraph layout function/name. Default NULL (no layout computed). Set to a layout to compute immediately, or use <code>sn_layout()</code> later.
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. 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.
...	Additional arguments passed to the layout function.

Value

A `cograph_network` object that can be further customized and rendered.

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

```
# From adjacency matrix (layout computed lazily on first plot)
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |> splot()

# From edge list
edges <- data.frame(from = c(1, 1, 2), to = c(2, 3, 3))
cograph(edges) |> splot(layout = "circle")

# Pipe-friendly customization
cograph(adj) |>
  sn_nodes(fill = "steelblue") |>
  sn_edges(color = "gray50") |>
  splot(layout = "circle")
```

CographLayoutCographLayout 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 type (e.g., "circle", "spring", "groups").

`...` Additional parameters for the layout algorithm.

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.

Returns: Data frame with x, y coordinates.

Method `normalize_coords()`: Normalize coordinates to 0-1 range with padding.

Usage:

```
CographLayout$normalize_coords(coords, padding = 0.1)
```

Arguments:

`coords` Matrix or data frame with x, y columns.
`padding` Numeric. Padding around edges (default 0.1).

Returns: Normalized coordinates.

Method `get_type()`: Get layout type.

Usage:

`CographLayout$get_type()`

Returns: Character string.

Method `get_params()`: Get layout parameters.

Usage:

`CographLayout$get_params()`

Returns: List of parameters.

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

```
# Create a circular layout
layout <- CographLayout$new("circle")

# Apply to network
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- CographNetwork$new(adj)
coords <- layout$compute(net)
```

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 Whether edges have weights.

node_labels Vector of node labels (priority: labels > label).

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 supported by `parse_input`, such as a matrix, edge list, igraph, statnet network, qgraph, or tna object.

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.

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 edges are aggregated.

Returns: A new CographNetwork object.

Method `clone_network()`: Clone the network with optional modifications.

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 x, y columns, one row per node.

Returns: The object itself, invisibly.

Method `set_node_aes()`: Set node aesthetics.

Usage:

```
CographNetwork$set_node_aes(aes)
```

Arguments:

`aes` List of aesthetic parameters.

Returns: The object itself, invisibly.

Method `set_edge_aes()`: Set edge aesthetics.

Usage:

```
CographNetwork$set_edge_aes(aes)
```

Arguments:

`aes` 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: Data frame with x, y coordinates.

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: CographTheme object.

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

```
# Create network from adjacency matrix
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- CographNetwork$new(adj)

# Access properties
net$n_nodes
net$n_edges
net$is_directed
```

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.

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.

Usage:

`CographTheme$merge(other)`

Arguments:

`other` Another CographTheme or list of parameters.

Returns: A new merged CographTheme.

Method `clone_theme()`: Clone the theme.

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

```
# Create a custom theme
theme <- CographTheme$new(
  background = "white",
  node_fill = "steelblue",
  edge_color = "gray60"
)
```

color_communities	<i>Color Nodes by Community</i>
-------------------	---------------------------------

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>method</code>	Community detection algorithm. See detect_communities for available methods. Default "louvain".
<code>palette</code>	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"
<code>...</code>	Additional arguments passed to detect_communities .

Value

A named character vector of colors (one per node), suitable for use with `splot()` `node_fill` parameter.

See Also

[detect_communities](#), [splot](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Basic usage with splot
splot(adj, node_fill = color_communities(adj))

# Custom palette
splot(adj, node_fill = color_communities(adj, palette = c("red", "blue")))
```

communities	<i>Community Detection</i>
-------------	----------------------------

Description

Detects communities/clusters in networks using various algorithms. Provides a unified interface to igraph's community detection functions.

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" - Louvain modularity optimization (default, fast) • "leiden" - Leiden algorithm (improved Louvain) • "fast_greedy" - Fast greedy modularity optimization • "walktrap" - Random walk-based detection • "infomap" - Information theoretic approach • "label_propagation" - Label propagation (very fast) • "edge_betweenness" - Girvan-Newman algorithm • "leading_eigenvector" - Leading eigenvector method • "spinglass" - Spinglass simulation • "optimal" - Exact modularity optimization (slow) • "fluid" - Fluid communities algorithm
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. If NULL, uses edge weights from the network if available, otherwise unweighted. Set to NA for explicitly unweighted.

resolution	Resolution parameter for modularity-based methods (louvain, leiden). Higher values yield more communities. Default 1.
directed	Logical; whether edge-betweenness should treat the network as directed. Default NULL (auto-detect for edge-betweenness). Other methods use their own directed/undirected handling.
seed	Random seed for reproducibility. Only applies to stochastic algorithms (louvain, leiden, infomap, label_propagation, spinglass).
...	Additional parameters passed to the specific algorithm. See individual functions for details.

Details

When called through this wrapper, methods that require undirected graphs ("louvain", "leiden", "fast_greedy", "leading_eigenvector", and "fluid") fall back to "walktrap" if the input graph is directed.

Algorithm Selection Guide:

Algorithm	Best For	Time Complexity
louvain	Large networks, general use	$O(n \log n)$
leiden	Large networks, better quality than louvain	$O(n \log n)$
fast_greedy	Medium networks	$O(n^2 \log n)$
walktrap	Networks with clear community structure	$O(n^2 \log n)$
infomap	Directed networks, flow-based	$O(E)$
label_propagation	Very large networks, speed critical	$O(E)$
edge_betweenness	Small networks, hierarchical	$O(E^2 n)$
leading_eigenvector	Networks with dominant structure	$O(n^2)$
spinglass	Small networks, allows negative weights	$O(n^3)$
optimal	Tiny networks only (<50 nodes)	NP-hard
fluid	When k is known	$O(E k)$

Value

A tidy cograph_communities data frame with columns:

node Node label (character)

community Community assignment (integer)

Metadata stored as attributes: "algorithm", "modularity", "network" (original input), "igraph_result".

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

```
# Create a network with community structure
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Default (Louvain)
  comm <- cograph::communities(g)
  print(comm)

  # Walktrap
  comm2 <- cograph::communities(g, method = "walktrap")
  print(comm2)
}
```

community_consensus	<i>Consensus Community Detection</i>
---------------------	--------------------------------------

Description

Runs a stochastic community detection algorithm multiple times and finds consensus communities via co-occurrence matrix thresholding. This approach produces more robust and stable community assignments than single 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
method	Community detection algorithm to use. Default "louvain". Must be a stochastic method (louvain, leiden, infomap, label_propagation, spinglass).

<code>n_runs</code>	Number of times to run the algorithm. Default 100.
<code>threshold</code>	Co-occurrence threshold for consensus. Default 0.5. Nodes that appear together in $\geq$ threshold proportion of runs are placed in the same community.
<code>seed</code>	Optional seed for reproducibility. If provided, the RNG state is initialized once before repeated runs and restored on exit.
<code>...</code>	Currently ignored. Each run calls the underlying <code>igraph::cluster_*</code> () function with its own defaults; no extra arguments are forwarded.

Details

The algorithm works as follows:

1. Run the specified algorithm `n_runs` times using the current RNG stream
2. Build a co-occurrence matrix counting how often each pair of nodes appears in the same community
3. Normalize to proportions (0-1)
4. Threshold to create a consensus graph (edge if co-occurrence $\geq$ threshold)
5. Run walktrap on the consensus graph to get 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 that of the final walktrap partition of the consensus graph, not of the original network.

References

Lancichinetti, A., & Fortunato, S. (2012). Consensus clustering in complex networks. *Scientific Reports*, 2, 336.

See Also

[communities](#), [community_louvain](#)

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Consensus from 50 Louvain runs
  cc <- community_consensus(g, method = "louvain", n_runs = 50)
  print(cc)

  # Stricter threshold for more robust communities
  cc2 <- community_consensus(g, threshold = 0.7, n_runs = 100)
}
```

community_edge_betweenness

Edge Betweenness Community Detection

Description

Girvan-Newman algorithm. Iteratively removes edges with highest betweenness centrality to reveal community structure.

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 network weights, NA for unweighted.
directed	Logical; treat graph as directed? Default TRUE.
edge.betweenness	Logical; return edge betweenness values? Default TRUE.
merges	Logical; return merge matrix? Default TRUE.
bridges	Logical; return bridge edges? Default TRUE.
modularity	Logical; return modularity scores? Default TRUE.

membership	Logical; return membership vector? Default TRUE.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

Value

A cograph_communities object

A cograph_communities object. See [detect_communities](#).

References

Girvan, M., & Newman, M.E.J. (2002). Community structure in social and biological networks. *PNAS*, 99(12), 7821-7826.

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_edge_betweenness(g)
membership(comm)

net <- as_cograph(matrix(runif(25), 5, 5))
com_eb(net)
```

community_fast_greedy *Fast Greedy Community Detection*

Description

Hierarchical agglomeration using greedy modularity optimization. Produces a dendrogram of community merges.

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 network weights, NA for unweighted.
merges	Logical; return merge matrix? Default TRUE.
modularity	Logical; return modularity scores? Default TRUE.
membership	Logical; return 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 object. The full igraph communities result, including the merge dendrogram when merges = TRUE, is kept in the "igraph_result" attribute.

A cograph_communities object. See [detect_communities](#).

References

Clauset, A., Newman, M.E.J., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70, 066111.

Examples

```
g <- igraph::make_graph("Zachary")  
comm <- community_fast_greedy(g)  
membership(comm)
```

community_fluid	<i>Fluid Communities Detection</i>
-----------------	------------------------------------

Description

Simulates fluid dynamics where communities compete for nodes. Requires specifying the number of communities.

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.

`...` Passed to [to_igraph](#), whose only other argument is `directed`; anything else raises an "unused argument" error.

Value

A `cograph_communities` object

A `cograph_communities` object. See [detect_communities](#).

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

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Detect exactly 2 communities
  comm <- community_fluid(g, no.of.communities = 2)
}

m <- matrix(runif(25), 5, 5); diag(m) <- 0
net <- as_cograph(m)
com_fl(net, no.of.communities = 2)
```

community_infomap	<i>Infomap Community Detection</i>
-------------------	------------------------------------

Description

Information-theoretic community detection based on random walk dynamics. Minimizes the map equation (description length of random walks).

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

<code>x</code>	Network input
<code>weights</code>	Edge weights for transitions. NULL uses network weights, NA for unweighted.
<code>v.weights</code>	Vertex weights (teleportation weights).
<code>nb.trials</code>	Number of optimization trials. Default 10.
<code>modularity</code>	Logical; calculate modularity? Default TRUE.
<code>seed</code>	Random seed for reproducibility. Default NULL.
<code>...</code>	Passed to to_igraph , whose only other argument is directed; anything else raises an "unused argument" error.

Value

A `cograph_communities` object

A `cograph_communities` object. See [detect_communities](#).

References

Rosvall, M., & Bergstrom, C.T. (2008). Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4), 1118-1123.

Examples

```

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")
  comm <- community_infomap(g, nb.trials = 20)
}
```

community_label_propagation

Label Propagation Community Detection

Description

Fast semi-synchronous label propagation algorithm. Each node 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 network weights, NA for unweighted.
mode	For directed graphs: "out" (default), "in", or "all".
initial	Initial labels (integer vector or NULL for unique labels).
fixed	Logical vector indicating which 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` object

A `cograph_communities` object. See [detect_communities](#).

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

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Basic label propagation
  comm <- community_label_propagation(g)

  # With some nodes fixed to specific communities
  initial <- rep(NA, igraph::vcount(g))
  initial[1] <- 1 # Node 1 in community 1
  initial[34] <- 2 # Node 34 in community 2
  fixed <- !is.na(initial)
  initial[is.na(initial)] <- seq_len(sum(is.na(initial)))
  comm2 <- community_label_propagation(g, initial = initial, fixed = fixed)
}

net <- as_cograph(matrix(runif(25), 5, 5))
com_lp(net)
```

community_leading_eigenvector

Leading Eigenvector Community Detection

Description

Detects communities using the leading eigenvector of the modularity matrix. Hierarchical divisive algorithm.

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

<code>x</code>	Network input
<code>weights</code>	Edge weights. NULL uses network weights, NA for unweighted.
<code>steps</code>	Maximum number of splits. Default -1 (until modularity decreases).
<code>start</code>	Starting community structure (membership vector).
<code>options</code>	ARPACK options list. Default uses <code>igraph::arpack_defaults()</code> .
<code>callback</code>	Optional callback function called after each split.
<code>extra</code>	Extra argument passed to callback.
<code>env</code>	Environment for callback evaluation.
<code>...</code>	Passed to to_igraph , whose only other argument is directed; anything else raises an "unused argument" error.

Value

A `cograph_communities` object

A `cograph_communities` object. See [detect_communities](#).

References

Newman, M.E.J. (2006). Finding community structure using the eigenvectors of matrices. *Physical Review E*, 74, 036104.

Examples

```

g <- igraph::make_graph("Zachary")
comm <- community_leading_eigenvector(g)
membership(comm)

net <- as_cograph(matrix(runif(25), 5, 5))
com_le(net)

```

community_leiden	<i>Leiden Community Detection</i>
------------------	-----------------------------------

Description

Leiden algorithm - an improved version of Louvain that guarantees well-connected communities. Supports CPM and modularity objectives.

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 network weights, NA for unweighted.
resolution	Resolution parameter. Default 1.
objective_function	Optimization objective: "CPM" (Constant Potts Model) or "modularity". Default "CPM".
beta	Parameter for randomness in refinement step. Default 0.01.

initial_membership Initial community assignments (optional).
n_iterations Number of iterations. Default 2. Use -1 for convergence.
vertex_weights Vertex weights for 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 object
 A cograph_communities object. See [detect_communities](#).

References

Traag, V.A., Waltman, L., & van Eck, N.J. (2019). From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9, 5233.

Examples

```

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Standard Leiden
  comm <- community_leiden(g)

  # Higher resolution for more communities
  comm2 <- community_leiden(g, resolution = 1.5)

  # Modularity objective
  comm3 <- community_leiden(g, objective_function = "modularity")
}

```

community_louvain	<i>Louvain Community Detection</i>
-------------------	------------------------------------

Description

Multi-level modularity optimization using the Louvain algorithm. Fast and widely used for large networks.

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 network weights, NA for unweighted.
resolution	Resolution parameter. Higher values = 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 object

A cograph_communities object. See [detect_communities](#).

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

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")
  comm <- community_louvain(g)
  membership(comm)

  # Reproducible result with seed
  comm1 <- community_louvain(g, seed = 42)
  comm2 <- community_louvain(g, seed = 42)
  identical(membership(comm1), membership(comm2))
}
```

community_optimal	<i>Optimal Community Detection</i>
-------------------	------------------------------------

Description

Finds the optimal community structure by maximizing modularity exactly. Very slow - only use for small networks (<50 nodes).

Usage

```
community_optimal(x, weights = NULL, ...)
```

```
com_op(x, weights = NULL, ...)
```

Arguments

x	Network input
weights	Edge weights. NULL uses network weights, NA for unweighted.
...	Passed to to_igraph , whose only other argument is directed; anything else raises an "unused argument" error.

Value

A cograph_communities object

A cograph_communities object. See [detect_communities](#).

Note

This is an NP-hard problem. Use only for tiny networks.

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

```
g <- igraph::make_ring(10)
comm <- community_optimal(g)
membership(comm)

net <- as_cograph(matrix(runif(25), 5, 5))
com_op(net)
```

community_sizes	<i>Get Community Sizes</i>
-----------------	----------------------------

Description

Get Community Sizes

Usage

```
community_sizes(x)
```

Arguments

x	A cograph_communities object
---	------------------------------

Value

Integer vector of community sizes

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
community_sizes(comm)
```

community_spinglass	<i>Spinglass Community Detection</i>
---------------------	--------------------------------------

Description

Statistical mechanics approach using simulated annealing. Can handle negative edge weights.

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

<code>x</code>	Network input
<code>weights</code>	Edge weights. NULL uses network weights, NA for unweighted.
<code>vertex</code>	Vertex to find community for (single community mode). NULL for full partitioning.
<code>spins</code>	Number of spins (maximum communities). Default 25.
<code>parupdate</code>	Parallel update mode. Default FALSE.
<code>start.temp</code>	Starting temperature. Default 1.
<code>stop.temp</code>	Stopping temperature. Default 0.01.
<code>cool.fact</code>	Cooling factor. Default 0.99.
<code>update.rule</code>	Update rule: "config" (default), "random", or "simple".
<code>gamma</code>	Gamma parameter for modularity. Default 1.
<code>implementation</code>	"orig" (default) or "neg" (for negative weights).
<code>gamma.minus</code>	Gamma for negative weights in "neg" implementation.
<code>seed</code>	Random seed for reproducibility. Default NULL.
<code>...</code>	Passed to to_igraph , whose only other argument is directed; anything else raises an "unused argument" error.

Value

A `cograph_communities` object

A `cograph_communities` object. See [detect_communities](#).

References

Reichardt, J., & Bornholdt, S. (2006). Statistical mechanics of community detection. *Physical Review E*, 74, 016110.

Examples

```

g <- igraph::make_graph("Zachary")
comm <- community_spinglass(g)
membership(comm)

net <- as_cograph(matrix(runif(25), 5, 5))
com_sg(net)

```

community_walktrap	<i>Walktrap Community Detection</i>
--------------------	-------------------------------------

Description

Detects communities via random walks. Nodes within the same community tend to 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 network weights, NA for unweighted.
steps	Number of random walk steps. Default 4.
merges	Logical; return merge matrix? Default TRUE.
modularity	Logical; return modularity scores? Default TRUE.
membership	Logical; return 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` object

A `cograph_communities` object. See [detect_communities](#).

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

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")

  # Default 4 steps
  comm <- community_walktrap(g)

  # More steps for larger communities
  comm2 <- community_walktrap(g, steps = 8)
}
```

compare_communities	<i>Compare Community Structures</i>
---------------------	-------------------------------------

Description

Compares two community structures using various similarity measures.

Usage

```
compare_communities(
  comm1,
  comm2,
  method = c("vi", "nmi", "split.join", "rand", "adjusted.rand")
)
```

Arguments

comm1	First community structure (communities object or membership vector)
comm2	Second community structure (communities object or membership vector)
method	Comparison method: "vi" (variation of information), "nmi" (normalized mutual information), "split.join", "rand" (Rand index), "adjusted.rand"

Value

Numeric similarity/distance value

Examples

```

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::make_graph("Zachary")
  c1 <- community_louvain(g)
  c2 <- community_leiden(g)
  compare_communities(c1, c2, "nmi")
}

```

complement_network	<i>Complement of a Network</i>
--------------------	--------------------------------

Description

Every pair of distinct nodes that is not joined in *x* is joined in the complement, and vice versa.

Usage

```

complement_network(
  x,
  weight = 1,
  loops = FALSE,
  keep_format = FALSE,
  directed = NULL
)

```

Arguments

<i>x</i>	Network input.
<i>weight</i>	Numeric. Weight to give the new edges. Default 1. Zero is how this representation stores "no edge", so <i>weight</i> = 0 raises a <i>cograph_bad_selection</i> error rather than returning an empty network.
<i>loops</i>	Logical. Include self-loops in the complement. Default FALSE.
<i>keep_format</i>	Logical. Return the input format when TRUE.
<i>directed</i>	Logical or NULL. If NULL (default), auto-detect.

Value

A *cograph_network* holding the complement, or the input format when *keep_format* = TRUE. Directedness is preserved.

See Also

[to_undirected](#), [binarize](#)

Examples

```
adj <- matrix(c(0, 1, 0,
               1, 0, 0,
               0, 0, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

complement_network(adj)
```

contract_nodes	<i>Contract Nodes into Groups</i>
----------------	-----------------------------------

Description

Replaces each group of nodes with a single node whose edges aggregate the edges of its members. The counterpart of `igraph::contract()` and `tidygraph's to_contracted()`, and the network form of what `summarize_clusters()` computes inside an analysis object.

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 group name to node labels.
weight	How to aggregate the weights of the edges that fall between two groups: "sum" (default), "mean", "max" or "min".
loops	Logical. Keep the within-group edges as self-loops on the contracted node. Default FALSE.
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with one node per group, labeled by group name, or the input format when `keep_format = TRUE`.

See Also

[summarize_clusters](#), [detect_communities](#), [split_components](#)

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 0, 1,
               1, 0, 0, 1,
               0, 1, 1, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

contract_nodes(adj, groups = c("left", "left", "right", "right"))
```

core_periphery	<i>Detect Core-Periphery Structure</i>
----------------	--

Description

Identifies core-periphery structure in a network using either continuous (Borgatti-Everett) or discrete methods. Core nodes are densely interconnected, while periphery nodes connect primarily to the core.

Usage

```
core_periphery(
  x,
  method = c("continuous", "discrete"),
  directed = NULL,
  iter = 100,
  digits = NULL,
  ...
)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object
<code>method</code>	Character string; either "continuous" (default, Borgatti-Everett model) or "discrete" (binary core/periphery assignment).
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>iter</code>	Integer; maximum number of iterations for the continuous algorithm. Default 100.
<code>digits</code>	Integer or NULL. Round numeric outputs to this many decimal places. Default NULL (no rounding).
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

Details

Continuous method (Borgatti-Everett): Seeks a coreness vector c (rescaled to the 0-1 range) whose ideal rank-1 pattern matrix (the outer product of the vector with itself) correlates as highly as possible with the adjacency matrix. The vector is approximated by initializing from the dominant eigenvector of the adjacency matrix and refining it by power iteration until convergence or `iter` steps; the achieved correlation is reported as the "fitness" attribute rather than being optimized directly.

Discrete method: Produces a binary core / periphery assignment. Starts from the continuous solution thresholded at the median, then greedily flips the single node assignment that most improves fitness until no flip improves it. The discrete fitness being maximized is $\text{density}(\text{core}) - \text{density}(\text{periphery})$; the "fitness" attribute reported for `method = "discrete"` is the correlation between the adjacency matrix and the ideal block pattern of that assignment.

Value

A data frame with class "cograph_core_periphery", one row per node, and columns:

node Node label.

role Character: "core" or "periphery".

coreness Numeric continuous coreness score, rescaled to $[0, 1]$. Reported for both methods.

The attributes "fitness", "core_density", "periphery_density" and "network" (the original input) carry the remaining 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](https://doi.org/10.1016/S03788733(99)000192)

See Also

[centrality](#), [network_summary](#)

Examples

```
# Core-periphery in a simple network
adj <- matrix(c(
  0, 1, 1, 1, 0,
  1, 0, 1, 1, 0,
  1, 1, 0, 1, 1,
  1, 1, 1, 0, 1,
  0, 0, 1, 1, 0
), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
cp <- cograph::core_periphery(adj)
cp

# Discrete assignment
cp_disc <- cograph::core_periphery(adj, method = "discrete")
cp_disc
```

Description

Aggregates node-level network weights to cluster-level summaries. Computes both macro (cluster-to-cluster) transitions and per-cluster transitions (how nodes connect inside each 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. The function extracts the weight matrix from `x$weights` or converts via `to_matrix()`. Clusters can be auto-detected from node attributes.
- tna** A tna object from the tna package. Extracts `x$weights`.
- cluster_summary** If already a cluster_summary, returns unchanged.

clusters Cluster/group assignments for nodes. Accepts multiple formats:

- NULL** (default) Auto-detect from `cograph_network`. Looks for columns named 'clusters', 'cluster', 'groups', or 'group' in `x$nodes`. Throws an error if no cluster column is found. This option only works when `x` is a `cograph_network`.
- 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 will be derived from unique values. 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. Controls how multiple node-to-node edges are summarized: "sum" (default) Sum of all edge weights. Best for count data (e.g., transition frequencies). Preserves total flow. "mean" Average edge weight. Best when cluster sizes differ and you want to control for size. Note: when input is already a transition matrix (rows sum to 1), "mean" avoids size bias. Example: cluster with 5 nodes won't have 5x the weight of cluster with 1 node. "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 number of possible edges. Normalizes by cluster size combinations. "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. Creates transition probabilities suitable for Markov chain analysis. Interpretation: "Given I'm in cluster A, what's the probability of transitioning to cluster B?" Required for use with tna package functions. Diagonal is zero; per-cluster data is in \$clusters. "raw" No normalization. Returns aggregated counts/weights as-is. Use for frequency analysis or when you need raw counts. Compatible with igraph's contract + simplify output. "cooccurrence" Symmetrize the matrix: $(A + t(A)) / 2$. For undirected co-occurrence analysis. "semi_markov" Row-normalize with duration weighting. For semi-Markov process analysis.
directed	Logical. If TRUE (default), treat network as directed. A->B and B->A are separate edges. If FALSE, edges are undirected and the matrix is symmetrized before processing.
compute_within	Logical. If TRUE (default), compute per-cluster transition matrices for each cluster. Each cluster gets its own $n_i \times n_i$ matrix showing internal node-to-node transitions. Set to FALSE to skip this computation for better performance when only the macro (cluster-level) summary is needed.

Details

This is the core function for Multi-Cluster Multi-Level (MCML) analysis. Use [as_tna](#) to convert results to tna objects for further analysis with the tna package.

Workflow:

Typical MCML analysis workflow:

```
# 1. Create network
net <- cograph(edges, nodes = nodes)
net$nodes$clusters <- group_assignments

# 2. Compute cluster summary
cs <- csum(net, type = "tna")

# 3. Convert to tna models
tna_models <- as_tna(cs)

# 4. Analyze/visualize
plot(tna_models$macro)
tna::centralities(tna_models$macro)
```

Between-Cluster Matrix Structure:

The `macro$weights` matrix has clusters as both rows and columns:

- Off-diagonal (row *i*, col *j*): Aggregated weight from cluster *i* to cluster *j*
- Diagonal (row *i*, col *i*): Per-cluster total (sum of internal edges in cluster *i*)

When `type = "tna"`, rows sum to 1 and diagonal values represent "retention rate" - the probability of staying inside the same cluster.

Choosing method and type:

Input data	Recommended	Reason
Edge counts	<code>method="sum", type="tna"</code>	Preserves total flow, normalizes to probabilities
Transition matrix	<code>method="mean", type="tna"</code>	Avoids cluster size bias
Frequencies	<code>method="sum", type="raw"</code>	Keep raw counts for analysis
Correlation matrix	<code>method="mean", type="raw"</code>	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*. Diagonal contains aggregated intra-cluster weight (retention / self-loops). Processing depends on `type`.

inits Numeric vector of length *k*. Initial state distribution across clusters, computed from column sums of the original matrix. Represents the proportion of incoming edges to each cluster.

clusters Named list with one element per cluster. Each element is a `tna` object containing:

weights *n_i* x *n_i* matrix for nodes inside that cluster. Shows internal transitions between nodes in the same cluster.

inits Initial distribution for the cluster.

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

```
mat <- matrix(runif(100), 10, 10); diag(mat) <- 0
rownames(mat) <- colnames(mat) <- LETTERS[1:10]

# Membership vector
cs <- csum(mat, c(1,1,1,2,2,2,3,3,3,3))
cs$macro$weights      # 3x3 cluster transition matrix

# Named list of clusters, TNA-normalized
clusters <- list(Alpha = LETTERS[1:3], Beta = LETTERS[4:6], Gamma = LETTERS[7:10])
cs <- csum(mat, clusters, type = "tna")
rowSums(cs$macro$weights) # all 1 (TNA probabilities)
```

degree_distribution *Degree Distribution Visualization*

Description

Creates a histogram or cumulative distribution plot of node degrees. By default, bins are integer-aligned (one bar per degree value) so each bar maps to an exact degree.

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>mode</code>	For directed networks: "all", "in", or "out". Default "all".
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>loops</code>	Logical. If TRUE (default), keep self-loops. Set FALSE to remove them.
<code>simplify</code>	How to combine multiple edges between the same node pair. Options: "sum" (default), "mean", "max", "min", or FALSE/"none" to keep multiple edges.
<code>cumulative</code>	Logical. If TRUE, show CCDF (complementary cumulative distribution: $P(\text{degree} \geq k)$) instead of frequency. Default FALSE.
<code>breaks</code>	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).
<code>bins</code>	Integer. Approximate number of bins. Overrides bin_width. Default NULL.
<code>bin_width</code>	Numeric. Width of each bin. Default NULL (auto: 1 when the degree range is ≤ 50 , otherwise Freedman-Diaconis).
<code>normalize</code>	Logical. If TRUE, the y-axis shows proportions (bars sum to 1) instead of counts. Default FALSE.
<code>log</code>	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, with "xy" producing a log-log CCDF (standard for power-law inspection).
<code>main</code>	Character. Plot title. Default "Degree Distribution".
<code>xlab</code>	Character. X-axis label. Default "Degree".
<code>ylab</code>	Character. Y-axis label. Default auto-chosen based on normalize and cumulative.
<code>col</code>	Character. Bar/line fill color. Default "steelblue".
<code>border</code>	Character. Bar border color. Default "white".
<code>...</code>	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)).

All five components are returned for both the histogram and the cumulative plot; cumulative = TRUE only changes what is drawn.

Examples

```
# Undirected network
adj <- matrix(c(0, 1, 1, 0, 1, 0, 1, 1,
               1, 1, 0, 1, 0, 1, 1, 0), 4, 4, byrow = TRUE)
cograph::degree_distribution(adj)
cograph::degree_distribution(adj, cumulative = TRUE)

# Directed network, in-degree
directed_adj <- matrix(c(0, 1, 0, 0, 0, 0, 1, 0,
                        1, 0, 0, 1, 0, 1, 0, 0), 4, 4, byrow = TRUE)
cograph::degree_distribution(directed_adj, mode = "in")
```

detect_communities	<i>Detect Communities in a Network</i>
--------------------	--

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 method (default, fast and accurate) "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 (improved Louvain)
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: Integer 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

```
# Basic usage
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
detect_communities(adj)

# Different algorithm
detect_communities(adj, method = "walktrap")
```

disparity_filter	<i>Disparity Filter</i>
------------------	-------------------------

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

<code>x</code>	A weight matrix, tna object, cograph_network, or igraph object.
<code>level</code>	Significance level (default 0.05). Lower values result in a sparser backbone (fewer edges retained).
<code>...</code>	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:

$$p = (1 - p_{ij})^{(k_i - 1)}$$

Edges are significant if $p < level$ for either endpoint.

Value

For matrices: a binary matrix (0/1) indicating significant edges. For tna, cograph_network, and igraph objects: a tna_disparity object containing the significance matrix, original weights, filtered weights, and summary statistics.

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

```
# Create a weighted network
mat <- matrix(c(
  0.0, 0.5, 0.1, 0.0,
  0.3, 0.0, 0.4, 0.1,
  0.1, 0.2, 0.0, 0.5,
  0.0, 0.1, 0.3, 0.0
), nrow = 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("A", "B", "C", "D")

# Extract backbone at 5% significance level
backbone <- disparity_filter(mat, level = 0.05)
backbone

# More stringent filter (1% level)
backbone_strict <- disparity_filter(mat, level = 0.01)
```

dispersion	<i>Dispersion (Backstrom-Kleinberg 2014)</i>
------------	--

Description

Per-pair measure of tie strength from the Facebook relationship-inference paper. For each pair (u, v) where v is a neighbor of u :

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 $\text{embeddedness} + c == 0$).

Matches `networkx.dispersion` bit-exact for all three call modes (single pair, single source, full matrix).

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, not their labels.
- 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

```
g <- igraph::make_graph("Zachary")
# Node 0 (R index 1) to node 33 (R index 34)
dispersion(g, u = 1, v = 34)
# All pairs from node 1
head(dispersion(g, u = 1))
```

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 companion to [triad_census](#) and underlies dyad-based reciprocity.

Usage

```
dyad_census(x, directed = NULL, ...)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

Details

For *undirected* networks every present edge is counted as a mutual dyad and the asymmetric count is always zero, so the census reduces to a present/absent split. The total number of dyads is $n(n-1)/2$ regardless of direction.

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.

References

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

See Also

[triad_census](#), [edge_reciprocity](#), [network_summary](#)

Examples

```
# Directed network with a mix of mutual and asymmetric ties
adj <- matrix(c(
  0, 1, 1, 0,
  1, 0, 0, 1,
  0, 0, 0, 1,
  0, 0, 0, 0
), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- LETTERS[1:4]
cograph::dyad_census(adj)
```

edge centrality

*Calculate Edge Centrality Measures***Description**

Computes centrality measures for edges in a network and returns a tidy data frame. Unlike node centrality, these measures describe edge importance.

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

<code>x</code>	Network input (matrix, igraph, network, cograph_network, tna object)
<code>measures</code>	Which measures to calculate. Default "all" calculates all available edge measures. Options: "betweenness", "weight", "overlap", "simmelian", "reciprocity".
<code>weighted</code>	Logical. Use edge weights if available. Default TRUE.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>cutoff</code>	Maximum path length for betweenness. Default -1 (no limit).
<code>invert_weights</code>	Logical or NULL. Invert weights for path-based measures? Default NULL (auto-detect: TRUE for tna objects, FALSE otherwise).
<code>alpha</code>	Numeric. Exponent for weight inversion. Default 1.
<code>digits</code>	Integer or NULL. Round numeric columns. Default NULL.
<code>sort_by</code>	Character or NULL. Column to sort by (descending). Default NULL.
<code>...</code>	Additional arguments forwarded to the graph constructor, namely loops and simplify (see centrality).

Details

Edge measures available, with the column(s) each one adds:

betweenness Number of shortest paths passing 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

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`.

Named numeric vector of edge betweenness values (named by `"from->to"`).

Examples

```
# Create test network
mat <- matrix(c(0,1,1,0, 1,0,1,1, 1,1,0,0, 0,1,0,0), 4, 4)
rownames(mat) <- colnames(mat) <- c("A", "B", "C", "D")

# All edge measures
edge_centrality(mat)

# Just betweenness
edge_centrality(mat, measures = "betweenness")

# Sort by betweenness to find bridge edges
edge_centrality(mat, sort_by = "betweenness")
mat <- matrix(c(0,1,1,0, 1,0,1,1, 1,1,0,0, 0,1,0,0), 4, 4)
rownames(mat) <- colnames(mat) <- c("A", "B", "C", "D")
edge_betweenness(mat)
```

edge_reciprocity	<i>Edge Reciprocity</i>
------------------	-------------------------

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 (weight / reverse_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

```
adj <- matrix(c(0, 0.8, 0, 0.3, 0, 0.5, 0.7, 0, 0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
cograph::edge_reciprocity(adj, directed = TRUE)
```

ego_networks	<i>Ego-Network Metrics</i>
--------------	----------------------------

Description

Extracts the ego network of each requested node (the node, its neighbors up to a given order, and the ties among them) and reports a tidy table of personal-network metrics: size, internal tie counts and densities, and Burt’s structural-hole measures. One row per ego.

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.
order	Integer neighborhood order defining the ego network. 1 (default) is the standard ego network (ego + 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".
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

Details

effective_size and constraint are computed on the full network (Burt’s measures are defined directly from each node’s order-1 ego network), reusing the same implementations as [centrality](#) so results match centrality(x, measures = c("effective_size", "constraint")).

Value

A tidy 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 + alters).
ego_density Edge density of the ego network including ego.
alter_ties Number of edges among the alters only (excluding ego).
alter_density Edge density among the alters. Low values indicate many structural holes / brokerage opportunities.
effective_size Burt's effective size of the ego network (order = 1 only).
constraint Burt's constraint (order = 1 only).

References

Burt, R.S. (1992). *Structural Holes: The Social Structure of Competition*. Harvard University Press.

See Also

[centrality](#) (for effective_size, constraint, dispersion), [select_neighbors](#), [neighborhood_overlap](#)

Examples

```
adj <- matrix(c(
  0, 1, 1, 0, 0,
  1, 0, 1, 0, 0,
  1, 1, 0, 1, 1,
  0, 0, 1, 0, 1,
  0, 0, 1, 1, 0
), 5, 5, byrow = TRUE)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
cograph::ego_networks(adj)
```

estrada_index

Estrada Index

Description

A graph-level spectral invariant derived from subgraph centrality:

$$EE(G) = \sum_{i=1}^n e^{\lambda_i}$$

where λ_i are the eigenvalues of the adjacency matrix. The Estrada index equals the total number of closed walks in the graph, weighted by walk length: $EE(G) = \sum_k M_k/k!$ where M_k is the number of closed walks of length k . It is the sum of subgraph centralities across all nodes.

Usage

```
estrada_index(x)
```

Arguments

`x` Network input (matrix, igraph, network, cograph_network, tna object).

Details

Matches `networkx.estrada_index` at machine epsilon (max relative difference $\sim 5e-15$ across random test graphs).

Value

A single numeric value — the Estrada index of the graph.

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 equivalent (sum of `subgraph_centrality(x)` equals `estrada_index(x)`).

Examples

```
# Karate club
g <- igraph::make_graph("Zachary")
estrada_index(g)
```

extract_motifs

Extract Motifs from Network Data

Description

Extract and analyze triad motifs from network data with flexible filtering, pattern selection, and statistical significance testing. Supports both individual-level analysis (with tna objects or grouped data) and aggregate analysis (with matrices or networks). The supplied adjacency is classified as directed dyads using the 16-class MAN system.

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
)

## S3 method for class 'cograph_motif_analysis'
print(x, n = 20, ...)

```

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. If provided, x should be NULL or a matrix of node labels.
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 people have each triad, "aggregate" analyzes the summed/single network. Default depends on input: "individual" for tna or when id provided, "aggregate" otherwise.
edge_method	Method for determining edge presence: "any" Edge exists if count > 0 (simple, recommended) "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 (e.g., 1.5 means 50\ The default 1.5 is calibrated for this method. For "percent", a proportion (e.g., 0.15 for 15\ When using "percent", set this explicitly (e.g., 0.15). 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" Exclude simple sequential patterns (chains/single edges). Excludes: 003, 012, 021C. Includes stars and triangles. "closed" Network without chain patterns. Excludes: 003, 012, 021C, 120C. Similar to network but also removes mutual+chain (120C).

	"all" Include all 16 MAN types, no filtering.
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 exclusively include. If provided, only these types are returned (overrides pattern/exclude).
top	Return only the top N results (by observed count or z-score). NULL returns all results. Default NULL.
by_type	If TRUE, group results by MAN type in output. 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	Random seed for reproducibility.
n	Number of motif rows to print.
...	Passed to methods; currently unused.

Details

Both individual and aggregate significance in this legacy extractor use a directed weighted stub-matching null: positive weights retain at least one integer stub, shuffled targets preserve the integerized in/out margins, and generated loops/parallel edges are reduced to a simple loopless projection for triad classification. This differs from aggregate `motifs()`, which delegates to `motif_census()` and its simple-graph rewiring null. Observed self-loops are excluded before activity gating, counting, and null construction. The selected `edge_method` is reapplied to each null replicate, but positive fractional weights retain at least one integer stub. This preserves support while potentially changing the mass scale used by "percent"/"expected" inference. Descriptive results and the default `edge_method = "any"` are unaffected.

Value

A `cograph_motif_analysis` object (list) containing:

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 Summary counts by motif type across individuals.

params List of parameters used

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 variant (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 (has structure): 021D (out-star), 021U (in-star), 102 (mutual pair), 111D (out-star+mutual), 111U (in-star+mutual), 201 (mutual+in-star), plus all triangle patterns

Sequential patterns (chains): 012 (single edge), 021C (A->B->C chain)

Empty: 003 (no edges)

See Also

`motifs()`, `subgraphs()`, `extract_triads()`, `motif_census()`

Other motifs: `extract_triads()`, `get_edge_list()`, `motif_census()`, `motifs()`, `plot.cograph_motif_analysis()`, `plot.cograph_motifs()`, `subgraphs()`, `triad_census()`

Examples

```
# Small aggregate example -- no significance test for speed
mat <- matrix(c(0,3,2,0, 0,0,5,1, 0,0,0,4, 2,0,0,0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("Plan","Execute","Monitor","Adapt")
m <- extract_motifs(mat, significance = FALSE)
print(m)

Mod <- tna::tna(head(tna::group_regulation, 100))
# Individual-level from tna -- keep n_perm tiny for example speed
extract_motifs(Mod, top = 10, significance = TRUE, n_perm = 10L, seed = 1)
# Filter to feed-forward loops only
extract_motifs(Mod, include_types = "030T", significance = FALSE)
```

extract_triads	<i>Extract Triads with Node Labels</i>
----------------	--

Description

Extract all triads from a network, preserving node labels. This allows users to see which specific node combinations form each motif pattern.

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	Minimum edge weight for an edge to be considered present. Type is determined by edges with weight > threshold. Default 0.
min_total	Minimum total weight across all 6 edges. Excludes trivial triads with low overall activity. Default 5.
directed	Logical. Treat network as directed? Default auto-detected.

Details

This function complements `motif_census()` by showing the actual node combinations that form each motif pattern. A typical workflow is:

1. Use `motif_census()` to identify over/under-represented patterns
2. Use `extract_triads()` with type filter to see which nodes form those patterns
3. Sort by `total_weight` to find the strongest triads

Type vs Weight distinction:

- **Type** is determined by edge presence (weight > threshold)
- **Weights** are the actual frequency counts, useful for ranking triads by strength

Value

A data frame with columns:

A, B, C Node labels for the three nodes in the triad

type MAN code (003, 012, ..., 300)

weight_AB, weight_BA, weight_AC, weight_CA, weight_BC, weight_CB Edge weights (frequencies) for all 6 possible directed edges

total_weight Sum of all 6 edge weights

See Also

[motifs\(\)](#), [subgraphs\(\)](#), [motif_census\(\)](#), [extract_motifs\(\)](#)

Other motifs: [extract_motifs\(\)](#), [get_edge_list\(\)](#), [motif_census\(\)](#), [motifs\(\)](#), [plot.cograph_motif_analysis\(\)](#), [plot.cograph_motifs\(\)](#), [subgraphs\(\)](#), [triad_census\(\)](#)

Examples

```
mat <- matrix(c(0,3,2,0, 0,0,5,1, 0,0,0,4, 2,0,0,0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("Plan", "Execute", "Monitor", "Adapt")
net <- as_cograph(mat)
```

```
# All triads, feed-forward loops, triads involving "Plan"
head(extract_triads(net))
extract_triads(net, type = "030T")
extract_triads(net, involving = "Plan")
```

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, or statnet network 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

<code>x</code>	Network input: <code>cograph_network</code> , <code>matrix</code> , <code>igraph</code> , <code>network</code> , or <code>tna</code> object.
<code>...</code>	Filter expressions using any edge column (e.g., <code>weight > 0.5</code> , <code>weight > mean(weight)</code> , <code>abs(weight) > 0.3</code>).
<code>keep_isolates</code>	Logical. Keep nodes that end up with no edges? Default <code>TRUE</code> , matching <code>igraph::delete_edges()</code> and <code>tidygraph</code> : filtering edges does not remove nodes. Set <code>FALSE</code> to drop them, or call remove_isolates() afterwards.
<code>keep_format</code>	Logical. If <code>TRUE</code> , <code>matrix</code> , <code>igraph</code> , and <code>statnet</code> network inputs are returned in that format. Default <code>FALSE</code> returns <code>cograph_network</code> (universal format).
<code>directed</code>	Logical or <code>NULL</code> . If <code>NULL</code> (default), auto-detect from matrix symmetry. Set <code>TRUE</code> to force directed, <code>FALSE</code> to force undirected. Only used for non- <code>cograph_network</code> inputs.
<code>.keep_isolates</code>	Deprecated. Use <code>keep_isolates</code> .

Value

A `cograph_network` object with filtered edges. If `keep_format = TRUE`, `matrix`, `igraph`, and `statnet` network inputs are converted back to that type. Nodes are never removed by the filter itself; when the filter strands a node a `cograph_isolates_created` warning is raised.

See [filter_edges](#).

See Also

[filter_nodes](#), [splot](#), [subset_edges](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0, .5, 0, .3, .6,
               .8, .3, 0, .4, 0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Keep only strong edges
filter_edges(adj, weight > 0.5)

# Matrix in, matrix out
filter_edges(adj, weight > 0.5, keep_format = TRUE)

# Pipe-friendly with cograph_network
as_cograph(adj) |>
```

```
filter_edges(weight > 0.3) |>
filter_nodes(degree >= 2) |>
splot()
```

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, or statnet network 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 in the nodes dataframe: `id`, `label`, `name`, `x`, `y`, `inits`, `color`, plus any custom
 - 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: <code>degree >= 3, label %in% c("A", "B"), pagerank > 0.1 & degree >= 2.</code>
	On a network with negative edge weights, betweenness, closeness and pagerank are undefined: they return NA with a <code>cograph_negative_weights</code> warning.
<code>keep_edges</code>	How to handle edges. One of: "internal" (default) Keep only edges between remaining nodes "none" Remove all edges
<code>keep_format</code>	Logical. If TRUE, matrix, igraph, and statnet network inputs are returned in that format. Default FALSE returns <code>cograph_network</code> (universal format).
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected. Only used for non- <code>cograph_network</code> inputs.
<code>.keep_edges</code>	Deprecated. Use <code>keep_edges</code> .

Value

A `cograph_network` object with filtered nodes. If `keep_format = TRUE`, matrix, igraph, and statnet network inputs are converted back to that type.

See Also

[filter_edges](#), [splot](#), [subset_nodes](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0, .5, 0, .3, .6,
               .8, .3, 0, .4, 0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Keep only high-degree nodes
filter_nodes(adj, degree >= 3)

# Filter by label, combined with degree
filter_nodes(adj, degree >= 2 & label != "D")
```

`fit_degree_distribution`

Fit Statistical Distributions to Degree Sequence

Description

Fits one or more statistical distributions to the degree sequence of a network via maximum likelihood estimation and evaluates goodness-of-fit using Kolmogorov-Smirnov tests. Returns a comparison table sorted by AIC.

Usage

```
fit_degree_distribution(
  x,
  distributions = NULL,
  mode = "all",
  directed = NULL,
  xmin = NULL,
  ...
)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>distributions</code>	Character vector of distributions to fit. Options: "power_law", "exponential", "poisson", "geometric". Default NULL fits all four.
<code>mode</code>	For directed networks: "all", "in", or "out". Determines which degree to extract. Default "all".
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>xmin</code>	Minimum degree to include in fitting. For power-law, NULL triggers automatic estimation (Clauset et al. 2009 via igraph). For other distributions, NULL defaults to 1.
<code>...</code>	Additional arguments (currently unused).

Details

Power-law (Pareto Type I): $P(k) \sim k^{-\alpha}$. When igraph is available, uses `igraph::fit_power_law()` implementing the Clauset et al. (2009) method. Otherwise, computes the simple MLE: $\alpha = 1 + n / \sum \log(k/k_{min})$.

Exponential: $P(k) \sim e^{-\lambda k}$. MLE: $\lambda = 1/\bar{k}$.

Poisson: $P(k) \sim \lambda^k e^{-\lambda} / k!$. MLE: $\lambda = \bar{k}$. Note: the KS test uses a continuous approximation for a discrete distribution; p-values are approximate.

Geometric: $P(k) \sim (1-p)^k p$. MLE: $p = 1/(1 + \bar{k})$.

`ks_stat` is always reported. `ks_p` comes from `stats::ks.test()` for the exponential and Poisson fits and from `igraph::fit_power_law()` for the automatic power-law fit; it is NA for the geometric fit and for the manual (non-igraph or explicit `xmin`) power-law fit, whose KS statistics are computed directly against the theoretical CDF without a reference distribution. 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" containing:

fits Named list, one entry per distribution, each with: `distribution`, `parameters` (named list of fitted params), `loglik`, `aic`, `bic`, `ks_stat`, `ks_p`.

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 degree vector used for fitting.

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

```
adj <- matrix(c(0, 1, 1, 0, 0,
               1, 0, 1, 1, 0,
               1, 1, 0, 1, 1,
               0, 1, 1, 0, 1,
               0, 0, 1, 1, 0), 5, 5, byrow = TRUE)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
fit <- cograph::fit_degree_distribution(adj,
  distributions = c("exponential", "poisson"))
print(fit)
```

from_qgraph

Convert a qgraph object to cograph parameters

Description

Extracts the network, layout, and all relevant arguments from a qgraph object and passes them to a cograph plotting engine. Reads resolved values from graphAttributes rather than raw 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 <code>qgraph::qgraph()</code>
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 <code>show_zero_edges = TRUE</code> .
show_zero_edges	Logical. Zero is how this representation stores "no edge", so an edge whose weight rounds to zero at <code>weight_digits</code> is dropped. With TRUE such an edge is instead drawn at the smallest magnitude <code>weight_digits</code> can express, carrying its sign; every other weight is 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.
...	Override any extracted parameter. Use qgraph-style names (e.g., <code>minimum</code>) or cograph names (e.g., <code>threshold</code>).

Details**Parameter Mapping:**

The following qgraph parameters are automatically extracted and mapped to 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/Donut:

- pie values -> donut_fill with donut_inner_ratio = 0.8 and donut_empty = FALSE
- pieColor -> donut_color

Important Notes:

- **edge_color and edge_width are NOT extracted** because qgraph bakes its cut-based fading into these vectors, producing near-invisible edges. cograph applies its own weight-based styling instead.
- The cut parameter is also not passed because it causes faint edges with hanging labels.
- Layout coordinates from qgraph are preserved with rescale=FALSE.
- If you override layout, rescale is automatically re-enabled.

Value

Invisibly, a named list of cograph parameters that can be passed to `splot()` or `soplot()`.

See Also

[cograph](#) for creating networks from scratch, [splot](#) and [soplot](#) for plotting engines, [from_tna](#) for tna object conversion

Examples

```
# Convert and plot a qgraph object
adj <- matrix(c(0, .5, .3, .5, 0, .4, .3, .4, 0), 3, 3)
q <- qgraph::qgraph(adj)
from_qgraph(q) # Plots with splot

# Use soplot engine instead
from_qgraph(q, engine = "soplot")

# Override extracted parameters
from_qgraph(q, node_fill = "steelblue", layout = "circle")

# Extract parameters without plotting
params <- from_qgraph(q, plot = FALSE)
names(params) # See what was extracted

# Works with themed qgraph objects
q_themed <- qgraph::qgraph(adj, theme = "colorblind", posCol = "blue")
from_qgraph(q_themed)
```

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 with cograph. 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 <code>tna::tna()</code>
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 <code>show_zero_edges = TRUE</code> .
show_zero_edges	Logical. Zero is how this representation stores "no edge", so an edge whose weight rounds to zero at <code>weight_digits</code> is dropped. With TRUE such an edge is instead drawn at the smallest magnitude <code>weight_digits</code> can express, carrying its sign; every other weight is unchanged. Default: FALSE.
...	Additional parameters passed to the plotting engine (e.g., <code>layout</code> , <code>node_fill</code> , <code>donut_color</code>).

Details**Conversion Process:**

The tna object's transition matrix becomes edge weights, labels become node labels, and initial state probabilities (`inits`) are mapped to `donut_fill` values to visualize starting state distributions.

Directedness is read from the tna object when available; otherwise it is inferred from matrix symmetry. Transition matrices are usually directed, while symmetric co-occurrence matrices are treated as undirected.

The default `donut_inner_ratio` of 0.8 creates thin rings that effectively visualize probability values without obscuring node labels.

Parameter Mapping:

The following `tna` properties are automatically extracted:

- **weights:** Transition matrix -> edge weights
- **labels:** State labels -> node labels
- **inits:** Initial probabilities -> `donut_fill` (0-1 scale)

TNA Visual Defaults:

The following visual defaults are applied for TNA plots (all can be overridden via `...`):

- `layout = "oval"`: Oval/elliptical node arrangement
- `node_fill`: Colors from TNA palette (Accent/Set3 based on state count)
- `node_size = 7`: Larger nodes for readability
- `arrow_size = 0.61`: Prominent directional arrows for directed networks
- `edge_color = "#003355"`: Dark blue edges
- `edge_labels = TRUE`: Show transition weights on edges
- `edge_label_size = 0.4`: Readable edge labels
- `edge_label_position = 0.7`: Labels positioned toward target
- `edge_start_style = "dotted"`: Dotted line at edge source for directed networks
- `edge_start_length = 0.2`: 20% of directed edges are dotted
- `edge_label_style = "estimate"` and `edge_label_leading_zero = FALSE`: labels show the weight alone, written without a leading zero (.42, not 0.42)
- `minimum = 0.01`: transitions weaker than 0.01 are not drawn

Value

Invisibly, a named list of cograph parameters that can be passed to `splot()` or `soplot()`.

See Also

[cograph](#) for creating networks from scratch, [splot](#) and [soplot](#) for plotting engines, [from_qgraph](#) for qgraph object conversion

Examples

```
# Convert and plot a tna object
model <- tna::tna(regulation_net)
from_tna(model) # Plots with donut rings showing initial probabilities

# Use soplot engine instead
from_tna(model, engine = "soplot")

# Customize the visualization
from_tna(model, layout = "circle", donut_color = c("steelblue", "gray90"))

# Extract parameters without plotting
params <- from_tna(model, plot = FALSE)
```

```
# Modify and plot manually
params$node_fill <- "coral"
do.call(splot, params)
```

get_data

Get Original Data from Cograph Network

Description

Extracts the original estimation data stored in a `cograph_network` object. This is the raw input data (e.g., sequence matrix from tna, edge list data frame) preserved for reference.

Usage

```
get_data(x)
```

Arguments

`x` A `cograph_network` object.

Value

The original data object, or NULL if not stored.

See Also

[as_cograph](#), [get_meta](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_data(net) # NULL (matrices don't store raw data)
```

get_edges

Get Edges from Cograph Network

Description

Extracts the edges data frame from a `cograph_network` object.

Usage

```
get_edges(x)
```

Arguments

`x` A `cograph_network` object.

Value

A data frame with one row per edge and columns `from` and `to` (integer row numbers into the node table, *not* labels) and `weight`, plus any extra edge columns the network carries. An undirected network stores one row per unordered pair. Use `as.data.frame.cograph_network` or `to_df` for the same table with the endpoints given as node labels.

See Also

[as_cograph](#), [n_edges](#), [get_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_edges(net)
```

get_edge_list

Extract Raw Edge List from TNA Model

Description

Extract individual-level transition counts as an edge list from a `tna` object.

Usage

```
get_edge_list(x, by_individual = TRUE, drop_zeros = TRUE)
```

Arguments

`x` A `tna` object created by `tna::tna()`

`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 Individual identifier (only if `by_individual = TRUE`)

from Source state label

to Target state label

count Number of transitions

See Also

[extract_motifs\(\)](#) for motif analysis using edge lists

Other motifs: [extract_motifs\(\)](#), [extract_triads\(\)](#), [motif_census\(\)](#), [motifs\(\)](#), [plot.cograph_motif_analysis\(\)](#), [plot.cograph_motifs\(\)](#), [subgraphs\(\)](#), [triad_census\(\)](#)

Examples

```
Mod <- tna::tna(head(tna::group_regulation, 100))

# Get edge list by individual
edges <- get_edge_list(Mod)
head(edges)

# Aggregate across individuals
agg_edges <- get_edge_list(Mod, by_individual = FALSE)
```

get_groups

Get Node Groups from Cograph Network

Description

Extracts the node groupings from a `cograph_network` object.

Usage

```
get_groups(x)
```

Arguments

`x` A `cograph_network` object.

Value

A data frame with node groupings, or NULL if not set. The data frame has columns:

- node: Node labels
- One of layer, cluster, or group: Group assignment

See Also

[set_groups](#), [splot](#)

Examples

```
mat <- matrix(runif(25), 5, 5)
rownames(mat) <- colnames(mat) <- LETTERS[1:5]
net <- as_cograph(mat)
net <- set_groups(net, list(G1 = c("A", "B"), G2 = c("C", "D", "E")))
get_groups(net)
```

get_labels	<i>Get Labels from Cograph Network</i>
------------	--

Description

Extracts the node labels vector from a cograph_network object.

Usage

```
get_labels(x)
```

Arguments

x A cograph_network object.

Value

A character vector of node labels.

See Also

[as_cograph](#), [get_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_labels(net)
```

get_layout	<i>Get a Registered Layout</i>
------------	--------------------------------

Description

Get a Registered Layout

Usage

```
get_layout(name)
```

Arguments

name Character. Name of the layout.

Value

The layout function, or NULL if not found.

Examples

```
get_layout("circle")
```

get_meta

Get Metadata from Cograph Network

Description

Extracts the consolidated metadata list from a `cograph_network` object. The metadata contains source type, layout info, and TNA metadata.

Usage

```
get_meta(x)
```

Arguments

`x` A `cograph_network` object.

Value

A list with components:

`source` Character string indicating input type

`layout` List with layout name and seed, or NULL

`tna` List with TNA metadata (type, group_name, group_index), or NULL

See Also

[as_cograph](#), [get_source](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_meta(net)
```

get_nodes	<i>Get Nodes from Cograph Network</i>
-----------	---------------------------------------

Description

Extracts the nodes data frame from a cograph_network object.

Usage

```
get_nodes(x)
```

Arguments

x A cograph_network object.

Value

A node metadata data frame, usually with id and label columns, plus layout or other metadata columns when present.

See Also

[as_cograph](#), [n_nodes](#), [get_edges](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_nodes(net)
```

get_shape	<i>Get a Registered Shape</i>
-----------	-------------------------------

Description

Get a Registered Shape

Usage

```
get_shape(name)
```

Arguments

name Character. Name of the shape.

Value

The shape drawing function, or NULL if not found.

Examples

```
get_shape("circle")
```

get_source

Get Source Type from Cograph Network

Description

Extracts the source type string from a `cograph_network` object's metadata.

Usage

```
get_source(x)
```

Arguments

`x` A `cograph_network` object.

Value

A character string indicating the input type (e.g., "matrix", "tna", "igraph", "edgelist"), or "unknown" if not set.

See Also

[as_cograph](#), [get_meta](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
get_source(net) # "matrix"
```

get_theme	<i>Get a Registered Theme</i>
-----------	-------------------------------

Description

Get a Registered Theme

Usage

```
get_theme(name)
```

Arguments

name	Character. Name of the theme.
------	-------------------------------

Value

The theme object, or NULL if not found.

Examples

```
get_theme("classic")
```

ggplot_robustness	<i>Compare Network Robustness (ggplot2)</i>
-------------------	---

Description

Creates a ggplot2 faceted visualization comparing robustness across multiple networks. Produces publication-quality figures similar to those in Nature Scientific Reports.

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

<code>...</code>	Named arguments: network names as names, network objects as values.
<code>networks</code>	Named list of networks (alternative to ...).
<code>measures</code>	Attack strategies to compare. Default <code>c("betweenness", "degree", "random")</code> .
<code>strategy</code>	Character string; "sequential" (default) recalculates centrality after each removal, "static" uses initial centrality ranking throughout.
<code>colors</code>	Named vector of colors for measures.
<code>title</code>	Overall title. Default <code>NULL</code> .
<code>n_iter</code>	Iterations for random. Default 1000.
<code>seed</code>	Random seed. Default <code>NULL</code> .
<code>type</code>	Removal type. Default "vertex".
<code>ncol</code>	Columns in facet. Default <code>NULL</code> (auto).
<code>free_y</code>	If <code>TRUE</code> , allow different y-axis scales per facet. Default <code>FALSE</code> .

Value

A `ggplot2` object.

Examples

```
if (requireNamespace("igraph", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE)) {

  g1 <- igraph::sample_pa(40, m = 2, directed = FALSE)
  g2 <- igraph::sample_gnp(40, 0.15)

  ggplot_robustness(
    "Teaching network" = g1,
    "Collaborative network" = g2,
    n_iter = 20
  )
}
```

group_centrality

Group Centrality (Everett-Borgatti 1999)

Description

Group centrality measures the importance of a *set* of nodes $C \subseteq V$ rather than a single node. Three variants are supported:

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", "closeness", "degree".
mode	For directed graphs with measure = "degree": "all" (both directions), "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 (matching NetworkX convention). For directed graphs, cograph uses $d(v, c)$ in the original direction, equivalent to NetworkX's "reverse then multi-source".

degree $GDC(C) = |N(C) \setminus C| / (|V| - |C|)$, the fraction of non-group nodes adjacent to at least one group member. mode = "in" / "out" pick the corresponding directed neighborhood.

Value

A single numeric scalar — the group centrality of the set nodes.

Divergence from NetworkX on betweenness

`networkx.group_betweenness centrality` uses the Puzis-Elovici-Dolev iterative algorithm, which produces results that diverge from the textbook Everett-Borgatti / Puzis 2007 "at least one node in C " definition on some graph topologies (verified via an independent Python brute-force). `cograph` implements the textbook formula directly; `group_closeness` and `group_degree` match NetworkX exactly.

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

```
g <- igraph::make_graph("Zachary")
group_centrality(g, nodes = c(1, 2, 3), measure = "betweenness")
group_centrality(g, nodes = c(1, 2, 3), measure = "closeness")
group_centrality(g, nodes = c(1, 2, 3), measure = "degree")
```

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 vs fine-grained codes) and split by actor (Human, AI, or both combined). 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

A data.frame where each row is one session and each column is one time step. Every column is named T1, T2, ... Tn and holds the action code at that step, with NA indicating the session ended before that time step; there are no identifier columns. Six variants are provided: coding (human actions by category, 9 states), coding_detailed (human actions by fine-grained code, 15 states), ai_coding (AI actions by category, 8 states), ai_detailed (AI actions by fine-grained code, 18 states), human_ai (both actors by category, 17 states), and human_ai_detailed (both actors by fine-grained code, 32 states).

Source

Human-AI programming interaction study, 34 projects, 429 sessions.

Examples

```
data(coding)
str(coding, list.len = 6)
dim(coding)
```

invert_weights	<i>Invert Edge Weights (Similarity to Distance and Back)</i>
----------------	--

Description

Turns strong ties into short distances, which is what path-based measures need when the weights are similarities rather than costs.

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; requires non-zero weights, which every stored edge has. "max_minus" $\max(w) - w$. The strongest edge becomes zero and is therefore dropped; a <code>cograph_edges_dropped</code> warning says how many. "reflect" $\max(w) + \min(w) - w$. Reverses the order of the weights while keeping every edge, so no edge is lost.
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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

invert_weights(adj)
invert_weights(adj, method = "reflect")
```

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 BFS-based two-coloring algorithm.

Value

Logical. TRUE if the matrix could represent a bipartite network, FALSE otherwise.

Examples

```
# Non-square matrix is bipartite
inc <- matrix(c(1, 0, 1, 1, 1, 0), 2, 3)
cograph::is_bipartite(inc)

# Square bipartite-compatible adjacency
adj <- matrix(c(0, 0, 1, 1,
               0, 0, 1, 0,
               1, 1, 0, 0,
               1, 0, 0, 0), 4, 4, byrow = TRUE)
cograph::is_bipartite(adj)

# Non-bipartite (triangle)
tri <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
cograph::is_bipartite(tri)
```

is_directed	<i>Check if Network is Directed</i>
-------------	-------------------------------------

Description

Checks whether a `cograph_network` is directed.

Usage

```
is_directed(x)
```

Arguments

`x` A `cograph_network` object.

Value

Logical: TRUE if directed, FALSE if undirected.

See Also

[as_cograph](#)

Examples

```
# Symmetric matrix -> undirected
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
cograph::is_directed(net) # FALSE

# Asymmetric matrix -> directed
mat2 <- matrix(c(0, 1, 0, 0, 0, 1, 0, 0, 0), nrow = 3)
net2 <- as_cograph(mat2)
cograph::is_directed(net2) # TRUE
```

is_tna_network	<i>Check if Network is TNA-based</i>
----------------	--------------------------------------

Description

Checks whether a `cograph_network` was created from a `tna` or `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.

See Also

[as_cograph](#)

Examples

```
# Non-TNA network
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
is_tna_network(net) # FALSE

model <- tna::tna(regulation_net)
net_tna <- as_cograph(model)
is_tna_network(net_tna) # TRUE
```

k_shortest_paths	<i>Find K Shortest Loopless Paths (Yen's Algorithm)</i>
------------------	---

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.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

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](https://doi.org/10.1287/mnsc.17.11.712)

See Also

[shortest_paths](#)

Examples

```
# Find 3 shortest paths in a small network
adj <- matrix(c(
  0, 1, 1, 0, 0,
  0, 0, 1, 1, 0,
  0, 0, 0, 1, 1,
  0, 0, 0, 0, 1,
  0, 0, 0, 0, 0
), 5, 5, byrow = TRUE)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
kp <- cograph::k_shortest_paths(adj, from = "A", to = "E", k = 3)
kp
```

layer_degree_correlation	<i>Degree Correlation Between Layers</i>
--------------------------	--

Description

Measures hub consistency across layers via degree correlation.

Usage

```
layer_degree_correlation(layers, mode = c("total", "in", "out"))  
  
ldegcor(layers, mode = c("total", "in", "out"))
```

Arguments

layers	List of adjacency matrices
mode	Degree type: "total", "in", "out"

Value

Correlation matrix between layer degree sequences

Examples

```
mat1 <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)  
mat2 <- matrix(c(0, 0, 1, 1, 0, 0, 0, 1, 0), 3, 3)  
layers <- list(L1 = mat1, L2 = mat2)  
layer_degree_correlation(layers, mode = "total")
```

layer_similarity	<i>Layer Similarity</i>
------------------	-------------------------

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
method	Comparison method: "jaccard" (default), "overlap", "hamming", "cosine" or "pearson"

Details

"jaccard", "overlap" and "hamming" compare edge *presence* ($A > 0$) and therefore ignore weights; "cosine" and "pearson" are computed on the raw cell values. The two matrices must have identical dimensions.

Value

A single numeric value. All methods except "hamming" return a similarity (higher = more alike); "hamming" returns a *distance* - the number of matrix cells whose edge presence differs between the two layers - so lower means more alike 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

```
A1 <- matrix(c(0,1,1,0, 1,0,0,1, 1,0,0,1, 0,1,1,0), 4, 4)
A2 <- matrix(c(0,1,0,0, 1,0,1,0, 0,1,0,1, 0,0,1,0), 4, 4)

layer_similarity(A1, A2, "jaccard") # Edge overlap
layer_similarity(A1, A2, "cosine")  # Weight similarity
```

layer_similarity_matrix
<i>Pairwise Layer Similarities</i>

Description

Computes similarity matrix for all pairs of layers.

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". Note that "hamming", accepted by layer_similarity , is <i>not</i> available here because it is a distance rather than a similarity.

Value

A symmetric L x L matrix of pairwise similarities with the layer names as dimnames and 1 on the diagonal.

Examples

```
nodes <- c("A", "B", "C")
t1 <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3, dimnames = list(nodes, nodes))
t2 <- matrix(c(0, 1, 1, 1, 0, 0, 1, 0, 0), 3, 3, dimnames = list(nodes, nodes))
layers <- list(T1 = t1, T2 = t2)

layer_similarity_matrix(layers, "cosine")
layer_similarity_matrix(layers, "jaccard")
```

layout_circle	<i>Circular Layout</i>
---------------	------------------------

Description

Arrange nodes evenly spaced around a circle.

Usage

```
layout_circle(network, order = NULL, start_angle = pi/2, clockwise = TRUE, ...)
```

Arguments

network	A CographNetwork or cograph_network object.
order	Optional vector specifying node order (indices or labels).
start_angle	Starting angle in radians (default: pi/2 for top).
clockwise	Logical. Arrange nodes clockwise? Default TRUE.
...	Additional arguments (ignored).

Value

Data frame with x, y coordinates.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- CographNetwork$new(adj)
coords <- layout_circle(net)
```

layout_groups	<i>Group-based Layout</i>
---------------	---------------------------

Description

Arrange nodes based on group membership. Groups are positioned in a circular arrangement around the center, with nodes within each group also arranged in a circle.

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 specifying group membership for each node. Can be numeric, character, or factor.
- group_positions Optional list or data frame with x, y coordinates for each group center.
- inner_radius Radius of nodes within each group (default: 0.15).
- outer_radius Radius for positioning group centers (default: 0.35).

Value

Data frame with x, y coordinates.

Examples

```
# Create a network with groups
adj <- matrix(0, 9, 9)
adj[1, 2:3] <- 1; adj[2:3, 1] <- 1 # Group 1
adj[4, 5:6] <- 1; adj[5:6, 4] <- 1 # Group 2
adj[7, 8:9] <- 1; adj[8:9, 7] <- 1 # Group 3
net <- CographNetwork$new(adj)
groups <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
coords <- layout_groups(net, groups)
```

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 (indices or labels).
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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)  
net <- CographNetwork$new(adj)  
coords <- layout_oval(net, ratio = 1.5)
```

layout_saqr	<i>Saqr Layout (Start/End transition flow)</i>
-------------	--

Description

Port of the Dynalytics Desktop "saqr" layout (Saqr et al., LAK25). Designed for directed transition networks: the Start node sits alone on the top row, the End node (if present) alone on the bottom row, and every other node is ranked by its outgoing weight from Start (strongest connections nearest Start) and split into 2 middle rows (≤ 10 middle nodes) or 3 (> 10). A sine envelope narrows the rows near Start/End for a lens-shaped silhouette, and the first middle row is zig-zag jittered.

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"). Falls back to the highest out-degree node when the label is not found.
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).

Details

If the start label is absent the highest out-degree node is used. The End row is only drawn when the end label is present.

Value

Data frame with x, y coordinates, one row per node.

Examples

```
adj <- matrix(0, 5, 5,
  dimnames = list(c("Start", "A", "B", "C", "End"),
    c("Start", "A", "B", "C", "End")))
adj["Start", "A"] <- 5; adj["Start", "B"] <- 3; adj["Start", "C"] <- 1
adj["A", "End"] <- 2; adj["B", "End"] <- 4; adj["C", "End"] <- 1
net <- CographNetwork$new(adj, directed = TRUE)
layout_saqr(net)
```

layout_spring	<i>Fruchterman-Reingold Spring Layout</i>
---------------	---

Description

Compute node positions using the Fruchterman-Reingold force-directed algorithm. Nodes connected by edges are attracted to each other while all nodes repel each other.

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	Rate of temperature decrease for exponential cooling (default: 0.95).
repulsion	Repulsion constant (default: 1.5).
attraction	Attraction constant (default: 1).
seed	Random seed for reproducibility.
initial	Optional initial coordinates (matrix or data frame). For animations, pass the previous frame's layout to ensure smooth transitions.
max_displacement	Maximum distance a node can move from its initial position (default: NULL = no limit). Useful for animations to prevent large jumps between frames. Values like 0.05-0.1 work well.
anchor_strength	Strength of force pulling nodes toward initial positions (default: 0). Higher values (e.g., 0.5-2) keep nodes closer to their starting positions. Only applies when initial is provided.

area	Area parameter controlling node spread (default: 1.5). Higher values spread nodes further apart.
gravity	Gravity force pulling nodes toward center (default: 0). Higher values (e.g., 0.5-2) prevent nodes from drifting apart.
init	Initialization method: "random" (default) or "circular".
cooling_mode	Cooling schedule: "exponential" (default, uses cooling parameter), "vcf" (Variable Cooling Factor - adapts based on movement), or "linear" (linear decrease over iterations).
...	Additional arguments (ignored).

Value

Data frame with x, y coordinates.

Examples

```
adj <- matrix(c(0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0), nrow = 4)
net <- CographNetwork$new(adj)
coords <- layout_spring(net, seed = 42)

# For animations: use previous layout as initial with constraints
coords2 <- layout_spring(net, initial = coords, max_displacement = 0.05)

# With gravity to keep nodes centered
coords3 <- layout_spring(net, gravity = 0.5, area = 2, seed = 42)

# With circular initialization and VCF cooling
coords4 <- layout_spring(net, init = "circular", cooling_mode = "vcf", seed = 42)
```

layout_target	<i>Target Layout (focal-node, topological)</i>
---------------	--

Description

Port of qgraph’s flow() layout. One node of interest (the target) is placed alone, then every other node is drawn in successive levels ordered by unweighted graph distance (BFS hops) from it. This shows how the target node connects out into 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 highest-degree node is used.
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

Unlike qgraph’s implementation, weights are binarized for layering (only connectivity matters) and disconnected nodes are placed in an extra trailing level instead of raising an error.

Value

Data frame with x, y coordinates, one row per node.

Examples

```
adj <- matrix(c(0, 1, 1, 0, 1, 0, 0, 1,
               1, 0, 0, 0, 0, 1, 0, 0), nrow = 4, byrow = TRUE)
net <- CographNetwork$new(adj)
layout_target(net, target = 1)
```

list_centralities	<i>Catalogue of the Centrality Measures</i>
-------------------	---

Description

A tidy table of every measure [centrality](#) can compute, with the facts you need before you read a column of results: which end of the scale marks a prominent node, whether the measure needs a community partition, whether it reads edge weights, and whether it is held back from type = "all" because its cost grows steeply.

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

Twelve measures are oriented so that a **low** value marks the more central node, and sorting their column the usual way puts the periphery on top. Filter with orientation = "lower" to see 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

```
# Every measure, with the facts needed to read its column
head(list_centralities())

# The measures where a low value marks the more central node
list_centralities(orientation = "lower")

# The measures held back from type = "all"
list_centralities(costly = TRUE)
```

list_layouts	List Available Layouts
--------------	------------------------

Description

List Available Layouts

Usage

```
list_layouts()
```

Value

Character vector of registered layout names.

Examples

```
list_layouts()
```

list_palettes	<i>List Available Color Palettes</i>
---------------	--------------------------------------

Description

Returns the names of all registered color palettes.

Usage

```
list_palettes()
```

Value

Character vector of palette names.

Examples

```
list_palettes()
```

list_shapes	<i>List Available Shapes</i>
-------------	------------------------------

Description

List Available Shapes

Usage

```
list_shapes()
```

Value

Character vector of registered shape names.

Examples

```
list_shapes()
```

list_svg_shapes	<i>List Registered SVG Shapes</i>
-----------------	-----------------------------------

Description

Get names of all registered custom SVG shapes.

Usage

```
list_svg_shapes()
```

Value

Character vector of registered shape names.

Examples

```
list_svg_shapes()
```

list_themes	<i>List Available Themes</i>
-------------	------------------------------

Description

List Available Themes

Usage

```
list_themes()
```

Value

Character vector of registered theme names.

Examples

```
list_themes()
```

membership	<i>Get Community Membership</i>
------------	---------------------------------

Description

Extracts a named membership vector from a communities result. Works with both `cograph_communities` data frames and `igraph communities` objects.

Usage

```
membership(x)
```

Arguments

`x` A `cograph_communities` or `igraph communities` object.

Value

Named integer vector of community assignments.

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
membership(comm)
```

motifs	<i>Network Motif Analysis</i>
--------	-------------------------------

Description

Two modes of directed MAN triad analysis for networks:

- **Census** (`named_nodes = FALSE`, default): Counts MAN type frequencies with significance testing. Nodes are exchangeable.
- **Instances** (`named_nodes = TRUE`, or use `subgraphs()`): Lists specific node triples forming each pattern. Nodes are NOT exchangeable.

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'
print(x, ...)

## 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,
  ...
)

```

Arguments

x Input data: a tna object, cograph_network, matrix, igraph, or data.frame (edge list).

named_nodes	Logical. If FALSE (default), performs census (type-level counts). If TRUE, extracts specific node triples (instance-level). <code>subgraphs()</code> is a convenience wrapper that sets this to TRUE.
actor	Character. Column name in the edge list metadata to group by. If NULL (default), auto-detects standard column names (<code>session_id</code> , <code>session</code> , <code>actor</code> , <code>user</code> , <code>participant</code>). If no grouping column found, performs aggregate analysis.
window	Numeric. Window size for windowed analysis. Splits each actor's transitions into windows of this size. NULL (default) means no windowing.
window_type	Character. Window type: "rolling" (default) or "tumbling". Only used when window is set.
pattern	Which MAN triad types to include in the analysis: "triangle" (default) Only the 7 closed triangle types: 030C, 030T, 120C, 120D, 120U, 210, 300. Excludes trivial open patterns (empty triads, single edges, chains, stars, mutual pairs). "network" All types except trivially open ones. Excludes 003 (empty), 012 (single edge), 021C (chain). "closed" Like "network" but also excludes 120C (mixed regulated). Excludes 003, 012, 021C, 120C. "all" All 16 MAN types, including empty and trivial patterns.
include	Character vector of MAN types to include exclusively. Overrides pattern and exclude.
exclude	Character vector of MAN types to exclude. Applied after pattern filter.
significance	Logical. Run permutation significance test? Default TRUE.
n_perm	Number of permutations for significance. When <code>significance = TRUE</code> , 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 is the only setting that reproduces results from earlier versions: it consumes a single RNG stream in replicate-then-unit order, so a given seed gives the historical numbers. <code>cores > 1</code> gives each replicate its own L'Ecuyer-CMRG stream, which makes a result depend on seed alone and not on the worker count or on how replicates were chunked – but those are a <i>different</i> set of draws, so the p-values will not match a <code>cores = 1</code> run of the same seed. They remain a valid permutation null, and repeated parallel runs of one seed agree exactly with each other at any cores. Forking is used where available; Windows uses a PSOCK cluster. Only the individual-level census null is parallelized. Values above <code>parallel::detectCores()</code> are capped with a <code>cograph_cores_capped</code> warning.
min_count	Inclusive minimum count to keep a row — rows with count $\geq$ min_count are retained. In instance mode (<code>named_nodes = TRUE</code>) this filters the observed column: at individual level the number of subjects exhibiting the triad, at aggregate level the triad's weighted edge mass (sum of its 6 directed edge weights). In census mode (<code>named_nodes = FALSE</code>) this filters the count column — the number of times each MAN type appears. Default 5 for instances, NULL for census (no filter).

edge_method	Method for determining edge presence: "any" (default; any positive edge), "expected" (observed/expected ratio), 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 transitions for a unit to be included. Default 5.
top	Return only the top N results. NULL returns all.
seed	Random seed for reproducibility.
...	Additional arguments passed to internal plot helpers.
type	Plot type: "triads" Network diagrams of specific node triples (instance mode) or falls back to patterns (census mode). Instance panels use a canonical representative of the MAN class: concrete labels identify participants, not their observed node-role orientation. Each panel title reads "<MAN code>: <description>" (e.g. "030T: Feed-forward") and, in census mode, appends the z-score and a significance star (* p<.05, ** p<.01, *** p<.001). Arranged in a grid. "types" Bar chart of MAN type frequencies. In census mode bars are colored by significance direction (see colors); in instance mode bars use a single fill because per-type significance would need an aggregation rule across multiple node-triple rows of the same type. "significance" Z-score bars per row of x\$results. In census mode each bar is one MAN type; in instance mode each bar is one concrete node-triple, labeled "<triple> [<MAN code>: <description>]". Bars are colored with the same three-tone rule (see colors). Requires significance = TRUE in the motifs() call. "patterns" Abstract MAN pattern diagrams showing the edge structure of each triad type. In census mode panel nodes are filled by significance direction (red sig over / blue sig under / grey ns); in instance mode panels use a single fill, same reason as "types".
n	Maximum number of items to plot. Default 15.
ncol	Number of columns in the triad/pattern grid. Default 5.
colors	Two-element color vector mapped to a three-tone significance scale (used by type = "significance", plus type = "types" and type = "patterns" in census mode): colors[1] fills items that are significantly under-represented (p < .05 and z < 0); colors[2] fills items that are significantly over-represented (p < .05 and z > 0); everything else is filled neutral grey ("9E9E9E"). Default c("#2166AC", "#B2182B") (blue for under, red for over). When significance was not run, type = "types" falls back to a single colors[1] fill and patterns nodes use colors[1].
node_size	Triad node radius (relative). Default 5. (type = "triads" only.)
label_size	Triad node-label font size in points. Default 11.
title_size	Per-panel title font size in points. Default 12.

stats_size	Per-panel statistics caption font size in points (e.g., $n=34$ $z=-55.3$ $p<.001$). Default 13.
legend_size	Bottom legend font size in points. Default 13.
legend	Logical. Show the abbreviation legend strip below the triad grid. Default TRUE. (type = "triads" only.)
motif_color	Color of triad nodes/edges/labels. Default "#800020" (deep burgundy). (type = "triads" only.)
spacing	Triangle spread inside each panel; > 1 pulls nodes inward, < 1 pushes them apart. Default 1.
base_size	Base font size for the ggplot2 themes used by type = "types" and type = "significance". Default 12.
combined	Logical: when TRUE (default) and type = "patterns" (or type = "triads" on unnamed-node input that falls back to pattern plotting), arrange the per-motif panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw into a layout the caller has already configured (e.g. via <code>panel_layout()</code>).

Details

Detects input type and analysis level automatically. For inputs with individual/group data (tna objects, cograph networks from edge lists with metadata), performs per-group analysis. For aggregate inputs (matrices, igraph), analyzes the single network. The unified motifs() and subgraphs() APIs classify the supplied adjacency as directed dyads in the 16-class MAN system. For the separate four-class undirected census, use motif_census(..., directed = FALSE).

For aggregate inputs, significance delegates to motif_census() and its loop-free simple-graph rewiring null. Individual weighted inputs use a directed stub-matching null: positive edge weights are converted to at least one integer stub, target stubs are shuffled while preserving each unit's integerized in/out margins, and the resulting multigraph (which may contain loops or parallel edges) is evaluated through its simple loopless triad projection. Observed self-loops are excluded before both counting and null construction.

With edge_method = "percent", edge presence is computed within each node triple: an edge's weight is divided by the sum of the six possible directed edge weights for that triple. A threshold above 1 is interpreted as a percentage (for example, 1.5 means 1.5 percent); a threshold at or below 1 is interpreted as a proportion.

Non-"any" significance has three important boundaries. For aggregate census input, observed counts use the selected threshold but the delegated null tests the unthresholded network; the function emits a warning. For individual census input, the threshold is reapplied to each integerized stub-null replicate. For individual named-instance input, the optimized null classifies raw stub presence and therefore does not reapply edge_method/edge_threshold. In all weighted individual paths, positive fractional weights retain at least one stub, which preserves support but can change the mass scale used by "percent"/"expected". These limitations do not affect descriptive results with significance = FALSE or the default edge_method = "any".

Value

A cograph_motif_result object (a list) with:

results Data frame of results. Census mode (`named_nodes = FALSE`): one row per retained, observed MAN type with columns `type`, `count`, and when `significance = TRUE` also expected, `z`, `p`, `sig`. Instance mode (`named_nodes = TRUE`): one row per concrete node-triple and MAN type with columns `triad`, `node1`, `node2`, `node3`, `type`, `observed`, and when `significance = TRUE` also expected, `z`, `p`, `sig`. At individual level, `observed` is the number of sessions/units in which that triple has that MAN type; one triple may therefore occupy multiple rows when its type differs across units.

type_summary Named table of MAN-type counts. In census mode the values come from the `count` column; in instance mode they come from `table(results$type)` and describe how many concrete node-triples fall under each MAN type. Sorted descending so `plot(., type = "patterns")` draws the most frequent types first.

level Analysis level: "individual" when the input carried per-subject sequence data (tna with `$data`, edge list with an `actor` column, `Nestimate` netobject built from `build_tna()`/similar), otherwise "aggregate" (a single transition matrix).

named_nodes Logical mirror of the `named_nodes` argument. Plot helpers gate per-type significance decoration on this so the instance-mode case (multiple triples per MAN type) doesn't get silently aggregated.

n_units Number of subjects/units. 1 at aggregate level, `nrow` of the input sequence data at individual level.

params List of the call's parameters (`pattern`, `edge_method`, `edge_threshold`, `significance`, `n_perm`, `min_count`, `labels`, `n_states`, and the window settings if any). Read by `print()` and the `plot()` dispatcher.

Invisibly returns the input `x` for "triads" and "patterns", or the underlying `ggplot` for "types" and "significance".

See Also

`subgraphs()`, `motif_census()`, `extract_motifs()`

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `plot.cograph_motif_anal`, `plot.cograph_motifs()`, `subgraphs()`, `triad_census()`

Examples

```
# Census from a matrix (no significance test -- fastest path)
mat <- matrix(c(0,3,2,0, 0,0,5,1, 0,0,0,4, 2,0,0,0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("Plan","Execute","Monitor","Adapt")
motifs(mat, significance = FALSE)
```

```
# With a minimal significance test (set n_perm >= 500 in practice)
motifs(mat, n_perm = 10L, seed = 1)
```

```
Mod <- tna::tna(head(tna::group_regulation, 100))
motifs(Mod, n_perm = 10L, seed = 1)
subgraphs(Mod, n_perm = 10L, seed = 1)
```

motif_census

*Network Motif Analysis***Description**

Analyze recurring subgraph patterns (motifs) in networks and test their statistical significance against null models.

Usage

```
motif_census(
  x,
  size = 3,
  n_random = 100,
  method = c("configuration", "gnm"),
  directed = NULL,
  seed = NULL
)

## S3 method for class 'cograph_motifs'
print(x, ...)
```

Arguments

<code>x</code>	A matrix, igraph object, or cograph_network
<code>size</code>	Motif size: 3 (triads) or 4 (tetrads). Default 3.
<code>n_random</code>	Number of random networks for the null model. Must be a whole number of at least 2. Default 100.
<code>method</code>	Null model method: "configuration" (preserves degree) or "gnm" (preserves edge count). Default "configuration".
<code>directed</code>	Logical. Treat as directed? Default auto-detected.
<code>seed</code>	Random seed for reproducibility. Default NULL. When supplied, the caller's RNG state is saved and restored.
<code>...</code>	Passed to methods; currently unused.

Value

A cograph_motifs data frame with one row per motif class and columns:

motif Motif class name (the 16 MAN codes for directed triads, the four undirected triad classes, or motif_<i> labels for size 4).

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; NA when the null is degenerate (null_sd = 0) and the observation differs from it.

p_value Two-sided empirical (add-one corrected) permutation p-value, not a Gaussian approximation.

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. Self-loops and multiple edges are removed before counting.

See Also

[motifs\(\)](#) for the unified API, [extract_motifs\(\)](#) for detailed triad extraction, [plot.cograph_motifs\(\)](#) for plotting

Other motifs: [extract_motifs\(\)](#), [extract_triads\(\)](#), [get_edge_list\(\)](#), [motifs\(\)](#), [plot.cograph_motif_analysis\(\)](#), [plot.cograph_motifs\(\)](#), [subgraphs\(\)](#), [triad_census\(\)](#)

Examples

```
# Create a directed network
mat <- matrix(c(
  0, 1, 1, 0,
  0, 0, 1, 1,
  0, 0, 0, 1,
  1, 0, 0, 0
), 4, 4, byrow = TRUE)

# Analyze triadic motifs
m <- motif_census(mat)
print(m)
plot(m)
```

mutate_edges

Add or Change Edge Attributes

Description

Add or Change Edge Attributes

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 <code>strong = abs_weight > 0.5</code> or <code>scaled = weight / max(weight)</code> .
community	Community detection method used when an expression refers to <code>same_community</code> , <code>from_community</code> or <code>to_community</code> . 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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

as.data.frame(mutate_edges(adj, strong = weight > 0.5))
```

mutate_nodes	<i>Add or Change Node Attributes</i>
--------------	--------------------------------------

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

<code>x</code>	Network input.
<code>...</code>	Named expressions, for example <code>hub = degree > 3</code> or <code>score = pagerank * 100</code> . Available names are the existing node columns plus every measure and predicate listed under select_nodes .
<code>keep_format</code>	Logical. Return the input format when TRUE. Note that only <code>igraph</code> and <code>cograph_network</code> formats can carry node attributes; a matrix cannot, and the new columns are lost.
<code>directed</code>	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

```
adj <- matrix(c(0, 1, 1, 1,
               1, 0, 1, 0,
               1, 1, 0, 0,
               1, 0, 0, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

as.data.frame(mutate_nodes(adj, deg = degree, hub = degree >= 3),
              what = "nodes")
```

`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 (if weighted), overlap, shared_neighbors.

See Also

[edge centrality](#), [simmelian strength](#)

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
cograph::neighborhood_overlap(adj)
```

network_bridges	<i>Bridge Edges</i>
-----------------	---------------------

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, data frame with from/to columns. If count_only = TRUE, integer count.

Examples

```
# Two triangles connected by single edge
adj <- matrix(0, 6, 6)
adj[1,2] <- adj[2,1] <- adj[1,3] <- adj[3,1] <- adj[2,3] <- adj[3,2] <- 1
adj[4,5] <- adj[5,4] <- adj[4,6] <- adj[6,4] <- adj[5,6] <- adj[6,5] <- 1
adj[3,4] <- adj[4,3] <- 1 # Bridge
network_bridges(adj) # Edge 3-4
network_bridges(adj, count_only = TRUE) # 1
```

network_clique_size	<i>Largest Clique Size</i>
---------------------	----------------------------

Description

Finds the size of the largest clique (complete subgraph) in the network. Also known as 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

Integer: size of the largest clique

Examples

```
# Triangle embedded in larger graph
adj <- matrix(c(0,1,1,1, 1,0,1,0, 1,1,0,0, 1,0,0,0), 4, 4)
network_clique_size(adj) # 3
```

network_cut_vertices	<i>Cut Vertices (Articulation Points)</i>
----------------------	---

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, vector of node indices (or names if graph is named). If count_only = TRUE, integer count.

Examples

```
# Bridge node connecting two components
adj <- matrix(c(0,1,1,0,0, 1,0,1,0,0, 1,1,0,1,0, 0,0,1,0,1, 0,0,0,1,0), 5, 5)
network_cut_vertices(adj) # Node 3 is cut vertex
network_cut_vertices(adj, count_only = TRUE) # 1
```

network_girth	<i>Network Girth (Shortest Cycle Length)</i>
---------------	--

Description

Computes the girth of a network - the length of the shortest cycle. Returns Inf for acyclic graphs (trees, DAGs).

Usage

```
network_girth(x, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
...	Passed to <code>to_igraph</code> , whose only other argument is directed; anything else raises an "unused argument" error.

Value

Integer: length of shortest cycle, or Inf if no cycles exist

Examples

```
# Triangle has girth 3
triangle <- matrix(c(0,1,1, 1,0,1, 1,1,0), 3, 3)
network_girth(triangle) # 3

# Tree has no cycles (Inf)
tree <- matrix(c(0,1,0, 1,0,1, 0,1,0), 3, 3)
network_girth(tree) # Inf
```

network_global_efficiency	<i>Global Efficiency</i>
---------------------------	--------------------------

Description

Computes the global efficiency of a network - the average of the inverse shortest path lengths between all pairs of nodes. Higher values indicate better global communication efficiency. Handles disconnected graphs gracefully (infinite distances contribute 0).

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	Edge weights (NULL for unweighted). Set to NA to ignore existing weights.

invert_weights	Logical or NULL. Invert weights so higher weights = shorter paths? Default NULL which auto-detects: TRUE for tna objects, FALSE otherwise (matching igraph/sna). Set TRUE for strength/frequency weights (qgraph style).
alpha	Numeric. Exponent for weight inversion: distance = 1/weight ^{alpha} . Default 1.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

Value

Numeric global efficiency. For unweighted simple graphs this is in $[0, 1]$; weighted graphs can exceed 1 when edge distances are below 1.

Examples

```
# Complete graph has efficiency 1
k4 <- matrix(1, 4, 4); diag(k4) <- 0
network_global_efficiency(k4) # 1

# Star has lower efficiency
star <- matrix(c(0,1,1,1, 1,0,0,0, 1,0,0,0, 1,0,0,0), 4, 4)
network_global_efficiency(star) # 0.75
```

network_local_efficiency
Local Efficiency

Description

Computes the average local efficiency across all nodes, delegating to `igraph::average_local_efficiency()`. `igraph` removes the node and measures the distances between its neighbors *through the rest of the network*, so the value can exceed the one Latora & Marchiori (2001) define, which restricts those distances to the subgraph induced on the neighbors. `centrality(x, measures = "local_efficiency")` reports the induced-subgraph form, matching `networkx`, `brainGraph` and the Brain Connectivity Toolbox. Both measure fault tolerance and local integration; the two agree whenever the neighbors have no detour available.

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	Edge weights (NULL for unweighted). Set to NA to ignore existing weights.
invert_weights	Logical or NULL. Invert weights so higher weights = shorter paths? Default NULL which auto-detects: TRUE for tna objects, FALSE otherwise (matching igraph/sna). Set TRUE for strength/frequency weights (qgraph style).
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 average local efficiency. For unweighted simple graphs this is in [0, 1]; weighted graphs can exceed 1 when edge distances are below 1.

Examples

```
# Complete graph: removing any node leaves complete subgraph, so local efficiency = 1
k5 <- matrix(1, 5, 5); diag(k5) <- 0
network_local_efficiency(k5) # 1

# Star: neighbors not connected to each other
star <- matrix(c(0,1,1,1,1, 1,0,0,0,0, 1,0,0,0,0, 1,0,0,0,0, 1,0,0,0,0), 5, 5)
network_local_efficiency(star) # 0

# Per-node values under the Latora definition
centrality(star, measures = "local_efficiency")
```

network_radius	<i>Network Radius</i>
----------------	-----------------------

Description

Computes the radius of a network - the minimum eccentricity across all nodes. The eccentricity of a node is the maximum shortest path distance to any other node. The radius is the smallest such maximum distance.

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.
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

Value

Numeric: the network radius

Examples

```
# Star graph: center has eccentricity 1, leaves have 2, so radius = 1
star <- matrix(c(0,1,1,1, 1,0,0,0, 1,0,0,0, 1,0,0,0), 4, 4)
network_radius(star) # 1
```

network_rich_club	<i>Rich Club Coefficient</i>
-------------------	------------------------------

Description

Computes the rich club coefficient for a given degree threshold *k*. Measures the tendency of high-degree nodes to connect to each other. A normalized version compares to random graphs.

Usage

```
network_rich_club(x, k = NULL, normalized = FALSE, n_random = 10, ...)
```

Arguments

<i>x</i>	Network input: matrix, igraph, network, cograph_network, or tna object
<i>k</i>	Degree threshold. Only nodes with degree > <i>k</i> are included. If NULL, uses median degree.
<i>normalized</i>	Logical. Normalize by random graph expectation? Default FALSE.
<i>n_random</i>	Number of random graphs for normalization. Default 10.
<i>...</i>	Passed to to_igraph , whose only other argument is directed; anything else raises an "unused argument" error.

Value

Numeric: rich club coefficient (> 1 indicates rich club effect when normalized). 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

```
# Scale-free networks often show rich-club effect
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_pa(50, m = 2, directed = FALSE)
  network_rich_club(g, k = 5)
}
```

network_small_world	<i>Small-World Coefficient (Sigma)</i>
---------------------	--

Description

Computes the small-world coefficient σ , defined as: $\sigma = (C / C_{\text{rand}}) / (L / L_{\text{rand}})$ where C is clustering coefficient, L is mean path length, and $_{\text{rand}}$ are values from equivalent random graphs.

Usage

```
network_small_world(x, n_random = 10, ...)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object
<code>n_random</code>	Number of Erdos-Renyi comparison graphs (same n and m as the observed graph). Default 10.
<code>...</code>	Passed to to_igraph , whose only other argument is <code>directed</code> ; anything else raises an "unused argument" error.

Details

Values > 1 indicate small-world properties. Typically small-world networks have $\sigma \gg 1$.

Value

Numeric: small-world coefficient σ . 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, and prefer a larger `n_random` than the default for anything you report.

Examples

```
# Watts-Strogatz small-world graph
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_smallworld(1, 20, 3, 0.1)
  network_small_world(g) # Should be > 1
}
```

network_summary	<i>Network-Level Summary Statistics</i>
-----------------	---

Description

Computes comprehensive network-level statistics for a network. Returns a data frame with one row containing various metrics including density, centralization scores, transitivity, and more.

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". Affects degree-based calculations. 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, include mean/sd centrality statistics. Default FALSE returns 18 basic metrics; TRUE returns 29 metrics.

<code>extended</code>	Logical. If TRUE, include additional structural metrics (girth, radius, clique size, cut vertices, bridges, efficiency). Default FALSE.
<code>digits</code>	Integer. Round numeric results to this many decimal places. Default 3.
<code>...</code>	Additional arguments (currently unused)

Value

A data frame with one row containing network-level statistics:

Basic measures (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 cut value (edge connectivity)
centralization_degree Degree centralization (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

Extended measures (when `extended = TRUE`):

girth Length of shortest cycle (Inf if acyclic)
radius Minimum eccentricity (shortest max-distance from any node)
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

Detailed measures (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

```
# Basic usage with adjacency matrix
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
network_summary(adj)

# With detailed statistics
network_summary(adj, detailed = TRUE)

# With extended structural metrics
network_summary(adj, extended = TRUE)

# All metrics
network_summary(adj, detailed = TRUE, extended = TRUE)

# From igraph object
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_gnp(20, 0.3)
  network_summary(g)
}
```

network_vertex_connectivity

Network Vertex Connectivity

Description

Computes the vertex connectivity of a network - the minimum number of vertices that must be removed to disconnect the graph (or make it trivial). Higher values indicate more robust network 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

Integer: minimum vertex cut size

Examples

```
# Complete graph K4 has vertex connectivity 3
k4 <- matrix(1, 4, 4); diag(k4) <- 0
network_vertex_connectivity(k4) # 3

# Path graph has vertex connectivity 1
path <- matrix(c(0,1,0,0, 1,0,1,0, 0,1,0,1, 0,0,1,0), 4, 4)
network_vertex_connectivity(path) # 1
```

network_wrangling	<i>Network Wrangling Verbs</i>
-------------------	--------------------------------

Description

cograph's verbs for reshaping a network. Every verb takes any supported input (matrix, edge list, igraph, statnet network, tna model, cograph_network), takes its options as named arguments, and returns a cograph_network — or the input format when keep_format = TRUE. There is no pipeline state to activate and nothing to unpack afterwards: use as.data.frame() for the tidy edge or node table.

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.

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.

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 worth knowing

- **Filtering edges does not remove nodes.** This matches `igraph::delete_edges()` and `tidygraph`. Nodes left without edges raise a `cograph_isolates_created` warning; call `remove_isolates()` to drop them, or pass `keep_isolates = FALSE`.
- **Undirected results stay undirected.** The weight matrix of an undirected result is symmetric, so nothing downstream re-detects it as directed.
- **Metadata survives.** Node groups, estimation data, layout coordinates and the original source type are carried through every verb.
- **Malformed selections are errors.** Unknown node names, out-of-range or fractional indices, unknown measure names and a malformed between raise a `cograph_bad_selection` error rather than warning and returning something plausible.

Related verbs elsewhere

`ego_networks()`, `shortest_paths()`, `disparity_filter()`, `detect_communities()`, `summarize_clusters()`, `aggregate_layers()`.

Examples

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# One call, named arguments, a tidy table out
as.data.frame(threshold_edges(adj, minimum = 0.4))
```

```
# Verbs compose
adj |>
  threshold_edges(minimum = 0.4) |>
  remove_isolates() |>
  mutate_nodes(deg = degree) |>
  as.data.frame(what = "nodes")
```

nodes

Get Nodes from Cograph Network (Deprecated)

Description

Extracts the nodes data frame from a `cograph_network` object. **Deprecated:** Use [get_nodes](#) instead.

Usage

```
nodes(x)
```

Arguments

`x` A `cograph_network` object.

Value

A node metadata data frame, usually with `id` and `label` columns, plus layout or other metadata columns when present.

See Also

[get_nodes](#), [as_cograph](#), [n_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
nodes(net) # Deprecated, use get_nodes(net) instead
```

normalize_weights	<i>Normalize Edge Weights</i>
-------------------	-------------------------------

Description

Rescales the weight matrix. Row normalization is what turns a transition count matrix into the transition probabilities that TNA models use.

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 total of all weights "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, or the whole matrix) whose total is zero is left at zero rather than producing NaN: there is nothing to distribute. Rows with a zero total are reported in a `cograph_zero_norm` warning so that the zeros are a stated result rather than a silent one.

"minmax" maps the weakest edge to `.Machine$double.eps` rather than to exactly 0, because 0 is how this representation stores "no edge": mapping to it would delete the weakest edge instead of rescaling it.

"max", "sum" and "minmax" rescale each edge independently and therefore keep any extra edge columns. "row" and "column" scale an edge by a total that differs at its two endpoints, so they break symmetry and return a directed network.

Row and column normalization are meaningful on directed networks. On an undirected network they still work but break symmetry, so the result is returned as directed.

Value

A `cograph_network` with rescaled weights, or the input format when `keep_format = TRUE`.

See Also

`binarize`, `invert_weights`, `symmetrize`

Examples

```
counts <- matrix(c(0, 3, 1,
                  2, 0, 4,
                  5, 1, 0), 3, 3, byrow = TRUE)
rownames(counts) <- colnames(counts) <- c("A", "B", "C")

normalize_weights(counts, method = "row")
normalize_weights(counts, method = "max")
```

<code>n_communities</code>	<i>Get Number of Communities</i>
----------------------------	----------------------------------

Description

Get Number of Communities

Usage

```
n_communities(x)
```

Arguments

`x` A `cograph_communities` object

Value

Integer count of communities

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
n_communities(comm)
```

n_edges	<i>Get Number of Edges</i>
---------	----------------------------

Description

Returns the number of edges in a `cograph_network`.

Usage

```
n_edges(x)
```

Arguments

`x` A `cograph_network` object.

Value

Integer: number of edges.

See Also

[as_cograph](#), [n_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
n_edges(net) # 3
```

n_nodes	<i>Get Number of Nodes</i>
---------	----------------------------

Description

Returns the number of nodes in a `cograph_network`.

Usage

```
n_nodes(x)
```

Arguments

`x` A `cograph_network` object.

Value

Integer: number of nodes.

See Also

[as_cograph](#), [n_edges](#), [get_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
n_nodes(net) # 3
```

overlay_communities	<i>Overlay Community Blobs on a Network Plot</i>
---------------------	--

Description

Render a network with [splot](#) and overlay smooth blob shapes highlighting node communities.

Usage

```
overlay_communities(
  x,
  communities,
  blob_colors = NULL,
  blob_alpha = 0.25,
  blob_linewidth = 0.7,
  blob_line_alpha = 0.8,
  ...
)
```

Arguments

<code>x</code>	A network object passed to splot : <code>tna</code> , <code>matrix</code> , <code>igraph</code> , or <code>cograph_network</code> .
<code>communities</code>	Community assignments in any format: a method name (e.g., <code>"walktrap"</code> , <code>"louvain"</code>), a numeric or factor membership vector (e.g., <code>c(1, 1, 2, 2, 3)</code>), a named list of character vectors, a <code>cograph_communities</code> object, or a <code>tna_communities</code> object.
<code>blob_colors</code>	Character vector of fill colors for blobs. Recycled if shorter than the number of communities. Default <code>NULL</code> uses the built-in blob palette.
<code>blob_alpha</code>	Numeric. Fill transparency (0-1). Default <code>0.25</code> .
<code>blob_linewidth</code>	Numeric. Border line width. Default <code>0.7</code> .
<code>blob_line_alpha</code>	Numeric. Border line transparency (0-1). Default <code>0.8</code> .
<code>...</code>	Additional arguments passed to splot .

Value

The [splot](#) result — a `cograph_network` object — invisibly. Called for the side effect of drawing.

Examples

```
set.seed(1)
mat <- matrix(runif(25), 5, 5,
              dimnames = list(LETTERS[1:5], LETTERS[1:5]))
diag(mat) <- 0
overlay_communities(mat, list(g1 = c("A","B"), g2 = c("C","D","E")))

comm <- cograph::communities(regulation_net, method = "infomap")
overlay_communities(regulation_net, comm)
```

palettes	<i>Color Palettes</i>
----------	-----------------------

Description

Built-in color palettes for network visualization.

Examples

```
palette_blues(5)
palette_reds(5)
```

palette_blues	<i>Blues Palette</i>
---------------	----------------------

Description

Generate a blue sequential palette.

Usage

```
palette_blues(n, alpha = 1)
```

Arguments

- n Number of colors to generate.
- alpha Transparency (0-1).

Value

Character vector of colors.

Examples

```
palette_blues(5)
```

palette_colorblind	<i>Colorblind-friendly Palette</i>
--------------------	------------------------------------

Description

Generate a colorblind-friendly palette using Wong's colors.

Usage

```
palette_colorblind(n, alpha = 1)
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).

Value

Character vector of colors.

Examples

```
palette_colorblind(5)
```

palette_diverging	<i>Diverging Palette</i>
-------------------	--------------------------

Description

Generate a diverging color palette (blue-white-red).

Usage

```
palette_diverging(n, alpha = 1, midpoint = "white")
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).
midpoint	Color for midpoint.

Value

Character vector of colors.

Examples

```
palette_diverging(5)
```

palette_pastel	<i>Pastel Palette</i>
----------------	-----------------------

Description

Generate a soft pastel color palette.

Usage

```
palette_pastel(n, alpha = 1)
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).

Value

Character vector of colors.

Examples

```
palette_pastel(5)
```

palette_rainbow	<i>Rainbow Palette</i>
-----------------	------------------------

Description

Generate a rainbow color palette.

Usage

```
palette_rainbow(n, alpha = 1)
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).

Value

Character vector of colors.

Examples

```
palette_rainbow(5)
```

palette_reds	<i>Reds Palette</i>
--------------	---------------------

Description

Generate a red sequential palette.

Usage

```
palette_reds(n, alpha = 1)
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).

Value

Character vector of colors.

Examples

```
palette_reds(5)
```

palette_viridis	<i>Viridis Palette</i>
-----------------	------------------------

Description

Generate colors from the viridis palette.

Usage

```
palette_viridis(n, alpha = 1, option = "viridis")
```

Arguments

n	Number of colors to generate.
alpha	Transparency (0-1).
option	Viridis option: "viridis", "magma", "plasma", "inferno", "cividis".

Value

Character vector of colors.

Examples

```
palette_viridis(5)
```

panel_layout	<i>Configure a custom multi-panel layout</i>
--------------	--

Description

Sets up a multi-panel device layout for use with cograph plotting functions called with `combined = FALSE`. Returns a `par()` snapshot of the previous device state so the caller can restore it via `on.exit(graphics::par(old_par))`.

Usage

```
panel_layout(spec, mar = c(2, 2, 3, 1), widths = NULL, heights = NULL)
```

Arguments

<code>spec</code>	Either a length-2 integer vector <code>c(nrow, ncol)</code> for a uniform grid, or a numeric matrix of panel positions to pass to <code>graphics::layout()</code> .
<code>mar</code>	Numeric vector of length 4 giving panel margins. Default <code>c(2, 2, 3, 1)</code> matches cograph's multi-panel margin convention.
<code>widths, heights</code>	Optional numeric vectors of column widths and row heights. Only valid when <code>spec</code> is a matrix; passed straight to <code>graphics::layout()</code> . Supplying them with a uniform-grid <code>spec</code> is an error, since <code>par(mfrow=...)</code> has no widths/heights concept.

Details

Use `spec = c(nrow, ncol)` for a uniform grid (delegates to `graphics::par(mfrow = ...)`). Use `spec = <matrix>` for a non-uniform layout (delegates to `graphics::layout()`); the matrix values name panel cells, so `matrix(c(1, 1, 2, 3), 2, 2)` produces one wide cell on top and two cells on the bottom row.

Value

Invisibly returns a list of previous `par()` settings that can be passed back to `graphics::par()` to restore the prior device state. For both `spec` shapes the snapshot includes `mfrow`, so `par(old_par)` also resets any `graphics::layout()` partitioning that this call introduced.

Combined-flag scope

`panel_layout()` composes with the `combined = FALSE` opt-out on cograph's multi-panel plot functions. Single-network calls like `splot(some_tna_object)` do not honor `combined` — there is nothing for it to gate. Pass `combined = FALSE` only to the multi-panel hosts: `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` / similar list-of-plottables inputs.

Examples

```
mat <- matrix(c(0, .5, .3, .5, 0, .4, .3, .4, 0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A", "B", "C")
net1 <- as_cograph(mat)
net2 <- as_cograph(mat * 0.5)

# Uniform 1 x 2 grid
op <- panel_layout(c(1, 2))
splot(net1, combined = FALSE)
splot(net2, combined = FALSE)
graphics::par(op)
```

```
plot.cograph_cluster_significance
      Plot Cluster Significance
```

Description

Creates a histogram of the null distribution with the observed value marked.

Usage

```
## S3 method for class 'cograph_cluster_significance'
plot(x, ...)
```

Arguments

```
x          A cograph_cluster_significance object
...        Additional arguments passed to hist
```

Value

Invisibly returns x

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
sig <- cluster_significance(g, comm, n_random = 20, seed = 42)
plot(sig)
```

`plot.cograph_communities`*Plot Community Structure*

Description

Visualizes network with community coloring using `splot`.

Usage

```
## S3 method for class 'cograph_communities'  
plot(x, network = NULL, ...)
```

Arguments

<code>x</code>	A <code>cograph_communities</code> object
<code>network</code>	The original network (required if not stored)
<code>...</code>	Additional arguments passed to <code>splot</code>

Value

The value returned by `splot` (invisibly). Called for the side effect of drawing the network with nodes grouped by community.

Examples

```
g <- igraph::make_graph("Zachary")  
comm <- community_louvain(g)  
mat <- igraph::as_adjacency_matrix(g, sparse = FALSE)  
plot(comm, network = mat)
```

`plot.cograph_core_periphery`*Plot Core-Periphery Structure*

Description

Visualizes the network with core nodes highlighted (larger, red) and periphery nodes de-emphasized (smaller, blue).

Usage

```
## S3 method for class 'cograph_core_periphery'
plot(
  x,
  core_color = "#E41A1C",
  periphery_color = "#377EB8",
  core_size = 12,
  periphery_size = 6,
  ...
)
```

Arguments

x	A cograph_core_periphery object from core_periphery .
core_color	Color for core nodes. Default "#E41A1C".
periphery_color	Color for periphery nodes. Default "#377EB8".
core_size	Numeric size for core nodes. Default 12.
periphery_size	Numeric size for periphery nodes. Default 6.
...	Additional arguments passed to splot .

Value

Invisible x.

Examples

```
adj <- matrix(c(0,1,1,1,0, 1,0,1,1,0, 1,1,0,1,1,
               1,1,1,0,1, 0,0,1,1,0), 5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
cp <- cograph::core_periphery(adj)
plot(cp)
```

plot.cograph_degree_fit

Plot method for cograph_degree_fit

Description

Overlays fitted distribution curves on a histogram of observed degrees.

Usage

```
## S3 method for class 'cograph_degree_fit'
plot(
  x,
  which = NULL,
  log = "",
  cols = NULL,
  lwd = 2,
  main = "Degree Distribution Fit",
  ...
)
```

Arguments

<code>x</code>	A <code>cograph_degree_fit</code> object from fit_degree_distribution .
<code>which</code>	Character vector of distribution names to display. Default <code>NULL</code> shows all fitted distributions.
<code>log</code>	Character string for log-scale axes: one of <code>""</code> (default), <code>"x"</code> , <code>"y"</code> or <code>"xy"</code> . Only <code>"y"</code> and <code>"xy"</code> actually log the histogram axis; the values containing <code>"x"</code> are accepted for compatibility but merely filter non-positive fitted curve values.
<code>cols</code>	Named or unnamed character vector of colors for distribution curves. Default uses a built-in palette.
<code>lwd</code>	Line width for fitted curves. Default 2.
<code>main</code>	Plot title. Default <code>"Degree Distribution Fit"</code> .
<code>...</code>	Additional arguments passed to hist .

Value

Invisible `NULL`.

Examples

```
adj <- matrix(c(0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
               1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
               0, 0, 1, 1, 0), 5, 5, byrow = TRUE)
fit <- cograph::fit_degree_distribution(adj)
plot(fit)
```

plot.cograph_motifs *Plot Network Motifs*

Description

Visualize motif frequencies and their statistical significance.

Usage

```
## 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,
  ...
)
```

Arguments

<code>x</code>	A <code>cograph_motifs</code> object from <code>motif_census()</code>
<code>type</code>	Plot type: "bar" (default) Bar chart of motif frequencies, colored by significance direction (over/under-represented). "heatmap" Heatmap of z-scores across motif types. "network" Network diagrams of the top motifs by lz-scorel.
<code>show_nonsig</code>	Show non-significant motifs? Default FALSE.
<code>top_n</code>	Show only top N motifs by lz-scorel. Default NULL (all).
<code>colors</code>	Three-element color vector for under-represented, neutral, and over-represented motifs. Default <code>c("#2166AC", "#F7F7F7", "#B2182B")</code> (blue/near-white/red).
<code>combined</code>	Logical: when TRUE (default) and <code>type = "network"</code> , arrange the per-motif panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw into a layout the caller has already configured (e.g. via <code>panel_layout()</code>). Has no effect for <code>type = "bar"</code> or <code>type = "heatmap"</code> .
<code>...</code>	For <code>type = "network"</code> , additional arguments passed to the per-motif igraph plot calls. The ggplot-based types ("bar", "heatmap") do not consume them.

Value

For `type = "bar"` and `type = "heatmap"`, a ggplot2 object. For `type = "network"`, NULL (the panels are drawn with base graphics for their side effect). `invisible(NULL)` with a message when no motif survives the `show_nonsig` / `top_n` filters.

See Also

`motif_census()` for the analysis that produces this object

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `motifs()`, `plot.cograph_motif_analysis()`, `subgraphs()`, `triad_census()`

Examples

```
mat <- matrix(sample(0:1, 100, replace = TRUE, prob = c(0.7, 0.3)), 10, 10)
diag(mat) <- 0
m <- motif_census(mat, directed = TRUE, n_random = 50)
plot(m)
plot(m, type = "network")
```

```
plot.cograph_motif_analysis
Plot Motif Analysis Results
```

Description

Create visualizations for motif analysis results including network diagrams of triads, bar plots of type distributions, and significance plots.

Usage

```
## 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,
  ...
)
```

Arguments

x	A cograph_motif_analysis object from extract_motifs()
type	Plot type: "triads" (default) Network diagrams of specific named triads, arranged in a grid. Each cell shows the three nodes and their edges. "types" Bar chart of MAN type frequencies.

	"significance" Z-score plot with one bar per node-triple and MAN type. Requires significance = TRUE in extract_motifs().
	"patterns" Abstract MAN pattern diagrams showing edge structure of each triad type without specific node labels.
n	Number of triads/patterns to show. Default 20.
colors	Two-element color vector mapped to a three-tone significance scale (used by type = "significance" and by type = "patterns" node fills): colors[1] fills items that are significantly under-represented ($p < .05$ and $z < 0$); colors[2] fills items that are significantly over-represented ($p < .05$ and $z > 0$); everything else is filled neutral grey ("#9E9E9E"). When significance was not run, patterns nodes use colors[1] as a single fill. Default c("#2166AC", "#B2182B") (blue for under, red for over).
res	Resolution for scaling (kept for backwards compatibility). Default 72.
node_size	Size of nodes in triad diagrams (1-10 scale). Default 5.
label_size	Font size for node labels (3-letter abbreviations). Default 7.
title_size	Font size for motif type title (e.g., "120C"). Default 7.
stats_size	Font size for statistics text (n, z, p). Default 5.
ncol	Number of columns in the plot grid. Default 5.
legend	Show abbreviation legend at bottom? Default TRUE.
color	Color for nodes, edges, and labels in triad diagrams. Default "#800020" (maroon).
spacing	Spacing multiplier between grid cells (0.5-2). Default 1.
combined	Logical: when TRUE (default) and type = "patterns", arrange the per-motif panels in an internal grid via graphics::par(mfrow=...). Set to FALSE to draw into a layout the caller has already configured (e.g. via panel_layout()).
...	Additional arguments (unused).

Value

Invisibly returns NULL for triad and pattern plots, or a ggplot2 object for types and significance plots.

See Also

[extract_motifs\(\)](#) for the analysis that produces this object, [motif_census\(\)](#) for statistical motif analysis

Other motifs: [extract_motifs\(\)](#), [extract_triads\(\)](#), [get_edge_list\(\)](#), [motif_census\(\)](#), [motifs\(\)](#), [plot.cograph_motifs\(\)](#), [subgraphs\(\)](#), [triad_census\(\)](#)

Examples

```
mat <- matrix(c(0,3,2,0, 0,0,5,1, 0,0,0,4, 2,0,0,0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("Plan", "Execute", "Monitor", "Adapt")
m <- extract_motifs(mat, significance = FALSE)
plot(m)
plot(m, type = "types")
```

`plot.cograph_network` *Plot cograph_network Object*

Description

Plot cograph_network Object

Usage

```
## S3 method for class 'cograph_network'  
plot(x, ...)
```

Arguments

<code>x</code>	A cograph_network object.
<code>...</code>	Additional arguments passed to <code>sn_render</code> .

Value

The input object `x`, invisibly.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)  
net <- cograph(adj)  
plot(net)
```

`plot.cograph_rich_club`*Plot Rich Club Results*

Description

Two plot types: "curve" (default) shows the rich club coefficient across thresholds with null model bands. "network" highlights rich club members on the network at a given threshold.

Usage

```
## S3 method for class 'cograph_rich_club'  
plot(x, type = c("curve", "network"), k = NULL, col = "#E41A1C", ...)
```

Arguments

x	A cograph_rich_club data frame.
type	Character. "curve" (default) or "network".
k	Numeric. For type = "network", the degree/strength threshold to visualize. If NULL, uses the threshold with the highest phi_norm (or phi if not normalized).
col	Line/node color for rich club. Default "#E41A1C".
...	Additional arguments passed to plot (curve) or splot (network).

Value

Invisible x.

Examples

```
g <- igraph::sample_pa(50, m = 2, directed = FALSE)
rc <- cograph::rich_club(g)
plot(rc)
```

plot.cograph_vulnerability

Plot Node Vulnerability

Description

Plot Node Vulnerability

Usage

```
## S3 method for class 'cograph_vulnerability'
plot(x, top = NULL, col = "steelblue", ...)
```

Arguments

x	A cograph_vulnerability object.
top	Integer or NULL. Show only top N nodes. Default NULL (all).
col	Bar color. Default "steelblue".
...	Additional arguments passed to barplot .

Value

Invisible x.

Examples

```
star <- matrix(c(0,1,1,1, 1,0,0,0, 1,0,0,0, 1,0,0,0), 4, 4)
rownames(star) <- colnames(star) <- c("hub", "a", "b", "c")
v <- cograph::vulnerability(star)
plot(v)
```

plot.tna_bootstrap	<i>Plot Bootstrap Results</i>
--------------------	-------------------------------

Description

Visualizes bootstrap analysis results with styling to distinguish significant from non-significant edges. Works with tna_bootstrap objects from the tna package.

Usage

```
## S3 method for class 'tna_bootstrap'
plot(x, ...)

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,
  ...
)
```

Arguments

x	A tna_bootstrap object (from tna::bootstrap).
...	Additional arguments passed to splot().
display	Display mode: • "styled" (default): All edges with styling to distinguish significant/non-significant • "significant": Only significant edges • "full": All edges without significance styling • "ci": Show CI bands on edges
edge_style_sig	Line style for significant edges (1=solid). Default 1.

edge_style_nonsig	Line style for non-significant edges (2=dashed). Default 2.
color_nonsig	Accepted for compatibility; styled mode currently uses a fixed pink color for non-significant edges.
show_ci	Logical: include CI bounds in edge labels? Default FALSE. Use display = "ci" for CI underlays on edges.
show_stars	Logical: show significance stars (*, **, ***) on edges? Default TRUE.
width_by	Optional: "cr_lower" to scale edge width by lower consistency range bound.
inherit_style	Logical: inherit colors/layout from original TNA model? Default TRUE.

Details

The function expects a `tna_bootstrap` object containing:

- `weights` or `weights_orig`: Original weight matrix
- `weights_sig`: Significant weights only (optional)
- `p_values`: P-value matrix
- `ci_lower`, `ci_upper`: Confidence interval bounds
- `level`: Significance level (default 0.05)
- `model`: Original TNA model for styling inheritance

Edge styling in "styled" mode:

- Significant edges: solid dark blue, bold labels with stars, rendered on top
- Non-significant edges: dashed pink, plain labels, rendered behind

Value

Invisibly returns the `cograph_network` object built by `splot()`. Called for the side effect of drawing.

Examples

```
# Mock a tna_bootstrap object with synthetic data
w <- matrix(c(0, .3, .1, .2, 0, .4, .3, .1, 0), 3, 3)
rownames(w) <- colnames(w) <- c("A", "B", "C")
p <- matrix(c(1, .01, .5, .03, 1, .001, .2, .8, 1), 3, 3)
boot <- list(weights = w, p_values = p,
             ci_lower = w - 0.05, ci_upper = w + 0.05, level = 0.05,
             model = list(weights = w, labels = c("A", "B", "C")))
class(boot) <- c("tna_bootstrap", "list")
splot(boot)
splot(boot, display = "significant")
```

plot.tna_disparity	<i>Plot Disparity Filter Result</i>
--------------------	-------------------------------------

Description

Plot Disparity Filter Result

Usage

```
## S3 method for class 'tna_disparity'
plot(x, type = c("backbone", "comparison"), combined = TRUE, ...)
```

Arguments

x	A tna_disparity object.
type	Plot type: "backbone" (default) or "comparison".
combined	Logical: when type = "comparison", controls whether the original vs. backbone panels are arranged in an internal 1 x 2 grid (TRUE, default) or drawn into a layout the caller has already configured (FALSE — pair with panel_layout()). Ignored for type = "backbone".
...	Additional arguments passed to splot .

Value

Invisibly returns the value from the underlying [splot](#) call. Called primarily for the side effect of producing a plot.

Examples

```
mat <- matrix(c(0.0, 0.5, 0.1, 0.0, 0.3, 0.0, 0.4, 0.1,
               0.1, 0.2, 0.0, 0.5, 0.0, 0.1, 0.3, 0.0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("A", "B", "C", "D")
disp <- disparity_filter(cograph(mat), level = 0.05)
plot(disp)
```

plot_alluvial	<i>Plot Alluvial Diagram</i>
---------------	------------------------------

Description

Creates an alluvial (Sankey) diagram showing aggregated flows between states. This is an alias for [plot_transitions\(\)](#) with aggregated flows (default).

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 several formats: • A transition matrix (rows = from, cols = to, values = counts) • Two vectors: pass before as x and after as second argument (contingency table computed automatically, like chi-square) • A 2-column data frame (raw observations; table computed automatically) • A data frame with columns: from, to, count • A list of matrices for multi-step transitions
from_title	Title for the left column. Default "From". For multi-step, use a vector of titles (e.g., c("T1", "T2", "T3", "T4")).
to_title	Title for the right column. Default "To". Ignored for multi-step.
title	Optional plot title. Applied via ggplot2::labs(title = title).
from_colors	Colors for left-side nodes. Default uses palette.
to_colors	Colors for right-side nodes. Default uses palette.
flow_fill	Fill color for flows. Default "#888888" (grey). In multi-step and individual-tracking plots, ignored when flow_color_by is set; simple two-column aggregate plots use flow_fill.
flow_alpha	Alpha transparency for flows. Default 0.4.
flow_color_by	Color flows by state. For multi-step aggregate flows, use "source" or "destination"; for individual trajectories, "first" and "last" are also supported. Default NULL uses flow_fill; simple 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", or "outside".
label_halo	Logical: add white halo around labels for readability? Default TRUE.
label_color	Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots; simple 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; simple two-column aggregate plots use fixed label font faces.
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.

<code>title_color</code>	Color of column title text. Default "black". Applied to multi-step and individual-tracking plots; simple two-column aggregate plots use black titles.
<code>title_fontface</code>	Font face of column titles. Default "bold". Applied to multi-step and individual-tracking plots.
<code>curve_strength</code>	Controls bezier curve shape (0-1). Default 0.6.
<code>show_values</code>	Logical: show transition counts on flows? Default FALSE.
<code>value_position</code>	Position of flow values: "center", "origin", "destination", "outside_origin", "outside_destination". Default "center".
<code>value_size</code>	Size of value labels on flows. Default 3.
<code>value_color</code>	Color of value labels. Default "black".
<code>value_halo</code>	Logical: add halo around flow value labels? Default NULL (inherits from <code>label_halo</code>). Applied to multi-step and individual-tracking plots.
<code>value_fontface</code>	Font face of flow value labels. Default "bold". Applied to multi-step and individual-tracking plots.
<code>value_nudge</code>	Distance of value labels from node edge when using "origin" or "destination" positions. Default 0.03.
<code>value_min</code>	Minimum count to show a flow value label in multi-step and individual-tracking plots. Default 0 (show all). Simple two-column aggregate plots show all nonzero value labels when <code>show_values = TRUE</code> .
<code>show_totals</code>	Logical: show total counts on nodes? Default FALSE.
<code>total_size</code>	Size of total labels. Default 4.
<code>total_color</code>	Color of total labels. Default "white".
<code>total_fontface</code>	Font face of total labels. Default "bold".
<code>conserve_flow</code>	Logical: should left and right totals match? Default TRUE. When FALSE, each side scales independently (allows for "lost" or "gained" items).
<code>min_flow</code>	Minimum flow value to display. Default 0 (show all).
<code>threshold</code>	Minimum edge weight to display. Flows below this value are removed. Combined with <code>min_flow</code> : effective minimum is <code>max(threshold, min_flow)</code> . Default 0.
<code>value_digits</code>	Number of decimal places for flow value labels and node totals. Default 2.
<code>column_gap</code>	Horizontal spread of columns (0-1) for multi-step and individual-tracking plots. Default 1 uses full width. Use smaller values (e.g., 0.6) to bring columns closer together.

Value

A `ggplot2` object.

See Also

[plot_transitions](#), [plot_trajectories](#)

Examples

```
mat <- matrix(c(50, 10, 5, 15, 40, 10), 2, 3)
rownames(mat) <- c("A", "B")
colnames(mat) <- c("X", "Y", "Z")
plot_alluvial(mat)
```

plot_bootstrap_forest *Forest Plot for Bootstrap Network Results*

Description

A ggplot2-based forest plot for `net_bootstrap`, `net_bootstrap_group`, `tna_bootstrap`, and `boot_glasso` objects. Each row is one network edge; horizontal bars span the confidence interval and a filled square marks the point estimate. A dashed reference line runs through zero.

Produces a ggplot2 forest plot where each row is one network edge, the square marks the bootstrap mean estimate, and the horizontal bar spans the selected interval. A dashed reference line runs through zero. Significant edges are highlighted in color; non-significant ones appear in grey (only shown when `show_nonsig = TRUE`).

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) draws the classic tall forest plot; "circular" arranges each edge as a spoke around a circle, with the inner ring at the data minimum and the outer ring at the data maximum; "grouped" arranges edges in sectors by source node where supported.
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 (greyed out)? Default TRUE.
sort_by	How to order edges on the y-axis (linear layout) or clockwise from top (radial layout): "estimate" (default, ascending), "significance" (most significant at top), or "name" (alphabetical).
n_top	Integer: restrict to the n_top edges with the largest absolute estimate. Applied after significance filtering. Default NULL.
node_colors	Optional node-color vector for grouped radial layouts.
sig_color	Color for significant CI bars and points. 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. Default "#CCCCCC".
ring_color	Color for the reference rings (radial layout only). Default "#C8C8C8".
median_color	Color for the dashed median ring (radial layout only). Default "#AAAAAA".
label_size	Text size for edge labels (radial and grouped layouts). Default NULL for automatic sizing in the main methods, or 2.8 for net_bootstrap_group.
label_color	Fixed color for edge labels (radial layout only). NULL (default) inherits the edge color (teal for significant, grey for non-significant).
point_size	Size of the estimate square. Default 3 (linear) or 2 (radial).
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

For net_bootstrap objects from stability inference, both a bootstrap confidence interval (ci_lower/ci_upper) and a consistency range (cr_lower/cr_upper) are available. Use interval = "both" to overlay both on the same plot.

Value

A ggplot object.

Examples

```
# Bootstrap a TNA built from sequence data (required by tna::bootstrap)
Mod <- tna::tna(head(tna::group_regulation, 100))
boot <- tna::bootstrap(Mod, iter = 50)
plot_bootstrap_forest(boot, n_top = 8)
```

plot_centrality	<i>Plot Centrality</i>
-----------------	------------------------

Description

Publication-quality visualization of one or more centrality measures. Accepts the data frame from [centrality](#) directly 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. Default pulls the classical five (degree, strength, betweenness, closeness, eigenvector) when x is a network; default NULL keeps all columns when x is already a centrality data frame.
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; in the "line" style this forces free y-axis per measure via faceting), "normalized" ([0, 1] within measure), "z" (standardized within measure), or "rank" (1..n, highest value = 1).
order_by	Character. For "bar"/"lollipop": which measure sorts nodes. Defaults to the first measure. Use "alpha" for alphabetical. For "line", this also controls node ordering unless "alpha" is requested.
top_n	Optional integer to keep only the top-N nodes (by order_by). Useful for large graphs.
highlight	Optional integer: highlight the top-N bars/lines per measure in full color; mute the rest. Default 0 (no highlighting).
cluster	Optional named vector or data-frame column mapping each node to a cluster/community. Colors nodes by cluster when supplied.
palette	Character or vector. "cograph" (default) uses cograph's teal-gold-leaf palette; "okabe" uses Okabe-Ito; "viridis" uses viridis; or supply a character vector of colors.
ncol	For faceted styles ("bar", "lollipop"): number of columns. Default NULL chooses sensibly based on measure count.
title	Plot title. Default NULL.
subtitle	Plot subtitle. Default NULL.
...	Passed to centrality when x is a network.

Details

Four styles are available:

"line" Faceted line view with one panel per measure. Nodes are ordered along the requested orientation and connected within each measure.

"bar" Horizontal bars, one facet per measure. Best for reading individual measure values.

"lollipop" Like "bar" but with a dot at the tip. Softer visual weight; useful on dense grids.

"dot" Dot-only variant of the lollipop style.

Value

A ggplot object.

Examples

```
adj <- matrix(c(0,1,1,0,0, 1,0,1,1,0, 1,1,0,1,1, 0,1,1,0,1, 0,0,1,1,0),
              5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
plot_centrality(adj)
plot_centrality(adj, style = "bar", highlight = 2)
```

plot_centrality_compare

Plot Centrality Comparison

Description

Compare a centrality measure across two or more groups using stacked, faceted, grouped, dumbbell, line, or two-group pyramid layouts. The "pyramid" style is a back-to-back horizontal bar chart for exactly two groups.

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. Names are used as group labels when group_labels is NULL.
measure	Character, a single centrality measure to compare. 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	Optional. 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"). Used by style = "facet".
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 on each side) or "normalized" ([0, 1] within each side before plotting).
show_values	Logical. Print the value inside each bar. 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).
ncol	Number of facet columns for style = "facet". Default NULL chooses automatically.
title	Plot title.
subtitle	Plot subtitle. Auto-generated when NULL.
centrality_args	Named list of additional arguments passed to centrality when inputs are networks.

Value

A ggplot object.

Examples

```
set.seed(1)
m1 <- matrix(runif(25), 5, 5); diag(m1) <- 0
m2 <- matrix(runif(25), 5, 5); diag(m2) <- 0
rownames(m1) <- colnames(m1) <- LETTERS[1:5]
rownames(m2) <- colnames(m2) <- LETTERS[1:5]
plot_centrality_compare(m1, m2, measure = "strength",
  group_labels = c("Pre", "Post"))
```

plot_centrality_distribution

Plot Centrality Distribution

Description

Histogram or density plot of any centrality measure. Accepts the output of [centrality](#) directly.

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. Which centrality measure to plot. Default "degree_all". Must match a column name in the centrality output.
type	Character. "histogram" (default) or "density".
normalize	Logical. Show proportions instead of counts. Default FALSE.
bins	Integer or NULL. Number of bins. Default NULL (auto).
log	Character. Log scaling: "", "y", or "xy". Values containing "x" are accepted for compatibility but only the y-axis is log-scaled by this plotting implementation. Default "".
col	Fill color. Default "steelblue".
border	Border color. Default "white".
main	Plot title. Default auto-generated from measure name.
xlab	X-axis label. Default auto-generated.
...	Additional arguments passed to barplot or plot .

Value

Invisibly returns the centrality values plotted.

Examples

```
adj <- matrix(c(0,1,1,0, 1,0,1,1, 1,1,0,1, 0,1,1,0), 4, 4)
rownames(adj) <- colnames(adj) <- LETTERS[1:4]
cograph::plot_centrality_distribution(adj, measure = "degree_all")
```

plot_centrality_heatmap

Plot Centrality Heatmap

Description

Heatmap of nodes (rows) by centrality measures (columns), z-standardized within measure so the diverging palette is meaningful. Optional row clustering groups nodes with similar centrality profiles.

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.
cluster_rows	Logical. Hierarchically cluster rows so nodes with similar profiles are adjacent. Default TRUE.
order_by	If cluster_rows = FALSE, optionally the name of a measure to sort rows by (descending). Default: first measure.
show_values	Logical. Print z-scores in 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 -> 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.
...	Passed to centrality when x is a network.

Value

A ggplot object.

Examples

```
adj <- matrix(c(0,1,1,0,0, 1,0,1,1,0, 1,1,0,1,1, 0,1,1,0,1, 0,0,1,1,0),
              5, 5)
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
plot_centrality_heatmap(adj)
```

plot_chord	<i>Chord Diagram</i>
------------	----------------------

Description

Draw a chord diagram where nodes are arcs on the outer ring and edges are curved ribbons (chords) connecting them. Arc size is proportional to total flow through each node and chord width is proportional to 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

<code>x</code>	A weight matrix, <code>cograph_network</code> , <code>CographNetwork</code> , <code>tna</code> , <code>igraph</code> , or list-like object with a matrix weights component.
<code>directed</code>	Logical. If <code>NULL</code> (default), auto-detected from matrix symmetry.
<code>segment_colors</code>	Colors for the outer ring segments. <code>NULL</code> uses a built-in vibrant palette.
<code>segment_border_color</code>	Border color for segments.
<code>segment_border_width</code>	Border width for segments.
<code>segment_pad</code>	Gap between segments in radians.
<code>segment_width</code>	Radial thickness of the outer ring as a fraction of the radius.
<code>chord_color_by</code>	How to color chords: "source" (default), "target", or a color vector of length matching the number of non-zero edges.
<code>chord_alpha</code>	Alpha transparency for chords.
<code>chord_border</code>	Border color for chords. <code>NA</code> for no border.
<code>self_loop</code>	Logical. Currently accepted for API compatibility; the current matrix preparation preserves self-loop chords.
<code>labels</code>	Node labels. <code>NULL</code> uses row names, <code>FALSE</code> suppresses labels.
<code>label_size</code>	Text size multiplier for labels.
<code>label_color</code>	Color for labels.
<code>label_offset</code>	Radial offset of labels beyond the outer ring.
<code>label_threshold</code>	Hide labels for nodes whose flow fraction is below this value.
<code>threshold</code>	Minimum absolute weight to show a chord.
<code>ticks</code>	Logical. Draw tick marks along the outer ring to indicate magnitude?
<code>tick_interval</code>	Spacing between ticks in the same units as the weight matrix. <code>NULL</code> (default) auto-selects a nice interval.
<code>tick_labels</code>	Logical. Show numeric labels at major ticks?
<code>tick_size</code>	Text size multiplier for tick labels.
<code>tick_color</code>	Color for tick marks and labels.
<code>start_angle</code>	Starting angle in radians (default $\pi/2$, top).
<code>clockwise</code>	Logical. Lay out segments clockwise?
<code>title</code>	Optional plot title.
<code>title_size</code>	Text size multiplier for the title.
<code>background</code>	Background color for the plot. <code>NULL</code> (default) uses the current device background.
<code>...</code>	Additional arguments (currently ignored).

Details

The diagram is drawn entirely with base R graphics using `polygon()` for segments and chords, and `bezier_points()` for the curved ribbons.

For directed networks, each segment is split into an outgoing half and an incoming half so that chords attach to the correct side. For undirected networks each edge is drawn once and the full segment arc is shared.

Value

Invisibly returns a list with components `segments` (data frame of segment angles and flows) and `chords` (data frame of chord endpoints and weights).

Examples

```
# Weighted directed matrix
mat <- matrix(c(
  0, 25, 5, 15,
  10, 0, 20, 8,
  3, 18, 0, 30,
  20, 5, 10, 0
), 4, 4, byrow = TRUE,
dimnames = list(c("A", "B", "C", "D"), c("A", "B", "C", "D")))

plot_chord(mat)
plot_chord(mat, chord_alpha = 0.6, ticks = TRUE)

# A transition network
plot_chord(regulation_net, ticks = TRUE, segment_width = 0.10)
```

plot_compare

Plot Network Difference (alias of plot_difference)

Description

`plot_compare()` is an alias of `plot_difference()`. It is **not deprecated**: `tna::plot_compare()` delegates to it by name (`cograph::plot_compare(x, y, ...)`), so the alias is part of the `tna` integration and must keep working. New `cograph` code may prefer the `plot_difference()` name; both call the same implementation.

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

```
m1 <- matrix(stats::runif(25), 5, 5)
m2 <- matrix(stats::runif(25), 5, 5)
rownames(m1) <- colnames(m1) <- LETTERS[1:5]
rownames(m2) <- colnames(m2) <- LETTERS[1:5]
plot_compare(m1, m2)
```

`plot_comparison_heatmap`*Plot Comparison Heatmap*

Description

Creates a heatmap visualization comparing two networks.

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, CographNetwork, tna, igraph, or list-like object with \$weights.
y	Second network: same type as x. NULL to plot just x.
type	What to display: "difference" (x - y), "x", or "y".
name_x	Label for first network in title. Default "x".
name_y	Label for second network in title. Default "y".
low_color	Color for low/negative values. Default "blue".
mid_color	Color for zero/middle values. Default "white".
high_color	Color for high/positive values. Default "red".
limits	Color scale limits. NULL for auto. Use c(-1, 1) for normalized.
show_values	Logical: display values in cells? Default FALSE.
value_size	Text size for cell values. Default 3.
digits	Decimal places for cell values. Default 2.
title	Plot title. NULL for auto-generated.
xlab	X-axis label. Default "Target".
ylab	Y-axis label. Default "Source".

Value

A ggplot2 object.

Examples

```
set.seed(42)
m1 <- matrix(runif(25), 5, 5)
m2 <- matrix(runif(25), 5, 5)
rownames(m1) <- colnames(m1) <- LETTERS[1:5]
rownames(m2) <- colnames(m2) <- LETTERS[1:5]
plot_comparison_heatmap(m1, m2)
plot_comparison_heatmap(m1, type = "x")
```

plot_degree_correlation

Plot Degree-Degree Correlation

Description

Scatter plot of each node's degree against the average degree of its neighbors. Reveals assortative (positive slope) or disassortative (negative slope) mixing patterns.

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. For directed networks: "all", "in", or "out". Default "all".
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 returns a data frame with columns node, degree, avg_neighbor_degree.

See Also

[centrality](#), [degree_distribution](#), [network_summary](#)

Examples

```
g <- igraph::sample_pa(100, m = 3, directed = FALSE)
cograph::plot_degree_correlation(g)
```

plot_difference

Plot Network Difference

Description

Plots the difference between two networks ($x - y$) using `splot`. Positive differences ($x > y$) are shown in green, negative ($x < y$) in red. Optionally displays node-level differences (e.g., initial probabilities) 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, CographNetwork, tna, igraph, list-like object with \$weights, plain list of networks, or group_tna. For group_tna with 2 groups, compares them directly. For more groups, plots all pairwise comparisons (or specify i, j).
y	Second network: same type as x. Ignored if x is a list or group_tna.
i	Index/name of first group when x is group_tna or a plain list. NULL plots all pairs for a group_tna of more than two groups, and selects the first element otherwise.
j	Index/name of second group when x is group_tna or a plain list. NULL plots all pairs for a group_tna of more than two groups, and selects the second element otherwise.
pos_color	Color for positive differences ($x > y$). Default "#009900" (green).
neg_color	Color for negative differences ($x < y$). Default "#C62828" (red).
labels	Node labels. NULL uses rownames or defaults.
title	Plot title. NULL for auto-generated title.
inits_x	Node values for x (e.g., initial probabilities). NULL to auto-extract from tna.
inits_y	Node values for y. NULL to auto-extract from tna.
show_inits	Logical: show node differences as donuts? Default NULL, which shows them when inits are available for both networks.
donut_inner_ratio	Inner radius ratio for donut (0-1). Default 0.8.
force	Logical: force plotting when more than 4 groups (many comparisons). Default FALSE.

combined	Logical: when TRUE (default) and x is a multi-group input that triggers all-pairs plotting, lay panels out in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw into a layout the caller has already configured (e.g. via <code>panel_layout()</code>). Has no effect for the single-pair path.
difference	Logical. If TRUE, x is treated as an already-subtracted difference network (no y needed). A <code>tna_comparison</code> object (from <code>tna::compare()</code>) is detected automatically and its <code>\$difference_matrix</code> is used.
...	Additional arguments passed to <code>splot()</code> .

Details

The function computes element-wise subtraction of the weight matrices. Edge colors indicate direction of difference:

- Green edges: x has higher weight than y
- Red edges: y has higher weight than x

When initial probabilities (inits) are provided or extracted from `tna` objects, nodes display donut charts showing the absolute difference, colored by direction:

- Green donut: x has higher initial probability
- Red donut: y has higher initial probability

For lists of networks (e.g., `group_tna`), specify which elements to compare using `i` and `j` parameters.

Value

Invisibly returns a list with elements `weights` (the element-wise difference matrix $x - y$) and `inits` (the node-value difference, or NULL when no inits were available). For the `group_tna` all-pairs path, a named list of such lists — one element per pair, named "`<group_i>-vs-<group_j>`".

See Also

`plot_compare`, a first-class alias of this function kept for the `tna` integration. `plot_difference()` is the preferred name.

Examples

```
set.seed(42)
m1 <- matrix(runif(25), 5, 5)
m2 <- matrix(runif(25), 5, 5)
rownames(m1) <- colnames(m1) <- LETTERS[1:5]
rownames(m2) <- colnames(m2) <- LETTERS[1:5]
plot_difference(m1, m2)

# With node-level differences
plot_difference(m1, m2,
  inits_x = c(.3, .2, .2, .15, .15),
  inits_y = c(.1, .4, .2, .2, .1))
```

plot_edge_diff_forest *Forest Plot for Bootstrap Edge Differences*

Description

Visualizes pairwise edge weight differences from a `boot_glasso` object. Each row (linear) or spoke (circular) is one edge pair; the CI bar spans the bootstrap CI of the difference; a dashed line/ring marks zero. Red = first edge larger; blue = second edge 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

<code>x</code>	A <code>boot_glasso</code> object with <code>\$boot_edges</code> and <code>\$edge_diff_p</code> .
<code>...</code>	Currently unused.
<code>alpha</code>	Significance threshold. Default <code>NULL</code> , which inherits <code>x\$alpha</code> , falling back to <code>0.05</code> .
<code>layout</code>	"linear" (default), "circular", "chord", or "tile". The chord layout places all edge names on a unit circle and connects significant pairs with bezier arcs; arc width and color encode the mean bootstrap difference. The tile layout draws the pairwise-difference matrix.

<code>show_nonsig</code>	Include non-significant pairs? Default FALSE.
<code>nonzero_only</code>	If TRUE, restrict to edges that are non-zero in the original network (identified via <code>\$original_pcor</code>). Useful for EBICglasso results where many edges are regularized to exactly zero. Default FALSE.
<code>sort_by</code>	"estimate" (default), "significance", or "name" (linear only).
<code>n_top</code>	Restrict to top N pairs by absolute difference.
<code>pos_color</code>	Color when <code>edge1 > edge2</code> . Default crimson.
<code>neg_color</code>	Color when <code>edge1 < edge2</code> . Default teal.
<code>nonsig_color</code>	Color for non-significant pairs.
<code>ring_color</code>	Ring color (circular/chord). Default light grey.
<code>label_size</code>	Text size. Default 2.3.
<code>label_color</code>	Fixed label color (NULL = inherit).
<code>point_size</code>	Size of estimate square (linear/circular). Default 2 for layout = "circular" and 3 otherwise.
<code>r_inner</code>	Inner ring radius (circular). Default 0.38.
<code>r_outer</code>	Outer ring radius (circular). Default 0.72.
<code>title</code>	Plot title.
<code>subtitle</code>	Plot subtitle.

Value

A ggplot object.

Examples

```
set.seed(1)
data1 <- as.data.frame(matrix(rnorm(60), 20, 3, dimnames = list(NULL, c("A","B","C"))))
# cs_iter only drives case-dropping stability, which this plot does not use.
bg <- Nestimate::boot_glasso(data1, iter = 50, cs_iter = 25,
                             centrality = c("strength", "expected_influence"))
plot_edge_diff_forest(bg)
```

<code>plot_edge_weights</code>	<i>Plot Edge Weight Distribution</i>
--------------------------------	--------------------------------------

Description

Histogram of edge weights in a network.

Usage

```
plot_edge_weights(
  x,
  normalize = FALSE,
  bins = NULL,
  log = "",
  directed = NULL,
  col = "steelblue",
  border = "white",
  main = "Edge Weight Distribution",
  xlab = "Weight",
  ...
)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna.
<code>normalize</code>	Logical. Show proportions. Default FALSE.
<code>bins</code>	Integer or NULL. Number of bins. Default NULL (auto).
<code>log</code>	Character. Log scaling. Default "".
<code>directed</code>	Logical or NULL. Default NULL (auto-detect).
<code>col</code>	Fill color. Default "steelblue".
<code>border</code>	Border color. Default "white".
<code>main</code>	Title. Default "Edge Weight Distribution".
<code>xlab</code>	X-axis label. Default "Weight".
<code>...</code>	Additional arguments passed to barplot .

Value

Invisibly returns the weight vector.

Examples

```
adj <- matrix(c(0, 2, 3, 2, 0, 1, 3, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
cograph::plot_edge_weights(adj)
```

`plot_heatmap`*Plot Network as Heatmap*

Description

Visualizes a network adjacency/weight matrix as a heatmap. Supports single networks, multi-cluster networks (block diagonal), and multi-layer networks (group_tna).

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

<code>x</code>	Network input: matrix, CographNetwork, cograph_network, tna, igraph, group_tna, or a list-like object with a <code>\$weights</code> matrix.
<code>cluster_list</code>	Optional list of character vectors defining node clusters. Creates a block-structured heatmap with clusters along diagonal.
<code>cluster_spacing</code>	Gap size between clusters (in cell units). Default 0.
<code>show_legend</code>	Logical: display color legend? Default TRUE.
<code>legend_position</code>	Position: "right" (default), "left", "top", "bottom", "none".
<code>legend_title</code>	Title for legend. Default "Weight".
<code>colors</code>	Color palette: vector of colors for gradient, or a palette name ("viridis", "heat", "blues", "reds", "greens", "diverging"). Default "viridis".
<code>limits</code>	Numeric vector c(min, max) for color scale. NULL for auto.
<code>midpoint</code>	Midpoint for diverging scales. NULL for auto (0 if data spans neg/pos).
<code>na_color</code>	Color for NA values. Default "grey90".
<code>show_values</code>	Logical: display values in cells? Default FALSE.
<code>value_size</code>	Text size for cell values. Default 2.5.
<code>value_color</code>	Color for cell value text. Default "black".
<code>value_fontface</code>	Font face for values: "plain", "bold", "italic", "bold.italic". Default "plain".
<code>value_fontfamily</code>	Font family for values: "sans", "serif", "mono". Default "sans".
<code>value_halo</code>	Halo color behind value labels for readability on dark cells. Set to a color (e.g., "white") to enable, or NULL (default) to disable.
<code>value_digits</code>	Decimal places for values. Default 2.
<code>show_diagonal</code>	Logical: show diagonal values? Default TRUE.
<code>diagonal_color</code>	Accepted for API compatibility; diagonal cells currently use the active fill scale unless hidden with <code>show_diagonal = FALSE</code> .
<code>cluster_labels</code>	Logical: show cluster/layer labels? Default TRUE.
<code>cluster_borders</code>	Logical: draw borders around clusters? Default TRUE.
<code>border_color</code>	Color for cluster borders. Default "black".
<code>border_width</code>	Width of cluster borders. Default 0.5.
<code>row_labels</code>	Row labels. NULL for auto (rownames or indices).
<code>col_labels</code>	Column labels. NULL for auto (colnames or indices).
<code>show_axis_labels</code>	Logical: show axis tick labels? Default TRUE.
<code>axis_text_size</code>	Size of axis labels. Default 8.
<code>axis_text_angle</code>	Angle for x-axis labels. Default 45.

title	Plot title. Default NULL.
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 <code>abs(value) < threshold</code> are set to zero. Default 0.
aspect_ratio	Aspect ratio. Default 1 (square cells).
...	Additional arguments (currently unused).

Details

For multi-cluster networks, provide `cluster_list` as a named list where each element is a vector of node names belonging to that cluster. The heatmap will be reordered to show clusters as blocks along the diagonal.

For `group_tna` objects (multiple separate networks), each network becomes a diagonal block. Off-diagonal blocks are empty (no inter-layer edges).

Value

A `ggplot2` object.

Examples

```
set.seed(1)
m <- matrix(runif(25), 5, 5)
rownames(m) <- colnames(m) <- LETTERS[1:5]
plot_heatmap(m)

# With clusters, values, and a different color scale
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D", "E"))
plot_heatmap(m, cluster_list = clusters, colors = "heat", show_values = TRUE)
```

plot_htna	<i>Plot Heterogeneous TNA Network (Multi-Group Layout)</i>
-----------	--

Description

Plots a TNA model with nodes arranged in multiple groups using geometric layouts:

- Circular: default for `layout = "auto"`, with groups on arcs
- Bipartite: two vertical columns or horizontal rows for exactly 2 groups
- Polygon: nodes along edges of a regular polygon for 3+ groups

Supports triangle (3), rectangle (4), pentagon (5), hexagon (6), and beyond.

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

<code>x</code>	A tna object, weight matrix, or <code>cograph_network</code> .
<code>node_list</code>	Node groups can be specified as: • A list of character vectors (node names per group) • A string column name from nodes data (e.g., "groups") • NULL to auto-detect from columns named: groups, cluster, community, etc. • NULL with <code>community</code> specified for algorithmic detection
<code>community</code>	Community detection method to use for auto-grouping. If specified, overrides <code>node_list</code> . See detect_communities for available methods: "louvain", "walk-trap", "fast_greedy", "label_prop", "infomap", "leiden".
<code>layout</code>	Layout type: "auto" (default), "bipartite", "polygon", or "circular". When "auto", uses the circular layout for any valid group count. "circular" places groups along arcs of a circle. Legacy values "triangle" and "rectangle" are supported as aliases for "polygon".

use_list_order	Logical. Use node_list order (TRUE) or weight-based order (FALSE). Only applies to bipartite layout.
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)) • Numeric vector of length n: Direct x-offsets for each node Only applies to bipartite layout.
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", "second", "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 for first group in bipartite layout. Default -2. Overridden by group_spacing if specified.
group2_pos	Position for second group in bipartite layout. Default 2. 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 (auto-computed from the largest group size). 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. Default 0.4 for visible curves.
group1_color	Color for first group nodes. Default "#4FC3F7".
group2_color	Color for second group nodes. Default "#fbb550".
group1_shape	Shape for first group nodes. Default "circle".
group2_shape	Shape for second group nodes. Default "square".

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 use the built-in group color palette.
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 use the built-in group shape palette.
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 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 drawn separately with curves that arc away from the opposing group. Default NULL (intra-group edges drawn normally by splot). Typical values: 0.3 to 1.0.
legend	Logical. Whether to show a legend. Default TRUE.
legend_position	Position for legend: "topright", "topleft", "bottomright", "bottomleft", "right", "left", "top", "bottom". Default "bottom".
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() . Default 0.8. The legend's symbols are sized from it. Like all cograph text it is scaled with the device, and with scale.
extend_lines	Logical or numeric. Draw extension lines from nodes. Only applies to bipartite layout. • FALSE (default): No extension lines • TRUE: Draw lines extending toward the other group (default length 0.1) • Numeric: Length of extension lines
scale	Scaling factor for spacing parameters. Use scale > 1 for high-resolution output (e.g., scale = 4 for 300 dpi). This scales polygon/circular radius and legend sizing; bipartite group positions are controlled by group1_pos, group2_pos, and group_spacing. Default 1.
nodes	Node metadata. Can be: • NULL (default): Use existing nodes data from cograph_network • Data frame: Must have label column for matching; if labels column exists, uses it for display text Display priority: labels column > label column (identifiers).
label_abbrev	Label abbreviation: NULL (none), integer (max chars), or "auto" (adaptive based on node count). Applied before passing to tplot.
...	Additional parameters passed to tplot().

Value

Invisibly returns the `tplot()` result: a `cograph_network` object. Called for the side effect of drawing.

Examples

```
# Create a 6-node network
mat <- matrix(runif(36, 0, 0.3), 6, 6)
diag(mat) <- 0
colnames(mat) <- rownames(mat) <- c("A", "B", "C", "D", "E", "F")

# Bipartite layout (2 groups)
groups <- list(Group1 = c("A", "B", "C"), Group2 = c("D", "E", "F"))
plot_htna(mat, groups)

# Polygon layout (3 groups)
groups3 <- list(X = c("A", "B"), Y = c("C", "D"), Z = c("E", "F"))
plot_htna(mat, groups3)
set.seed(1)
mat <- matrix(runif(36, 0, 0.3), 6, 6); diag(mat) <- 0
colnames(mat) <- rownames(mat) <- LETTERS[1:6]
groups <- list(G1 = LETTERS[1:3], G2 = LETTERS[4:6])
htna(mat, groups)
```

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 arranged inside elliptical cluster shells with full within-cluster and between-cluster edges drawn at the individual-node level. The **top layer** collapses each cluster into a single summary pie-chart node whose colored slice represents, by default, the cluster's share of the initial state distribution (see `summary_pie` for the alternative self-retention interpretation), with edges carrying the aggregated between-cluster weights. Dashed inter-layer lines connect each detail node to its corresponding summary node, making the hierarchical mapping explicit.

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 weight matrix, tna object, cograph_network, cluster_summary, or mcml/mcml_pc object (the latter from Nestimate::build_mcml_pc(), rendered undirected via its meta\$directed flag). When a cluster_summary is provided (e.g., from csum), all aggregation has already been performed and the cluster_list, aggregation, and nodes parameters are ignored. See the Input Formats section for details. |
| cluster_list | <p>How to assign nodes to clusters. Accepts:</p> <ul style="list-style-type: none"> • A named list of character vectors — each element contains the node names belonging to that cluster, and the list names become the cluster labels (e.g., list(GroupA = c("A", "B"), GroupB = c("C", "D"))). • A string giving a column name in the node metadata (from a cograph_network) to use as the grouping variable. • NULL — attempt auto-detection from common column names (cluster, group, etc.) in node metadata. <p>Ignored when x is a cluster_summary.</p> |
| expand | <p>Names of clusters whose member states are drawn as separate nodes in the top (macro) layer; "all" or TRUE expands every cluster. The bottom layer always shows the partition, so an expanded state appears as its own summary node while staying inside its cluster's shell below, linked by the dashed line. Default NULL draws one summary node per cluster.</p> <p>The expanded macro is re-counted from x with a refined partition (an expanded cluster contributes one group per member state), because a k x k aggregate cannot be disaggregated after the fact. That needs the source, so passing a pre-built cluster_summary or mcml instead of the data falls back to Nestimate::macro_network() and raises a cograph_expand_unavailable error when that is not available.</p> |
| mode | <p>What values to display on edges:</p> <p>"weights" (default) Shows raw aggregated edge values. Useful when absolute magnitudes (e.g., total co-occurrences) matter.</p> |

	"tna" Row-normalizes the summary matrix so each row sums to 1, producing transition probabilities. Automatically enables <code>edge_labels</code> and <code>summary_edge_labels</code> unless you explicitly set them to <code>FALSE</code> .
theme	Visual preset controlling node and edge styling. One of: "classic" (default) The historical look — pie-chart nodes and straight summary edges, with thin borders and slightly larger detail nodes. "rich" Donut nodes on both layers plus curved (qgraph-style) summary edges and splot self-loops. "light" Like "rich" but with no cluster-shell outline and a softer shell fill. The granular style arguments (<code>node_donut</code>, <code>curved_edges</code>) override the preset when supplied.
layer_spacing	Vertical position of the summary (top) layer, which is what decides how tall the figure is. • <code>NULL</code> (default): placed automatically, just clear of the bottom layer (<code>inter_layer_gap</code> sets the clearance). The figure then has a fixed shape, and a taller image only adds white space. • <code>"fill"</code>: the gap between the layers is stretched so the figure uses the full height of the image it is drawn on. Change the image height and the plot follows. Shapes stay round; only the space between the layers grows. Never tighter than the automatic layout. • A single positive number: the distance from the centre of the bottom layer to the centre of the summary layer, in the same units as <code>spacing</code>. Overrides <code>inter_layer_gap</code>. A value small enough to overlap the two layers raises a <code>cograph_layers_overlap</code> warning.
spacing	Distance from the center to each cluster's position in the bottom layer. Larger values spread clusters farther apart. Default 3.
shape_size	Radius of each cluster's elliptical shell in the bottom layer. Increase when nodes overlap or shells feel cramped. Default 1.2.
summary_size	Size of the pie-chart summary nodes in the top layer. Controls the visual radius of each pie chart. Default 4.
skew_angle	Perspective tilt angle in degrees (0–90). At 0 the bottom layer is viewed from directly above (fully circular); at 90 it collapses to a flat line. Values around 45–70 give a natural table-top perspective. Default 60.
aggregation	Method for collapsing individual edge weights into between-cluster and within-cluster summaries: "sum" (default) Total flow — appropriate when you care about the volume of all transitions between clusters. "mean" Average flow per node pair — useful when clusters differ in size and you want a size-normalized comparison. "max" Strongest single edge — highlights the dominant connection between each pair of clusters. Ignored when <code>x</code> is a <code>cluster_summary</code>.
minimum	Edge weight threshold. Edges with absolute weight below this value are not drawn. Set to a small positive value (e.g., 0.01) to remove visual noise from near-zero edges. Default 0 (show all).

colors	Character vector of colors for the clusters. The first color is applied to the first cluster, and so on. Must have length equal to the number of clusters, or it will be recycled. When NULL (default), colors are auto-generated from a colorblind-safe palette.
legend	Logical. Whether to draw a legend mapping cluster names to colors. Default TRUE.
show_labels	Logical. Show node labels in the bottom layer. Default TRUE. Set to FALSE for dense networks where labels create clutter.
nodes	Node metadata data frame for custom display labels. Must contain a label column whose values match the row/column names of the weight matrix. If a labels column also exists, those values are used as display text (e.g., full names instead of codes). Display priority: labels column > label column. Ignored when x is a cluster_summary or cograph_network (which carries its own node metadata).
label_size	Text size (cex) for bottom-layer node labels. NULL (default) auto-scales to 0.6. Increase for readability in publication figures; decrease for dense networks.
label_abbrev	Controls label abbreviation to reduce overlap: • NULL — no abbreviation (show full labels). • An integer — truncate labels to this many characters. • "auto" — adaptively abbreviates based on the total number of nodes: more nodes triggers shorter abbreviations.
node_size	Size of individual detail nodes in the bottom layer. This controls the pie-chart radius for each node. Default 2.4.
node_shape	Shape for detail nodes in the bottom layer. Supported values: "circle", "square", "diamond", "triangle". Can be a single value applied to all nodes or a character vector of length equal to the number of nodes (one shape per node). Default "circle".
cluster_shape	Accepted for backward compatibility. Summary nodes are currently drawn as pie charts, so this parameter does not change their shape.
title	Main plot title displayed above the figure. Default NULL (no title).
subtitle	Subtitle displayed below the title. Default NULL (no subtitle).
title_size	Text size (cex.main) for the title. Default 1.2.
subtitle_size	Text size (cex.sub) for the subtitle. Default 0.9.
legend_position	Where to place the legend: "right", "left", "top", "bottom", or "none" to suppress it entirely. Default "right".
legend_size	Text size (cex) for legend labels. Default 0.7.
legend_pt_size	Point size (pt.cex) for legend symbols. Default 1.2.
summary_labels	Logical. Show cluster name labels next to the summary pie-chart nodes in the top layer. Default TRUE.
summary_label_size	Text size for summary labels. Default 0.8.

summary_label_position	Position of summary labels relative to nodes: 1 = below, 2 = left, 3 = above, 4 = right. Default 3 (above).
summary_label_color	Color for summary labels. Default "gray20".
summary_arrows	Logical. Draw arrowheads on summary-layer directed edges. Default TRUE. For fully undirected networks prefer directed = FALSE, which also suppresses these arrowheads and draws each symmetric edge pair only once.
summary_arrow_size	Size of arrowheads on summary edges. Default 0.10.
node_donut	Logical or NULL. Force donut node rendering on (TRUE) or off (FALSE), overriding theme. NULL (default) follows the preset (donut for "rich"/"light").
node_donut_inner_ratio	Hole size (0–1) of the detail-node donut ring. Default 0.55.
summary_donut_inner_ratio	Hole size (0–1) of the top-layer summary donut ring. 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. Force curved summary edges on or off, overriding theme. NULL (default) follows the preset.
summary_curve	Numeric or NULL. Curvature of curved summary edges (only used when curved). NULL auto-selects (0.25 for directed, straight for undirected).
summary_pie	Character scalar controlling what the colored slice of the top-layer pie chart represents. One of: "init" (default) The cluster's share of the initial state distribution (<code>cs\$macro\$init[i]</code>). Answers "how often do sequences start in this cluster?" Summed across clusters the colored slices equal 1. "self" The cluster's self-retention share of out-strength (<code>bw[i, i] / rowSums(bw)[i]</code>). Answers "how sticky is this cluster — how much of its outgoing flow loops back to itself?" Each pie is normalized independently.
edge_color_by	How to color edges on all layers: "auto" (default) Color edges by their cluster when the weights are non-negative (transition networks), but switch to sign-based coloring automatically when any negative weight is present (correlation / association networks). "cluster" Always color edges by the source cluster's color. "sign" Always color edges by weight sign — positive in <code>edge_positive_color</code> , negative in <code>edge_negative_color</code> . Sign coloring uses each edge's absolute weight for the threshold (minimum) and line-width scaling, so negative edges are drawn rather than dropped.
edge_positive_color	Color for positive-weight edges when sign coloring is active. Default "#2E7D32" (green).

edge_negative_color	Color for negative-weight edges when sign coloring is active. Default "#C62828" (red).
between_arrows	Logical. Draw arrowheads on between-cluster edges in the bottom layer. Default FALSE.
edge_width_range	Numeric vector $c(\min, \max)$ controlling the line-width range for within-cluster edges in the bottom layer. The weakest edge gets $\min$ and the strongest gets $\max$. Default $c(0.3, 1.3)$.
between_edge_width_range	Numeric vector $c(\min, \max)$ for between-cluster edges in the bottom layer (shell-to-shell lines). Default $c(0.5, 2.0)$.
summary_edge_width_range	Numeric vector $c(\min, \max)$ for summary edges in the top layer. Default $c(0.5, 2.0)$.
edge_alpha	Transparency (0–1) for within-cluster edges. Lower values make these edges more subtle, keeping focus on between-cluster structure. Default 0.35.
between_edge_alpha	Transparency (0–1) for between-cluster edges in the bottom layer. Default 0.6.
summary_edge_alpha	Transparency (0–1) for summary-layer edges. Default 0.7.
inter_layer_alpha	Transparency (0–1) for the dashed inter-layer lines connecting detail nodes to their summary node. Lower values make these scaffolding lines less visually dominant. Default 0.5.
edge_labels	Logical. Show numeric weight labels on within-cluster edges. Default FALSE (automatically set to TRUE when mode = "tna").
edge_label_size	Text size for within-cluster edge labels. Default 0.5.
edge_label_color	Color for within-cluster edge labels. Default "gray40".
edge_label_digits	Number of decimal places for edge weight labels on both layers. Default 2.
summary_edge_labels	Logical. Show numeric weight labels on summary-layer edges. Default FALSE (automatically set to TRUE when mode = "tna").
summary_edge_label_size	Text size for summary edge labels. Default 0.6.
top_layer_scale	Numeric vector $c(x_scale, y_scale)$ controlling the horizontal and vertical radii of the oval on which summary nodes are placed, as multiples of spacing. Widen with $c(1.0, 0.25)$ or flatten with $c(0.8, 0.15)$ to adjust the top-layer shape. Default $c(0.8, 0.25)$.
inter_layer_gap	Vertical gap between the top of the bottom layer and the bottom of the top layer, as a multiple of spacing. Increase to separate the layers more. Default 0.6.

node_radius_scale	Radius of the circle on which nodes are arranged inside each cluster shell, as a fraction of shape_size. Increase to push nodes outward toward the shell border; decrease to pack them tighter. Default 0.55.
shell_alpha	Fill transparency (0–1) for cluster shells. Higher values make shells more opaque, giving stronger visual grouping but potentially obscuring edges. Default 0.15.
shell_border_width	Line width for cluster shell borders. Default 0.75 (thin). theme = "light" drops the outline entirely.
node_border_color	Border color for detail nodes in the bottom layer. Default "gray30".
node_border_width	Line width for detail-node borders in the bottom layer. Default 0.4 (thin). Increase for heavier outlines.
summary_border_color	Border color for summary pie-chart nodes. Default "gray20".
summary_border_width	Border line width for summary nodes. Default 0.6 (thin).
label_color	Text color for detail node labels. Default "gray20".
label_position	Accepted for backward compatibility. Detail labels are currently positioned automatically to the left or right of each node.
directed	Logical or NULL. NULL (default) auto-detects: a cluster_summary/mcml input uses its own \$meta\$directed flag; other objects use their \$directed field when present; a plain matrix is undirected when symmetric (the same contract as splot). When TRUE, every non-zero cell of the weight matrices is drawn as a directed edge with an arrowhead. When FALSE (undirected, e.g. co-occurrence weights): arrowheads are suppressed on all three edge layers (within-cluster, between-cluster, and summary), each symmetric pair is drawn once instead of twice (the upper triangle is used; a warning is issued if the weights are not symmetric), edge labels move to the edge midpoint, and matrix input is aggregated with type = "cooccurrence" (symmetrized counts) instead of the row-normalized type = "tna". Overrides summary_arrows and between_arrows.
...	Additional arguments (currently unused).

Details

Use `plot_mcml` when you need a simultaneous micro/macro view of cluster structure — the bottom layer reveals internal cluster dynamics while the top layer provides a bird's-eye summary. For a flat multi-cluster plot without the summary layer, see [plot_mtna](#). For stacked multilevel/multiplex layers, see [plot_mlna](#).

Two workflows:

1. **Direct:** pass a weight matrix (or `tna` / `cograph_network` object) together with `cluster_list`. The function calls `csum` internally to compute aggregated weights.
2. **Pre-computed:** call `csum` yourself, inspect or modify the result, then pass the `cluster_summary` object as `x`. This avoids redundant computation when you plot the same clustering repeatedly with different visual settings.

Mode:

- "weights" (default) — displays raw aggregated edge values. Use this when the absolute magnitude of transitions matters.
- "tna" — row-normalizes the summary matrix to transition probabilities (rows sum to 1) and automatically enables edge labels on both layers (unless you explicitly set `edge_labels` or `summary_edge_labels` to FALSE).

Directionality: `directed = NULL` (default) auto-detects directedness from the input: `cluster_summary/mcml` objects carry it in `$meta$directed`, and plain matrices are treated as undirected when symmetric. Directed edges get arrowheads; undirected weights (e.g., co-occurrence aggregations) are drawn as a single plain line per symmetric pair on every layer, with no arrowheads. Pass `directed = TRUE/FALSE` to override the detection.

Layout logic: 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 top-layer summary nodes are placed on an oval above the bottom layer whose proportions are controlled by `top_layer_scale`.

Value

Invisibly returns the `cluster_summary` object used for plotting. This object can be passed back to `plot_mcml()` to avoid recomputation, inspected with `print()`, or fed to [as_tna](#) for further analysis.

Input Formats

`x` accepts the following types:

matrix A square numeric weight matrix with row/column names matching the node identifiers in `cluster_list`.

tna A TNA model object. The `$weights` matrix is extracted automatically.

cograph_network A cograph network object. Weights are extracted via `to_matrix()` and node metadata (display labels) is read from the `$nodes` data frame.

cluster_summary A pre-computed summary from [csum](#). When this type is passed, the `cluster_list`, `aggregation`, and `nodes` parameters are ignored because the summary already contains everything needed.

mcml / mcml_pc A Nestimate multi-cluster multi-layer object; handled exactly like a `cluster_summary`, with `mcml_pc` rendered undirected via its `meta$directed` flag.

Edge Types

The plot contains four distinct edge categories, each with its own set of visual parameters:

Within-cluster (bottom) Edges connecting nodes inside the same cluster shell. Controlled by `edge_width_range`, `edge_alpha`, `edge_labels`, `edge_label_size`, `edge_label_color`, and `edge_label_digits`.

Between-cluster (bottom) Edges from one cluster shell to another, drawn between shell borders. Controlled by `between_edge_width_range` and `between_edge_alpha`.

Summary (top) Edges between summary pie-chart nodes in the top layer. Controlled by `summary_edge_width_range`, `summary_edge_alpha`, `summary_edge_labels`, `summary_edge_label_size`, `summary_arrows`, and `summary_arrow_size`.

Inter-layer (dashed) Dashed lines connecting each detail node to its cluster's summary node. Controlled by `inter_layer_alpha`.

Customization Quick Reference

Visual element	Key parameters
Cluster spacing / perspective	<code>spacing</code> , <code>skew_angle</code>
Cluster shell appearance	<code>shape_size</code> , <code>shell_alpha</code> , <code>shell_border_width</code> , <code>colors</code>
Detail nodes	<code>node_size</code> , <code>node_shape</code> , <code>node_border_color</code>
Detail labels	<code>show_labels</code> , <code>label_size</code> , <code>label_abbrev</code> , <code>label_color</code> , <code>label_position</code>
Summary nodes	<code>summary_size</code> , <code>summary_border_color</code> , <code>summary_border_width</code>
Summary labels	<code>summary_labels</code> , <code>summary_label_size</code> , <code>summary_label_color</code> , <code>summary_label_position</code>
Within-cluster edges	<code>edge_width_range</code> , <code>edge_alpha</code> , <code>edge_labels</code>
Between-cluster edges	<code>between_edge_width_range</code> , <code>between_edge_alpha</code>
Summary edges	<code>summary_edge_width_range</code> , <code>summary_edge_alpha</code> , <code>summary_edge_labels</code> , <code>summary_arrows</code>
Directed vs undirected	<code>directed</code>
Inter-layer lines	<code>inter_layer_alpha</code>
Top-layer layout	<code>top_layer_scale</code> , <code>inter_layer_gap</code>
Title / legend	<code>title</code> , <code>subtitle</code> , <code>legend</code> , <code>legend_position</code>

See Also

[csum](#) for pre-computing aggregated cluster data, [plot_mtna](#) for flat multi-cluster visualization (no summary layer), [plot_mlna](#) for stacked multilevel/multiplex layer visualization, [aggregate_weights](#) for the low-level weight aggregation used internally, [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)

cs <- csum(regulation_net, clusters)
plot_mcml(cs, mode = "tna", edge_labels = TRUE)
```

plot_mixed_network	<i>Plot Mixed Network</i>
--------------------	---------------------------

Description

Plot a network combining symmetric (undirected) and asymmetric (directed) matrices with appropriate edge styling.

Creates a network visualization combining edges from a symmetric matrix (rendered as straight undirected edges) and an asymmetric matrix (rendered 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 representing undirected relationships. These edges will be drawn straight without arrows.
asym_matrix	An asymmetric matrix representing directed relationships. These edges will be drawn curved with arrows. Reciprocal edges curve in opposite directions.
layout	Layout algorithm or coordinate matrix. Default "oval".
sym_color	Color for symmetric/undirected edges. Default "ivory4".
asym_color	Color for asymmetric/directed edges. Can be a single color or a vector of two colors for positive/negative directions. Default "#003355" (dark blue, matching TNA style).
curvature	Curvature magnitude for directed edges. Default 0.3.
edge_width	Edge width(s). If NULL (default), scales automatically by edge weight like TNA plots. Pass a numeric value to override.

node_size	Node size. Default 7.
title	Plot title. Default NULL.
threshold	Minimum absolute edge weight to display. Values with <code>abs(value) < threshold</code> are set to zero (edge removed). Default 0. Zero-weight edges are always removed regardless of this setting.
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 named numeric vector of initial state probabilities (length = number of nodes). When provided, nodes are drawn as donuts with the fill proportion equal to the initial probability. Default NULL.
...	Additional arguments passed to <code>splot()</code> .

Value

Invisibly returns a list with the combined edge data and filtered symmetric/asymmetric matrices.

Examples

```
# Create symmetric matrix (undirected)
sym <- matrix(0, 4, 4, dimnames = list(LETTERS[1:4], LETTERS[1:4]))
sym[1,2] <- sym[2,1] <- 0.5
sym[3,4] <- sym[4,3] <- 0.6

# Create asymmetric matrix (directed)
asym <- matrix(0, 4, 4, dimnames = list(LETTERS[1:4], LETTERS[1:4]))
asym[1,3] <- 0.7
asym[3,1] <- 0.3
asym[2,4] <- 0.8
asym[4,2] <- 0.4

# Plot combined network
plot_mixed_network(sym, asym, title = "Mixed Network")
```

plot_mlna

Multilevel Network Visualization

Description

Visualizes multilevel/multiplex networks where multiple layers are stacked in a 3D perspective view. Each layer contains nodes connected by solid edges (within-layer), while dashed lines connect nodes between adjacent layers (inter-layer edges). Each layer is enclosed in a parallelogram shell giving a pseudo-3D appearance.

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	Layers can be specified as: • A list of character vectors (node names per layer) • A string column name from nodes data (e.g., "layer") • NULL to auto-detect from columns named: layer, layers, groups, etc. • NULL with community specified for algorithmic detection
community	Community detection method to use for auto-layering. If specified, overrides layer_list. See detect_communities for available methods: "louvain", "walk-trap", "fast_greedy", "label_prop", "infomap", "leiden".
layout	Node layout within layers: "horizontal" (default) spreads nodes horizontally, "circle" arranges nodes in an ellipse, "spring" uses force-directed placement based on within-layer connections.
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 (for 3D effect). Default 4.
skew_angle	Angle of perspective skew in degrees. Default 25.
node_spacing	Node placement ratio within layer (0-1). Default 0.7. Higher values spread nodes closer to the layer edges.
colors	Vector of colors for each layer. Default auto-generated.
shapes	Vector of shapes for each layer. Default cycles through "circle", "square", "diamond", "triangle".
edge_colors	Vector of edge colors by source layer. If NULL (default), uses darker versions of layer colors.
within_edges	Logical. Show edges within layers (solid lines). Default TRUE.
between_edges	Logical. Show edges between adjacent layers (dashed lines). Default TRUE.
between_style	Line style for between-layer edges. Default 2 (dashed). Use 1 for solid, 3 for dotted.
show_border	Logical. Draw parallelogram shells around layers. Default TRUE.
legend	Logical. Whether to show legend. Default TRUE.

legend_position	Position for legend. Default "topright".
curvature	Edge curvature for within-layer edges. Default 0.15.
node_size	Size of nodes. Default 3.
minimum	Minimum edge weight threshold. Edges below this are hidden. Default 0.
scale	Scaling factor for spacing parameters. Use scale > 1 for high-resolution output (e.g., scale = 4 for 300 dpi). This multiplies layer_spacing, layer_width, and layer_depth to maintain proper proportions at higher resolutions. Default 1.
show_labels	Logical. Show node labels. Default TRUE.
nodes	Node metadata. Can be: • NULL (default): Use existing nodes data from cograph_network • Data frame: Must have label column for matching; if labels column exists, uses it for display text Display priority: labels column > label column (identifiers).
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

```
set.seed(42)
m <- matrix(runif(225, 0, 0.3), 15, 15); diag(m) <- 0
nodes <- paste0("N", 1:15)
colnames(m) <- rownames(m) <- nodes
layers <- list(Macro = nodes[1:5], Meso = nodes[6:10], Micro = nodes[11:15])
plot_mlna(m, layers)

plot_mlna(m, layers, layout = "circle", between_style = 2, minimum = 0.1)

set.seed(1)
nodes <- paste0("N", 1:9)
m <- matrix(runif(81, 0, 0.3), 9, 9); diag(m) <- 0
colnames(m) <- rownames(m) <- nodes
layers <- list(L1 = nodes[1:3], L2 = nodes[4:6], L3 = nodes[7:9])
mlna(m, layers)
```

plot_ml_heatmap

Multilayer Network Heatmap

Description

Visualizes multiple network layers as heatmaps on tilted 3D-perspective planes, similar to the plot_mlna network visualization style.

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, cograph_network, or a single matrix with layer_list specified.
layer_list	Optional list defining layers, column name string, or NULL for auto-detection from cograph_network nodes.
colors	Color palette: "viridis", "heat", "blues", "reds", "inferno", "plasma", or a vector of colors. Default "viridis".

layer_spacing	Vertical spacing between layers, in data units. A plane is $nrow(x) * compress$ units tall, so a fixed spacing that suits a small network makes a larger one overlap itself. NULL (the default) scales the spacing to the plane so planes never collide; pass a number for the older absolute behavior.
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 of the matrix? Default TRUE. Without them a plane is an anonymous grid and a reader cannot tell which cell is which pair. Every plane shares one node ordering, so the names are drawn once, against the front plane: rows down its left edge, columns along its lower edge.
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 (rendered as background). Default 0.

Value

A ggplot2 object.

Examples

```
set.seed(1)
layers <- list(
  L1 = matrix(runif(16), 4, 4),
  L2 = matrix(runif(16), 4, 4),
  L3 = matrix(runif(16), 4, 4))
plot_ml_heatmap(layers)
plot_ml_heatmap(layers, show_connections = TRUE, colors = "plasma")
```

plot_motifs	<i>Plot a motif/subgraph result</i>
-------------	-------------------------------------

Description

Tab-completion-friendly wrapper around the `plot.cograph_motif_result` S3 method. Functionally identical to `plot(x, ...)` on a `cograph_motif_result` object, but exposes the `type / n / ncol / colors` arguments to editor autocompletion.

Usage

```
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,
  ...
)
```

Arguments

<code>x</code>	A <code>cograph_motif_result</code> object from <code>motifs()</code> or <code>subgraphs()</code> .
<code>type</code>	Plot type: "triads" Network diagrams of specific node triples (instance mode) or falls back to patterns (census mode). Instance panels use a canonical representative of the MAN class: concrete labels identify participants, not their observed node-role orientation. Each panel title reads "<MAN code>: <description>"

	(e.g. "030T: Feed-forward") and, in census mode, appends the z-score and a significance star (* $p < .05$, ** $p < .01$, *** $p < .001$). Arranged in a grid.
"types"	Bar chart of MAN type frequencies. In census mode bars are colored by significance direction (see colors); in instance mode bars use a single fill because per-type significance would need an aggregation rule across multiple node-triple rows of the same type.
"significance"	Z-score bars per row of <code>x\$results</code> . In census mode each bar is one MAN type; in instance mode each bar is one concrete node-triple, labeled "<triple> [<MAN code>: <description>]". Bars are colored with the same three-tone rule (see colors). Requires <code>significance = TRUE</code> in the <code>motifs()</code> call.
"patterns"	Abstract MAN pattern diagrams showing the edge structure of each triad type. In census mode panel nodes are filled by significance direction (red sig over / blue sig under / grey ns); in instance mode panels use a single fill, same reason as "types".
<code>n</code>	Maximum number of items to plot. Default 15.
<code>ncol</code>	Number of columns in the triad/pattern grid. Default 5.
<code>colors</code>	Two-element color vector mapped to a three-tone significance scale (used by <code>type = "significance"</code> , plus <code>type = "types"</code> and <code>type = "patterns"</code> in census mode): <code>colors[1]</code> fills items that are significantly under-represented ($p < .05$ and $z < 0$); <code>colors[2]</code> fills items that are significantly over-represented ($p < .05$ and $z > 0$); everything else is filled neutral grey ("9E9E9E"). Default <code>c("#2166AC", "#B2182B")</code> (blue for under, red for over). When significance was not run, <code>type = "types"</code> falls back to a single <code>colors[1]</code> fill and patterns nodes use <code>colors[1]</code> .
<code>node_size</code>	Triad node radius (relative). Default 5. (<code>type = "triads"</code> only.)
<code>label_size</code>	Triad node-label font size in points. Default 11.
<code>title_size</code>	Per-panel title font size in points. Default 12.
<code>stats_size</code>	Per-panel statistics caption font size in points (e.g., $n=34$ $z=-55.3$ $p<.001$). Default 13.
<code>legend_size</code>	Bottom legend font size in points. Default 13.
<code>legend</code>	Logical. Show the abbreviation legend strip below the triad grid. Default TRUE. (<code>type = "triads"</code> only.)
<code>motif_color</code>	Color of triad nodes/edges/labels. Default "#800020" (deep burgundy). (<code>type = "triads"</code> only.)
<code>spacing</code>	Triangle spread inside each panel; > 1 pulls nodes inward, < 1 pushes them apart. Default 1.
<code>base_size</code>	Base font size for the ggplot2 themes used by <code>type = "types"</code> and <code>type = "significance"</code> . Default 12.
<code>...</code>	Additional arguments passed to internal plot helpers.

Value

Invisibly returns the input `x` (or the underlying ggplot for the "types" and "significance" types, matching the S3 method).

See Also[motifs](#), [subgraphs](#)**Examples**

```
g <- igraph::sample_gnp(20, 0.2, directed = TRUE)
m <- motifs(g)
plot_motifs(m)
plot_motifs(m, type = "types")
```

`plot_mtna`*Multi-Cluster TNA Network Plot*

Description

Visualizes multiple network clusters with summary edges between clusters and individual edges within clusters. Each cluster is displayed as a shell shape containing its nodes.

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

<code>x</code>	A tna object, weight matrix, or <code>cograph_network</code> .
<code>cluster_list</code>	Clusters can be specified as: • A list of character vectors (node names per cluster) • A string column name from nodes data (e.g., "groups") • <code>NULL</code> with <code>community</code> specified for auto-detection • <code>NULL</code> with a <code>cograph_network</code> that has a common cluster/group column
<code>community</code>	Community detection method to use for auto-clustering. If specified, overrides <code>cluster_list</code> . See detect_communities for available methods.
<code>layout</code>	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 (0-1 relative to shape_size). Default 0.5.
colors	Vector of colors for each cluster. Default auto-generated.
shapes	Vector of shapes for each cluster. Defaults cycle through "circle", "square", "diamond", "triangle", "pentagon", "hexagon", "star", and "cross"; summary shells draw non-shell shapes with the circular fallback.
edge_colors	Vector of edge colors by source cluster. Default auto-generated.
bundle_edges	Logical. Bundle inter-cluster edges through channels. Default TRUE.
bundle_strength	How tightly to bundle edges (0-1). Default 0.8.
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. Draw a border 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. Default 3.
layout_margin	Margin around the layout as fraction of range. Default 0.15.
scale	Scaling factor for high-resolution output. Values greater than 1 reduce node, edge, label, and legend sizes by $\sqrt{\text{scale}}$ while leaving cluster spacing and shape_size unchanged. Default 1.
show_labels	Logical. Show node labels inside clusters. Default FALSE.
nodes	Node metadata. Can be: • NULL (default): Use existing nodes data from <code>cograph_network</code> • Data frame: Must have <code>label</code> column for matching; if <code>labels</code> column exists, uses it for display text Display priority: <code>labels</code> column > <code>label</code> column (identifiers).
label_size	Label text size. Default NULL (auto-scaled).
label_abbrev	Label abbreviation: NULL (none), integer (max chars), or "auto" (adaptive based on node count).
cluster_shape	Accepted for compatibility; currently unused. Use <code>shapes</code> to control cluster shell shapes.
...	Additional parameters passed to <code>plot_tna()</code> .

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

```
set.seed(42)
nodes <- paste0("N", 1:20)
m <- matrix(runif(400, 0, 0.3), 20, 20); diag(m) <- 0
colnames(m) <- rownames(m) <- nodes
clusters <- list(N = nodes[1:5], E = nodes[6:10],
                 S = nodes[11:15], W = nodes[16:20])
plot_mtna(m, clusters, summary_edges = TRUE)
set.seed(1)
nodes <- paste0("N", 1:12)
m <- matrix(runif(144, 0, 0.3), 12, 12); diag(m) <- 0
colnames(m) <- rownames(m) <- nodes
clusters <- list(C1 = nodes[1:4], C2 = nodes[5:8], C3 = nodes[9:12])
mtna(m, clusters)
```

`plot_netobject_group` *Plot a Group of Nestimate netobjects*

Description

Creates a multi-panel plot for a `netobject_group` list, one panel per group. Mirrors `plot_group_permutation()` in structure.

Usage

```
plot_netobject_group(
  x,
  nrow = NULL,
  ncol = NULL,
  common_scale = TRUE,
  title_prefix = NULL,
  combined = TRUE,
  ...
)

## S3 method for class 'netobject_group'
plot(x, ...)
```

Arguments

x	A netobject_group object (named list of netobjects).
nrow	Integer: number of rows in the panel grid. Auto-computed if NULL.
ncol	Integer: number of columns in the panel grid. Auto-computed if NULL.
common_scale	Logical: use the same maximum weight across all panels? Default TRUE.
title_prefix	Character: optional prefix added before each group name in panel titles.
combined	Logical: when TRUE (default), arrange the panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw each panel into the active device without altering <code>par()</code> , e.g. when laying panels out yourself with <code>panel_layout()</code> .
...	Additional arguments passed to <code>splot()</code> .

Value

Invisibly returns x. With a single group the `splot()` result for that panel (a `cograph_network`) is returned instead, and with an empty group list NULL.

Examples

```
mat <- matrix(c(0, .5, .3, .5, 0, .4, .3, .4, 0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A", "B", "C")
net1 <- as_cograph(mat)
net2 <- as_cograph(mat * 0.5)
grp <- structure(list(G1 = net1, G2 = net2), class = c("netobject_group", "list"))
plot_netobject_group(grp)
```

plot_netobject_ml	<i>Plot a Multilevel Nestimate netobject</i>
-------------------	--

Description

Creates a side-by-side plot for a `netobject_ml` object, showing the between-person and within-person networks.

Usage

```
plot_netobject_ml(
  x,
  layout = NULL,
  common_scale = TRUE,
  titles = c("Between-person", "Within-person"),
  combined = TRUE,
  ...
)

## S3 method for class 'netobject_ml'
plot(x, ...)
```

Arguments

<code>x</code>	A <code>netobject_ml</code> object with <code>\$between</code> and <code>\$within</code> networks.
<code>layout</code>	Character: layout algorithm. Default <code>NULL</code> , which resolves to "oval" (deterministic).
<code>common_scale</code>	Logical: use the same maximum weight for both panels? Default <code>TRUE</code> .
<code>titles</code>	Character vector of length 2: panel titles. Default <code>c("Between-person", "Within-person")</code> .
<code>combined</code>	Logical: when <code>TRUE</code> (default), draws both panels in an internal 1 x 2 grid. Set to <code>FALSE</code> to render into a layout the caller already configured (e.g. via panel_layout()).
<code>...</code>	Additional arguments passed to <code>splot()</code> .

Value

Invisibly returns `x`.

Examples

```
mat <- matrix(c(0, .5, .3, .5, 0, .4, .3, .4, 0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A", "B", "C")
btw <- as_cograph(mat)
wth <- as_cograph(mat * 0.6)
ml <- structure(list(between = btw, within = wth), class = c("netobject_ml", "list"))
plot_netobject_ml(ml)
```

plot_network_evolution

Plot Network Evolution (Small Multiples)

Description

Displays a network at different time points side by side. Accepts an edge list data frame with a time column, or a pre-built list of networks. All panels share the same node layout for visual comparison.

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, and a time column, OR a list of network objects (matrices, igraph, etc.).
time	Character. Name of the time/group column in x. Ignored if x is a list.
slices	Integer or NULL. Number of equal-width time bins. Default NULL uses unique values of the time column.
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. Default NULL (auto from time values).
layout	Layout specification. Default "spring".
ncol	Integer. Grid columns. Default auto.
node_size	Numeric. Default 5.
seed	Integer or NULL. Default 42.
combined	Logical: when TRUE (default), arrange period panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw into a layout the caller has already configured (e.g. via <code>panel_layout()</code>).
...	Additional arguments passed to <code>splot</code> .

Value

Invisible list of per-panel networks or edge-list data frames.

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")
cograph::plot_network_evolution(edges, time = "week", cumulative = TRUE)
```

plot_net_bootstrap_group

Plot a Group Bootstrap Result

Description

Plots each cluster's `net_bootstrap` in a grid, routing every panel through `splot.net_bootstrap` so significance styling (solid vs dashed edges) is preserved. Earlier versions extracted `bs$original` per cluster and handed plain `netobjects` to `splot()`, which dispatches to `splot.netobject` — that path has no concept of significance, so every edge rendered identically.

Usage

```
plot_net_bootstrap_group(
  x,
  nrow = NULL,
  ncol = NULL,
  common_scale = TRUE,
  combined = TRUE,
  ...
)

## S3 method for class 'net_bootstrap_group'
plot(x, ...)
```

Arguments

x	A net_bootstrap_group object (list of net_bootstrap).
nrow, ncol	Grid dimensions. Defaults to auto-computed square layout.
common_scale	Logical: use the same maximum weight across panels? Default TRUE.
combined	Logical: when TRUE (default), arrange panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to FALSE to draw into a layout the caller already configured (e.g. via panel_layout()).
...	Additional arguments passed to <code>splot.net_bootstrap</code> (e.g. <code>display = "significant"</code> , <code>show_stars = FALSE</code>).

Value

Invisibly returns x. With a single group the [splot\(\)](#) result for that panel (a `cograph_network`) is returned instead, and with an empty group list NULL.

Examples

```
set.seed(1)
seqs <- data.frame(T1 = sample(c("A","B","C"), 30, replace = TRUE),
                   T2 = sample(c("A","B","C"), 30, replace = TRUE))
grp <- Nestimate::cluster_network(seqs, k = 2)
gbs <- Nestimate::bootstrap_network(grp, iter = 10)
plot_net_bootstrap_group(gbs)
```

plot_net_stability	<i>Plot Centrality Stability Results</i>
--------------------	--

Description

Visualizes the centrality stability analysis from a `net_stability` object. Shows how centrality correlations drop as cases are removed.

Usage

```
plot_net_stability(x, ...)
```

Arguments

`x` A `net_stability` object (from `Nestimate::centrality_stability`).

`...` Additional graphical arguments.

Value

Invisibly returns `x`.

Examples

```
set.seed(1)
seqs <- data.frame(T1 = sample(c("A", "B", "C"), 30, replace = TRUE),
                  T2 = sample(c("A", "B", "C"), 30, replace = TRUE))
net <- Nestimate::build_network(seqs, method = "tna")
cs <- Nestimate::centrality_stability(net, iter = 10)
plot_net_stability(cs)
```

plot_robustness	<i>Plot Network Robustness</i>
-----------------	--------------------------------

Description

Creates a visualization of network robustness showing the fraction of remaining nodes in the largest connected component during sequential node/edge removal. Supports comparison of multiple attack strategies.

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 , or named arguments to pass networks for on-the-fly computation.
x	Network for computing robustness on-the-fly.
measures	Character vector of attack strategies to compare. Default c("betweenness", "degree", "random").
colors	Named vector of colors. Default: green=Degree, red=Betweenness, blue=Random (matching Nature paper style).
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. Default 1000.
seed	Random seed. Default NULL.
type	Removal type. Default "vertex".

Value

Invisibly returns combined data frame of all robustness results.

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_pa(50, m = 2, directed = FALSE)

  # Quick comparison of all strategies
  plot_robustness(x = g, n_iter = 20)

  # Or compute separately
  rob1 <- robustness(g, measure = "betweenness")
  rob2 <- robustness(g, measure = "degree")
  rob3 <- robustness(g, measure = "random", n_iter = 20)
  plot_robustness(rob1, rob2, rob3)
}
```

Description

Visualize higher-order pathways as smooth blobs overlaid on a network layout. Source nodes are blue, target nodes are red.

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, or simplicial_complex (an unordered complex — see ordered). When x is a tna or netobject with sequence data and pathways is NULL, higher-order pathways are built automatically using the method parameter.
pathways	Character vector of pathway strings, a list of character vectors, a net_hon / net_hypa object, or any data.frame with a path column (e.g., the output of Nestimate::mogen_transitions()). If a data.frame with a path column is passed as x and pathways is NULL, it is auto-promoted to pathways and the

state set is derived from the path strings — `plot_simplicial(mgt)` works directly. String separators: "A B -> C", "A -> B -> C", "A, B, C", "A - B - C", "A B C". Last state is the target. When a `data.frame` is passed and a `count` column is present, rows are sorted by count descending before `max_pathways` is applied. When `NULL` and `x` is a model with sequence data, pathways are built automatically.

<code>method</code>	Pathway source when auto-building from a <code>tna/netobject</code> : "hon" (default, higher-order network), "hypha" (anomalous paths via hypergeometric null), or "rules" (association-rule itemsets via <code>Nestimate::association_rules</code> ; rules are rendered as single-colored blobs because itemsets are undirected).
<code>max_pathways</code>	Maximum number of pathways to display. HON pathways are ranked by count, HYPA by anomaly ratio. <code>NULL</code> shows all. Default 10.
<code>pathway_index</code>	Optional positive integer vector selecting ranked pathways after extraction and ranking, before <code>max_pathways</code> is applied. For example, 2 plots the second-ranked pathway and 2:4 plots pathways ranked second through fourth.
<code>anomaly</code>	HYPA anomaly type to display when plotting a <code>net_hypa</code> object or auto-building HYPA pathways via <code>method = "hypha"</code> . One of "all", "over", or "under". Default "all". Ignored (with a warning) for non-HYPA inputs such as <code>net_hon</code> , <code>net_association_rules</code> , <code>net_link_prediction</code> , character pathway vectors, or <code>method = "hon" / "rules"</code> , which have no anomaly concept.
<code>layout</code>	"circle" (default) or a coordinate matrix.
<code>labels</code>	Display labels. <code>NULL</code> uses state names.
<code>node_color</code>	Source node fill color.
<code>target_color</code>	Target node fill color.
<code>ring_color</code>	Donut ring color.
<code>node_size</code>	Node point size.
<code>label_size</code>	Label text size.
<code>label_color</code>	Label text color (default "#e8e8e8", very light grey). Light grey reads on both white and dark fills when the auto-contrast halo is enabled (it is by default). Applied to both source and target labels unless <code>target_label_color</code> overrides for targets.
<code>target_label_color</code>	Target-node label color. <code>NULL</code> (default) reuses <code>label_color</code> .
<code>label_halo</code>	Logical. Draw a contrasting halo behind each label so it stays readable on any fill — node disc, blob, or the white canvas. Default <code>TRUE</code> . The halo is the only reliable way to keep, e.g., white labels legible when <code>node_color</code> is also light.
<code>label_halo_color</code>	Halo color. <code>NULL</code> (default) auto-picks black or white based on the luminance of <code>label_color</code> , so a white label gets a dark halo and vice versa.
<code>label_halo_width</code>	Halo thickness in plot units. Default 0.035; raise for chunkier outlines, lower for subtler ones, or set to 0 to disable without touching <code>label_halo</code> .
<code>label_halo_alpha</code>	Halo opacity (0–1). Default 0.6 reads as a soft glow rather than a hard outline; raise toward 1 for sharper contrast on very busy backgrounds.

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	Draw soft drop shadows?
title	Plot title.
dismantled	If TRUE, one panel per pathway arranged in a grid layout.
ncol	Number of columns in the grid when dismantled = TRUE. Default NULL auto-selects based on the number of pathways.
ordered	Is each higher-order structure a PATH or a SET? TRUE treats the last state of every pathway as its target (HON / HYPA / MOGen). FALSE treats every member as co-equal: there is no target, so no node is painted with target_color, no direction cue is drawn, and the panel title is a member list rather than an arrow. NULL (default) reads it off the input — net_association_rules and simplicial_complex are sets, everything else is a path.
direction	Draw the traversal inside each per-pathway panel: a light-to-dark core ramp along the path, a ring whose gold peaks on the side facing the next state, and an arrowhead just outside each node aimed at its successor. NULL (default) enables them exactly when dismantled = TRUE. A simplex is a set of vertices, so the combined overlay — where blobs overlap and a state can sit in several pathways at once — cannot express direction; direction = TRUE with dismantled = FALSE is an error rather than a silent no-op. Also forced off when the caller has collapsed the source/target two-tone (undirected input such as net_association_rules).
direction_cues	Which cues to draw, any of "shade", "ring", "arrows". Default all three.
node_radius	Node core radius in data units, used only on the directed path (rings and cores become polygons there so the ring gradient and the arrow offset are expressible; geom_point() sizes are device millimeters and cannot answer either). NULL (default) scales it to the panel extent so the nodes keep the size they have today.
legend	Draw the in-figure legend strip beneath a dismantled grid. Default TRUE when direction is on.
...	Additional arguments passed to Nestimate::build_hon() or Nestimate::build_hypa() when auto-building.

Details

Supports direct use with tna and netobject models: when x has sequence data, HON or HYPA pathways are built automatically (requires the **Nestimate** package). Pathways can also be passed as net_hon or net_hypa objects, with labels auto-translated when x is a tna/netobject.

Value

Invisibly, a ggplot object for the combined overlay. With dismantled = TRUE the arranged grid is returned instead: a gtable when **gridExtra** is available, otherwise a plain list of the per-pathway ggplot objects. NULL is returned when there is nothing to draw (no pathways could be extracted). Called for the side effect of drawing.

Examples

```
set.seed(1)
mat <- matrix(runif(16), 4, 4,
              dimnames = list(LETTERS[1:4], LETTERS[1:4]))
diag(mat) <- 0
plot_simplicial(mat, c("A B -> C", "B C -> D"))
```

plot_temporal	<i>Temporal Network Prism (3D Glass Box)</i>
---------------	--

Description

Displays a network at different time points as vertical planes inside a 3D oblique-projection box, with time flowing left to right. Each network plane extends into the depth of the box.

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

<code>x</code>	An edge list data frame with columns <code>from</code> , <code>to</code> , and a time column, OR a <code>cograph_network</code> (reads time from stored data), OR a named list of network objects.
<code>time</code>	Character. Name of the time column.
<code>slices</code>	Integer or NULL. Number of equal-width time bins. Default NULL uses unique time values.
<code>cumulative</code>	Logical. If TRUE, edges accumulate. Default FALSE.
<code>labels</code>	Character vector of layer labels. Default <code>auto</code> .
<code>layout</code>	Character or matrix. Character values currently use a shared Fruchterman-Reingold/spring layout; a matrix supplies shared coordinates. Default <code>"spring"</code> .
<code>node_size</code>	Numeric. Node size. Default 2.5.
<code>node_color</code>	Character or vector. Node fill color. A single color applies everywhere. An unnamed vector is recycled across <i>layers</i> , coloring each plane as a whole. A named vector is matched to node names instead and colors each <i>node</i> the same on every plane, which is what makes a node identifiable as it moves through the stack; names not present in the network are an error rather than silent. See also <code>color_by</code> . The original text of this parameter continues: a single color applies to all layers, or a vector of length <code>n_layers</code> for per-layer colors. Default <code>"steelblue"</code> .
<code>color_by</code>	One of <code>"layer"</code> (the default, and the historical behavior) or <code>"node"</code> . Chooses what an unnamed <code>node_color</code> vector indexes. A named <code>node_color</code> always colors by node and ignores this argument.
<code>node_shape</code>	Integer. Point shape (pch). Default 21 (filled circle).
<code>node_border</code>	Character. Node border color. Default <code>"gray30"</code> .
<code>edge_color</code>	Character or vector. Edge color (single or per-layer). Default <code>"#E41A1C"</code> .
<code>edge_width</code>	Numeric. Base edge width. Actual width scales by weight. Default 1.5.
<code>edge_alpha</code>	Numeric. Edge transparency (0-1). Default 0.35.
<code>plane_color</code>	Character or vector. Plane fill color (single or per-layer). Default <code>"gray92"</code> .
<code>plane_alpha</code>	Numeric. Plane fill transparency (0-1). Default 0.2.
<code>plane_border</code>	Character. Plane border color. Default <code>"gray60"</code> .
<code>plane_lty</code>	Integer. Plane border line type. Default 2 (dashed).
<code>box</code>	Logical. Draw 3D bounding box. Default TRUE.
<code>box_color</code>	Character. Box edge color. Default <code>"gray40"</code> .
<code>connections</code>	Logical. Draw lines connecting same nodes across planes. Default FALSE.
<code>connection_color</code>	Character. Default <code>"gray50"</code> .

connection_alpha	Numeric. Default 0.15.
minimum	Numeric. Minimum edge weight to display. Default 0.
show_labels	Logical. 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. Default 42.
...	Additional arguments (currently unused).

Value

Invisible list of adjacency matrices per layer.

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

A drop-in replacement for qgraph::qgraph() that uses cograph's splot engine. Accepts qgraph parameter names for seamless migration from qgraph to cograph.

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: "circle", "spring", "oval", or a coordinate matrix
theme	Plot theme ("colorblind", "gray", etc.)

<code>mar</code>	Plot margins (numeric vector of length 4)
<code>cut</code>	Edge emphasis threshold
<code>edge.label.position</code>	Position of edge labels along edge (0-1)
<code>edge.label.cex</code>	Edge label size multiplier
<code>edge.color</code>	Edge colors
<code>vsize</code>	Node size
<code>pie</code>	Pie/donut fill values (e.g., initial probabilities)
<code>pieColor</code>	Pie/donut segment colors
<code>lty</code>	Line type for edges (1=solid, 2=dashed, 3=dotted)
<code>directed</code>	Logical, is the graph directed?
<code>minimum</code>	Minimum edge weight to display
<code>posCol</code>	Color for positive edges
<code>negCol</code>	Color for negative edges
<code>arrowAngle</code>	Arrow head angle in radians. Default NULL, which leaves <code>splot()</code> 's own <code>arrow_angle</code> default of $\pi/6$ (30 degrees) in place.
<code>title</code>	Plot title
<code>...</code>	Additional arguments passed to <code>splot()</code>

Value

Invisibly returns the `cograph_network` object from `splot()`.

Examples

```
# Simple usage
m <- matrix(runif(25), 5, 5)
plot_tna(m)

# With qgraph-style parameters
plot_tna(m, vsize = 15, edge.label.cex = 2, layout = "circle")

# With custom colors
plot_tna(m, color = palette_colorblind(5), vsize = 10)

m <- matrix(runif(25), 5, 5)
tplot(m)
```

plot_trajectories	<i>Plot Individual Trajectories</i>
-------------------	-------------------------------------

Description

Creates an alluvial-style diagram where each individual's trajectory is shown as a separate line. This is an alias for `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

<code>x</code>	Data frame with one column per time point and one row per individual trajectory.
<code>from_title</code>	Column titles. Default NULL, which uses the column names of <code>x</code> . Pass a character vector to override them.
<code>title</code>	Optional plot title. Applied via <code>ggplot2::labs(title = title)</code> .
<code>from_colors</code>	Colors for left-side nodes. Default uses palette.
<code>flow_color_by</code>	Color trajectory lines by state. Supports "source", "destination", "first", "last", or NULL. Default "first".
<code>node_width</code>	Width of node rectangles (0-1 scale). Default 0.08.
<code>node_border</code>	Border color for nodes. Default NA (no border).
<code>node_spacing</code>	Vertical spacing between nodes (0-1 scale). Default 0.02.
<code>label_size</code>	Size of node labels. Default 3.5.
<code>label_position</code>	Position of node labels: "beside" (default), "inside", "above", "below", "outside". Applied to first and last columns. See <code>mid_label_position</code> for middle columns.
<code>mid_label_position</code>	Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as <code>label_position</code> . Default NULL uses <code>label_position</code> value.
<code>label_halo</code>	Logical: add white halo around labels for readability? Default TRUE.
<code>label_color</code>	Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots; simple two-column aggregate plots use black external labels and white inside labels.
<code>label_fontface</code>	Font face of state name labels ("plain", "bold", "italic", "bold.italic"). Default "plain". Applied to multi-step and individual-tracking plots; simple two-column aggregate plots use fixed label font faces.
<code>label_nudge</code>	Distance between node edge and label (in plot units). Default 0.02. Used by multi-step and individual-tracking plots.
<code>title_size</code>	Size of column titles. Default 5.
<code>title_color</code>	Color of column title text. Default "black". Applied to multi-step and individual-tracking plots; simple two-column aggregate plots use black titles.
<code>title_fontface</code>	Font face of column titles. Default "bold". Applied to multi-step and individual-tracking plots.
<code>curve_strength</code>	Controls bezier curve shape (0-1). Default 0.6.

line_alpha	Alpha for individual tracking lines. Default 0.3.
line_width	Width of individual tracking lines. Default 0.5.
jitter_amount	Vertical jitter for individual lines (0-1). 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".
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 (inherits from 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). Simple two-column aggregate plots show all nonzero value labels 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 full width. Use smaller values (e.g., 0.6) to bring columns closer together.
proportional_nodes	Logical: size nodes proportionally to counts in individual-tracking plots? Default TRUE.
node_label_format	Format string for node labels with {state} and {count} placeholders in individual-tracking plots. Default NULL (plain state name). Example: "{state} (n={count})".
bundle_size	Controls line bundling for large datasets. Default NULL (no bundling). Integer >= 2: each drawn line represents that many cases. Numeric in (0,1): reduce to this fraction of original lines (e.g., 0.15 keeps about 15 percent of lines).
bundle_legend	Logical or character: show annotation when bundling is active? Default TRUE shows "Each line ~ N cases" below the plot. Pass a string to use custom text (with {n} placeholder for count).
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 ggplot2 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 elegant alluvial/Sankey diagram showing how items flow from one set of categories to another. Useful for visualizing cluster transitions, state changes, or any categorical mapping.

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 several formats: • A transition matrix (rows = from, cols = to, values = counts) • Two vectors: pass before as x and after as second argument (contingency table computed automatically, like chi-square) • A 2-column data frame (raw observations; table computed automatically) • A data frame with columns: from, to, count • A list of matrices for multi-step transitions
from_title	Title for the left column. Default "From". For multi-step, use a vector of titles (e.g., c("T1", "T2", "T3", "T4")).

to_title	Title for the right column. Default "To". Ignored for multi-step.
title	Optional plot title. Applied via <code>ggplot2::labs(title = title)</code> .
from_colors	Colors for left-side nodes. Default uses palette.
to_colors	Colors for right-side nodes. Default uses palette.
flow_fill	Fill color for flows. Default "#888888" (grey). In multi-step and individual-tracking plots, ignored when <code>flow_color_by</code> is set; simple two-column aggregate plots use <code>flow_fill</code> .
flow_alpha	Alpha transparency for flows. Default 0.4.
flow_color_by	Color flows by state. For multi-step aggregate flows, use "source" or "destination"; for individual trajectories, "first" and "last" are also supported. Default NULL uses <code>flow_fill</code> ; simple 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". Applied to first and last columns. See <code>mid_label_position</code> for middle columns.
mid_label_position	Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as <code>label_position</code> . Default NULL uses <code>label_position</code> value.
label_halo	Logical: add white halo around labels for readability? Default TRUE.
label_color	Color of state name labels. Default "black". Applied to multi-step and individual-tracking plots; simple 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; simple two-column aggregate plots use fixed label font faces.
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; simple two-column aggregate plots use black titles.
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".
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 (inherits from 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). Simple two-column aggregate plots show all nonzero value labels 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".
conserve_flow	Logical: should left and right totals match? Default TRUE. When FALSE, each side scales independently (allows for "lost" or "gained" items).
min_flow	Minimum flow value to display. Default 0 (show all).
threshold	Minimum edge weight to display. Flows below this value are removed. Combined with min_flow: effective minimum is max(threshold, min_flow). 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 full width. Use smaller values (e.g., 0.6) to bring columns closer together.
track_individuals	Logical: draw individual lines instead of aggregated flows? Default FALSE. When TRUE, each row in the data frame becomes a separate line.
line_alpha	Alpha for individual tracking lines. Default 0.3.
line_width	Width of individual tracking lines. Default 0.5.
jitter_amount	Vertical jitter for individual lines (0-1). Default 0.8.
proportional_nodes	Logical: size nodes proportionally to counts in individual-tracking plots? Default TRUE.
node_label_format	Format string for node labels with {state} and {count} placeholders in individual-tracking plots. Default NULL (plain state name). Example: "{state} (n={count})".
bundle_size	Controls line bundling for large datasets. Default NULL (no bundling). Integer >= 2: each drawn line represents that many cases. Numeric in (0,1): reduce to this fraction of original lines (e.g., 0.15 keeps about 15 percent of lines).

`bundle_legend` Logical or character: show annotation when bundling is active? Default TRUE shows "Each line ~ N cases" below the plot. Pass a string to use custom text (with {n} placeholder for count).

`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".

Details

The function creates smooth bezier curves connecting nodes from the left column to the right column. Flow width is proportional to the transition count. Nodes are sized proportionally to their total flow.

Value

A ggplot2 object.

Examples

```
# From a transition matrix
mat <- matrix(c(50, 10, 5, 15, 40, 10, 5, 20, 30), 3, 3, byrow = TRUE,
              dimnames = list(c("Light", "Resource", "Intense"),
                              c("Light", "PBL", "Resource")))
plot_transitions(mat, from_title = "Time 1", to_title = "Time 2")

# From a 2-column data frame (auto-contingency)
df <- data.frame(time1 = c("A", "A", "B", "B", "C"),
                  time2 = c("X", "Y", "X", "Z", "Y"))
plot_transitions(df)
```

```
print.cograph_communities
```

Print Community Structure

Description

Print Community Structure

Usage

```
## S3 method for class 'cograph_communities'
print(x, ...)
```

Arguments

x A cograph_communities object.
 ... Ignored.

Value

Invisibly returns the original object.

Examples

```
g <- igraph::make_graph("Zachary")
comm <- community_louvain(g)
print(comm)
```

```
print.cograph_degree_fit
```

Print method for cograph_degree_fit

Description

Displays the comparison table of fitted distributions sorted by AIC.

Usage

```
## S3 method for class 'cograph_degree_fit'
print(x, digits = 4, ...)
```

Arguments

x A cograph_degree_fit object from [fit_degree_distribution](#).
 digits Number of decimal places. Default 4.
 ... Additional arguments passed to print.data.frame.

Value

Invisible x.

Examples

```
adj <- matrix(c(0, 1, 1, 0, 0,
               1, 0, 1, 1, 0,
               1, 1, 0, 1, 1,
               0, 1, 1, 0, 1,
               0, 0, 1, 1, 0), 5, 5, byrow = TRUE)
fit <- cograph::fit_degree_distribution(adj,
  distributions = c("exponential", "poisson"))
print(fit)
```

print.cograph_network	<i>Print cograph_network Object</i>
-----------------------	-------------------------------------

Description

Print cograph_network Object

Usage

```
## S3 method for class 'cograph_network'
print(x, ...)
```

Arguments

- x A cograph_network object.
- ... Ignored.

Value

The input object x, invisibly.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- cograph(adj)
print(net)
```

project_bipartite	<i>Project Bipartite Network to One-Mode</i>
-------------------	--

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: $\text{shared} / (\text{total}_i + \text{total}_j - \text{shared})$ 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

```
# Incidence matrix: 4 students x 3 courses
inc <- matrix(c(1, 1, 0,
                1, 0, 1,
```

```

      0, 1, 1,
      1, 1, 1), 4, 3, byrow = TRUE)
rownames(inc) <- paste0("S", 1:4)
colnames(inc) <- paste0("C", 1:3)

# Student co-enrollment (weighted)
cograph::project_bipartite(inc, mode = "rows", method = "sum")

# Course overlap (Jaccard similarity)
cograph::project_bipartite(inc, mode = "columns", method = "jaccard")

# Newman's weighted projection
cograph::project_bipartite(inc, mode = "rows", method = "newman")

```

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, 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 (all nodes reach equal proportions of the graph); values close to 1 indicate strong hierarchical structure. Matches `networkx.global_reaching centrality` exactly.

Value

A single numeric value in $[0, 1]$.

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

```
# Star graph: highly hierarchical (directed out from center)
adj <- matrix(0, 5, 5)
adj[1, 2:5] <- 1
rownames(adj) <- colnames(adj) <- LETTERS[1:5]
reaching_global(adj, mode = "out")
```

register_layout

Register a Custom Layout

Description

Register a new layout algorithm that can be used for network visualization.

Usage

```
register_layout(name, layout_fn)
```

Arguments

name	Character. Name of the layout.
layout_fn	Function. A function that computes node positions. Should accept a Cograph-Network object and return a matrix with x, y columns.

Value

Invisible NULL.

Examples

```
# Register a simple random layout under a new name. Registering an existing
# name (for example "random") would replace the built-in layout for the rest
# of the session, so pick a name of your own.
register_layout("my_random", function(network, ...) {
  n <- network$n_nodes
  cbind(x = stats::runif(n), y = stats::runif(n))
})
```

register_shape	<i>Register a Custom Shape</i>
----------------	--------------------------------

Description

Register a new shape that can be used for node rendering.

Usage

```
register_shape(name, draw_fn)
```

Arguments

name	Character. Name of the shape.
draw_fn	Function. A function that draws the shape. Should accept parameters: x, y, size, fill, border_color, border_width, ...

Value

Invisible NULL.

Examples

```
# Register a custom hexagon shape under a new name. Registering an existing
# name (for example "hexagon") would replace the built-in shape for the rest
# of the session, so pick a name of your own.
register_shape("my_hexagon", function(x, y, size, fill, border_color, border_width, ...) {
  angles <- seq(0, 2 * pi, length.out = 7)
  grid::polygonGrob(
    x = x + size * cos(angles),
    y = y + size * sin(angles),
    gp = grid::gpar(fill = fill, col = border_color, lwd = border_width)
  )
})
```

register_svg_shape	<i>Register Custom SVG Shape</i>
--------------------	----------------------------------

Description

Register an SVG file or string as a custom node shape.

Usage

```
register_svg_shape(name, svg_source)
```

Arguments

- name Character: unique name for this shape (used in node_shape parameter).
- svg_source Character: path to SVG file OR inline SVG string.

Value

Invisible NULL. The shape is registered for use with sn_nodes().

Examples

```
# Register an inline SVG shape
register_svg_shape("simple_star",
  '<svg viewBox="0 0 100 100">
    <polygon points="50,5 20,99 95,39 5,39 80,99" fill="currentColor"/>
  </svg>')

# Use it in a network
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |> sn_nodes(shape = "simple_star") |> splot()
```

register_theme	<i>Register a Custom Theme</i>
----------------	--------------------------------

Description

Register a new theme for network visualization.

Usage

```
register_theme(name, theme)
```

Arguments

- name Character. Name of the theme.
- theme A CographTheme object or a list of theme parameters.

Value

Invisible NULL.

Examples

```
# Register a custom theme
register_theme("custom", list(
  background = "white",
  node_fill = "steelblue",
  node_border = "navy",
  edge_color = "gray50"
))
```

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)`: 30 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  
plot(regulation_net, tna_styling = TRUE)
```

remove_edges	<i>Remove Edges from a Network</i>
--------------	------------------------------------

Description

Remove Edges from a Network

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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

remove_edges(adj, from = "A", to = "B")
```

remove_isolates	<i>Remove Isolated Nodes</i>
-----------------	------------------------------

Description

Drops every node with no edges. Filtering edges deliberately keeps nodes (see [filter_edges](#)), so this is the explicit way to prune the isolates a filter left behind.

Usage

```
remove_isolates(x, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input: <code>cograph_network</code> , <code>matrix</code> , <code>igraph</code> , <code>network</code> , <code>tna</code> , or an edge-list data frame.
keep_format	Logical. If TRUE, <code>matrix</code> , <code>igraph</code> , <code>statnet network</code> and <code>tna</code> inputs are returned in that format. Default FALSE returns a <code>cograph_network</code> .
directed	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with the isolated nodes removed (or the input format when `keep_format = TRUE`). Node order is otherwise preserved and edge indices are remapped to the new node numbering.

See Also

[filter_edges](#), [split_components](#), [filter_nodes](#)

Examples

```
adj <- matrix(0, 4, 4, dimnames = list(LETTERS[1:4], LETTERS[1:4]))
adj["A", "B"] <- adj["B", "A"] <- 1

# C and D have no edges
remove_isolates(adj)
```

remove_nodes	<i>Remove Nodes from a Network</i>
--------------	------------------------------------

Description

Remove Nodes from a Network

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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

remove_nodes(adj, nodes = "B")
```

rename_nodes	<i>Rename Nodes</i>
--------------	---------------------

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. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

Value

A `cograph_network` with the renamed nodes, or the input format when `keep_format = TRUE`. Labels not named in `from` are left alone.

See Also

[reorder_nodes](#), [set_nodes](#)

Examples

```
adj <- matrix(c(0, 1, 1, 0), 2, 2)
rownames(adj) <- colnames(adj) <- c("A", "B")

get_labels(rename_nodes(adj, from = "A", to = "Alpha"))
get_labels(rename_nodes(adj, from = c(A = "Alpha", B = "Beta")))
```

render-ggplot

ggplot2 Conversion

Description

Convert Cograph network to `ggplot2` object.

Value

A `ggplot2` object representing the network.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
p <- sn_ggplot(adj)
```

render-grid	<i>Grid Rendering</i>
-------------	-----------------------

Description

Main grid-based rendering functions.

Value

See individual functions: [soplot](#) returns a `cograph_network` object invisibly; [sn_ggplot](#) returns a `ggplot2` object.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
soplot(adj)
```

reorder_nodes	<i>Reorder the Nodes of a Network</i>
---------------	---------------------------------------

Description

Changes the order the nodes are stored in, which is the order plotting functions lay them out in. The network itself is 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", "strength" to sort by. Sorting by a measure is descending.
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

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

```
adj <- matrix(c(0, 1, 1, 1,
               1, 0, 1, 0,
               1, 1, 0, 0,
               1, 0, 0, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

get_labels(reorder_nodes(adj, order = "degree"))
get_labels(reorder_nodes(adj, order = c("D", "C", "B", "A")))
```

reverse_edges	<i>Reverse Edge Direction</i>
---------------	-------------------------------

Description

Transposes the weight matrix, so every arc runs the other way. TNA users reach for this to look at where transitions came from rather than where they went.

Usage

```
reverse_edges(x, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

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

```
adj <- matrix(c(0, .5, 0,
               0, 0, .7,
               0, 0, 0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

reverse_edges(adj)
```

rich_club

*Rich Club Coefficient***Description**

Computes the rich club curve across all prominence thresholds, measuring whether prominent nodes preferentially direct their strongest ties toward each other. Supports both unweighted (Colizza et al. 2006) and weighted (Opsahl et al. 2008) formulations.

Usage

```
rich_club(
  x,
  rich = c("k", "s"),
  weighted = TRUE,
  normalized = TRUE,
  n_random = 100,
  directed = NULL,
  seed = NULL,
  digits = NULL,
  ...
)
```

Arguments

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>rich</code>	Character. Prominence definition: "k" (degree, default) or "s" (strength / weighted degree).
<code>weighted</code>	Logical. If TRUE (default), compute the weighted rich club coefficient. If FALSE, compute the unweighted version (density among rich nodes).
<code>normalized</code>	Logical. If TRUE (default), normalize against degree-preserving random graphs and include confidence intervals. The null graphs are drawn with <code>igraph::sample_degseq()</code> (which fixes the degree sequence); for a weighted rich club the observed edge weights are additionally reshuffled across the null edges, following Opsahl et al. (2008).
<code>n_random</code>	Integer. Number of random graphs for normalization. Default 100.
<code>directed</code>	Logical or NULL. Default NULL (auto-detect).
<code>seed</code>	Integer or NULL. Random seed for reproducibility. Default NULL.
<code>digits</code>	Integer or NULL. Round numeric output. Default NULL.
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

Details

Unweighted: $\phi(k) = 2E_{>k} / (N_{>k}(N_{>k} - 1))$

Weighted: $\phi^w(k) = W_{>k} / \sum_{l=1}^{E_{>k}} w_l^{ranked}$

Normalization: $\phi_{norm} = \phi_{obs} / \bar{\phi}_{rand}$. A value > 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: the observed coefficient divided by the null mean, the null mean itself, and the 2.5\ 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.

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

```
g <- igraph::sample_pa(50, m = 2, directed = FALSE)
rc <- cograph::rich_club(g, n_random = 20)
plot(rc)
```

rich_club_local	<i>Local Rich Club Score</i>
-----------------	------------------------------

Description

For each node, measures whether it preferentially directs its strongest ties toward prominent nodes. A score > 1 means the node’s ties to prominent nodes are stronger than average.

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	Integer or logical vector indicating which nodes are prominent (1/TRUE = prominent), OR a numeric threshold. If NULL, nodes above median degree (or strength) 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 (descending). Default "score".
...	Currently unused; directed is already an explicit argument above and to_igraph accepts no others.

Details

For each node i: $r_i = \bar{w}_{i \rightarrow rich} / \bar{w}_i$

Value

A plain data frame with one row per node and columns node (node label) and score, sorted by sort_by descending ("score" by default; pass sort_by = NULL to keep node order). Values > 1 indicate the node directs disproportionately strong ties to prominent nodes; a node with no neighbors, no prominent neighbor, or zero mean tie weight 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

```
adj <- matrix(c(0,5,3,1, 5,0,4,2, 3,4,0,1, 1,2,1,0), 4, 4)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
cograph::rich_club_local(adj, prominence = c(1, 1, 0, 0))
```

robustness

Network Robustness Analysis

Description

Performs a targeted attack or random failure analysis on a network, calculating the size of the largest connected component after sequential vertex or edge removal.

In a targeted attack, vertices are sorted by degree or betweenness centrality (or edges by betweenness), and successively removed from highest to lowest. In a random failure analysis, vertices/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; sort by "betweenness", "degree", or "random". Default: "betweenness"

<code>strategy</code>	Character string; "sequential" (default) recalculates centrality after each removal. "static" computes centrality once on the original network and removes nodes in that fixed order (brainGraph-style). Only affects targeted attacks; random removal is unaffected.
<code>n_iter</code>	Integer; number of iterations for random analysis. Default: 1000 (matching brainGraph convention)
<code>mode</code>	For directed networks: "all", "in", or "out". Default "all".
<code>seed</code>	Random seed for reproducibility. Default NULL.
<code>...</code>	Passed to <code>to_igraph</code> , whose only other argument is directed; anything else raises an "unused argument" error.

Details

Three attack strategies are available:

Targeted Attack - Betweenness (default): Vertices/edges are sorted by betweenness centrality and removed from highest to lowest. This targets nodes that bridge different network regions.

Targeted Attack - Degree: Vertices are sorted by degree and removed from highest to lowest. This targets highly connected hub nodes. Note: for edge attacks, degree is not available; use betweenness instead.

Random Failure: Vertices/edges are removed in random order, averaged over `n_iter` iterations. This simulates random component failures.

Strategy: The `strategy` parameter controls how targeted attacks work:

- "sequential" (default): Recalculates centrality after each removal. This is a stronger attack because removing a hub changes which nodes become the new bridges/hubs.
- "static": Computes centrality once on the original network and removes nodes in that fixed order (as in brainGraph). This matches the original Albert et al. (2000) method.

Scale-free networks are typically robust to random failures but vulnerable to targeted attacks, while 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 human-readable label for the analysis, one of "Targeted vertex attack", "Targeted edge attack", "Random vertex removal" or "Random edge removal" - not the bare type argument

The original number of vertices/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](https://doi.org/10.1038/35019019)

See Also

[plot_robustness](#), [robustness_auc](#)

Examples

```
# Create a scale-free network
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_pa(50, m = 2, directed = FALSE)

  # Targeted attack by betweenness
  rob_btw <- robustness(g, measure = "betweenness")

  # Targeted attack by degree
  rob_deg <- robustness(g, measure = "degree")

  # Random failure
  rob_rnd <- robustness(g, measure = "random", n_iter = 50)

  # View results
  head(rob_btw)
}
```

robustness_auc	<i>Calculate Area Under Robustness Curve (AUC)</i>
----------------	--

Description

Computes the area under the robustness curve using trapezoidal integration. Higher AUC indicates a more robust network. Maximum AUC is 1.0.

Usage

```
robustness_auc(x)
```

Arguments

x A robustness result from [robustness](#).

Value

Numeric AUC value between 0 and 1.

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_pa(30, m = 2, directed = FALSE)

  rob_btw <- robustness(g, measure = "betweenness")
  rob_rnd <- robustness(g, measure = "random", n_iter = 20)

  cat("Betweenness attack AUC:", round(robustness_auc(rob_btw), 3), "\n")
  cat("Random failure AUC:", round(robustness_auc(rob_rnd), 3), "\n")
}
```

robustness_summary	<i>Summary of Robustness Analysis</i>
--------------------	---------------------------------------

Description

Provides a summary comparing robustness metrics across attack strategies.

Usage

```
robustness_summary(..., x = NULL, measures = NULL, n_iter = 1000)
```

Arguments

- ... Robustness results to summarize.
- x Network for on-the-fly computation.
- measures Measures to compute if x provided.
- n_iter Iterations for random. 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.

Examples

```
g <- igraph::sample_pa(30, m = 2, directed = FALSE)
robustness_summary(x = g, measures = c("degree", "random"), n_iter = 10)
```

select_bridges	<i>Select Bridge Edges</i>
----------------	----------------------------

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

```
# Create network with bridge  
adj <- matrix(0, 5, 5)  
adj[1, 2] <- adj[2, 1] <- 1  
adj[2, 3] <- adj[3, 2] <- 1 # Bridge  
adj[3, 4] <- adj[4, 3] <- 1  
adj[4, 5] <- adj[5, 4] <- 1  
adj[3, 5] <- adj[5, 3] <- 1  
rownames(adj) <- colnames(adj) <- LETTERS[1:5]  
  
select_bridges(adj)
```

select_component	<i>Select Connected Component</i>
------------------	-----------------------------------

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

```
# Create disconnected network
adj <- matrix(0, 6, 6)
adj[1, 2] <- adj[2, 1] <- 1
adj[1, 3] <- adj[3, 1] <- 1
adj[4, 5] <- adj[5, 4] <- 1
adj[5, 6] <- adj[6, 5] <- 1
```



```
adj[4, 6] <- adj[6, 4] <- 1
rownames(adj) <- colnames(adj) <- LETTERS[1:6]

# Largest component
select_component(adj, which = "largest")

# Component containing node "A"
select_component(adj, which = "A")
```

select_edges

Select Edges with Lazy Computation

Description

A powerful edge selection function with lazy computation (only computes metrics actually referenced), multiple selection modes, and structural awareness (bridges, communities, 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: <code>between = list(c("A", "B"), c("C", "D"))</code> .
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 <code>same_community</code> 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 <code>igraph::delete_edges()</code> and <code>tidygraph::filtering_edges</code> does not remove nodes. Set FALSE to drop them, or call remove_isolates() afterwards.
keep_format	Logical. If TRUE, matrix, igraph, and statnet network inputs are returned in that format. Default FALSE returns <code>cograph_network</code> .
directed	Logical or NULL. If NULL (default), auto-detect.
.keep_isolates	Deprecated. Use <code>keep_isolates</code> .

Details

Selection modes are combined with AND logic:

- `select_edges(x, top = 10, involving = "A")` selects top 10 edges **among those involving node A**
- All criteria must be satisfied for an edge to be selected

Edge metrics are computed lazily - only those actually referenced in expressions or required by selection modes are computed.

Value

A `cograph_network` object with selected edges. If `keep_format = TRUE`, matrix, igraph, and statnet network 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

```
adj <- matrix(c(0, .5, .8, 0, .5, 0, .3, .6,
               .8, .3, 0, .4, 0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

select_edges(adj, weight > 0.5)
select_edges(adj, top = 3)
select_edges(adj, involving = "A")
select_edges(adj, between = list(c("A", "B"), c("C", "D")))
```

select_edges_between	<i>Select Edges Between Node Sets</i>
----------------------	---------------------------------------

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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Edges between {A, B} and {C, D}
select_edges_between(adj, set1 = c("A", "B"), set2 = c("C", "D"))
```

```
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

<code>x</code>	Network input.
<code>nodes</code>	Character or integer. Node names or indices.
<code>...</code>	Additional filter expressions.
<code>keep_isolates</code>	Keep nodes that end up with no edges? Default TRUE.
<code>keep_format</code>	Keep input format? Default FALSE.
<code>directed</code>	Auto-detect if NULL.

Value

A `cograph_network` with edges involving the specified nodes.

See Also

[select_edges](#), [select_edges_between](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Edges involving A
select_edges_involving(adj, nodes = "A")

# Edges involving A or B
select_edges_involving(adj, nodes = c("A", "B"))
```

select_k_core	<i>Select the k-Core of a Network</i>
---------------	---------------------------------------

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.

Usage

```
select_k_core(x, k, keep_format = FALSE, directed = NULL)
```

Arguments

x	Network input.
k	Integer. The core number.
keep_format	Logical. Return the input format when TRUE.
directed	Logical or NULL. If NULL (default), auto-detect.

Value

A cograph_network holding the k-core, or the input format when keep_format = TRUE. An empty network when no node reaches coreness k.

References

Seidman, S. B. (1983). Network structure and minimum degree. *Social Networks*, 5(3), 269–287.

See Also

[select_nodes](#), [centrality](#)

Examples

```
adj <- matrix(c(0, 1, 1, 1,
               1, 0, 1, 0,
               1, 1, 0, 0,
               1, 0, 0, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

select_k_core(adj, 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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Direct neighbors of A
select_neighbors(adj, of = "A")

# Neighbors up to 2 hops
select_neighbors(adj, of = "A", order = 2)
```

select_nodes

*Select Nodes with Lazy Centrality Computation***Description**

A more nuanced node selection function that improves upon `filter_nodes()` with lazy centrality computation (only computes measures actually referenced), multiple selection modes, and global context variables for structural awareness.

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

<code>x</code>	Network input: <code>cograph_network</code> , <code>matrix</code> , <code>igraph</code> , <code>network</code> , or <code>tna</code> object.
<code>...</code>	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 in the nodes dataframe: id, label, name, x, y, inits, color, plus any custom Centrality measures degree, indegree, outdegree, strength, instrength, outstrength, betweenness, closeness, eigenvector, pagerank, hub, authority, coreness. Any other measure <code>centrality()</code> computes can be named too; see <code>list_centralities()</code>. 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, and statnet network inputs are returned in that format. Default FALSE returns <code>cograph_network</code> .
directed	Logical or NULL. If NULL (default), auto-detect.
.keep_edges	Deprecated. Use <code>keep_edges</code> .

Details

Selection modes are combined with AND logic (like `tidygraph/dplyr`):

- `select_nodes(x, top = 10, component = "largest")` selects top 10 nodes **within** the largest component
- All criteria must be satisfied for a node to be selected

Centrality measures are computed lazily - only measures actually referenced in expressions or the `by` parameter are computed. This makes `select_nodes()` faster than `filter_nodes()` for large networks.

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`, and `statnet` network inputs are converted back to that type.

See Also

[filter_nodes](#), [select_neighbors](#), [select_component](#), [select_top](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0, .5, 0, .3, .6,
               .8, .3, 0, .4, 0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

select_nodes(adj, degree >= 3)
select_nodes(adj, top = 2, by = "pagerank")
select_nodes(adj, neighbors_of = "A", order = 2)
select_nodes(adj, component = "largest")
```

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 to apply.
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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Top 2 by degree
select_top(adj, n = 2)

# Top 2 by PageRank
select_top(adj, n = 2, by = "pagerank")
```

select_top_edges	<i>Select Top N Edges</i>
------------------	---------------------------

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.
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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Top 3 edges by weight
select_top_edges(adj, n = 3)

# Top 2 by edge betweenness
select_top_edges(adj, n = 2, by = "edge_betweenness")
```

set_edges	<i>Set Edges in Cograph Network</i>
-----------	-------------------------------------

Description

Replaces the edges in a cograph_network object. Expects a data frame with from, to, and optionally weight columns.

Usage

```
set_edges(x, edges_df)
```

Arguments

`x` A `cograph_network` object.

`edges_df` A data frame with columns: from, to, and optionally weight.

Value

The modified `cograph_network` object.

See Also

[as_cograph](#), [get_edges](#), [set_nodes](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
new_edges <- data.frame(from = c(1, 2), to = c(2, 3), weight = c(0.5, 0.8))
net <- set_edges(net, new_edges)
get_edges(net)
```

set_groups

Set Node Groups

Description

Assigns node groupings to a `cograph_network` object. Groups are stored as metadata with a type column ("layer", "cluster", or "group") for use by specialized plot functions.

Usage

```
set_groups(
  x,
  groups = NULL,
  type = c("group", "cluster", "layer"),
  nodes = NULL,
  layers = NULL,
  clusters = NULL
)
```

Arguments

`x` A `cograph_network` object.

`groups` Node groupings in one of these formats:

- Character string: Community detection method ("louvain", "walktrap", "fast_greedy", "label_prop", "infomap", "leiden")
- Named list: Group name -> node vector mapping (e.g., `list(A = c("N1", "N2"), B = c("N3", "N4"))`)

	• Unnamed vector: Group assignment per node (same order as nodes) • Data frame: Must have "node"/"nodes" column plus one of "layer"/"layers", "cluster"/"clusters", or "group"/"groups" (plural forms are automatically normalized to singular) • NULL: Use nodes + one of layers/clusters vectors
type	Group type. One of "group" (default), "cluster", or "layer". Ignored when using layers or clusters vector arguments since the type is inferred from which argument is provided.
nodes	Character vector of node labels. Use with layers, clusters, to specify groupings via vectors instead of a data frame.
layers	Character/factor vector of layer assignments (same length as nodes).
clusters	Character/factor vector of cluster assignments (same length as nodes).

Value

The modified `cograph_network` object with `node_groups` set.

See Also

[get_groups](#), [splot](#), [detect_communities](#)

Examples

```
set.seed(1)
mat <- matrix(runif(100), 10, 10)
mat <- (mat + t(mat)) / 2; diag(mat) <- 0
rownames(mat) <- colnames(mat) <- paste0("N", 1:10)
net <- as_cograph(mat)

# Named list -> layers
net <- set_groups(net, list(
  Macro = paste0("N", 1:3),
  Meso  = paste0("N", 4:7),
  Micro = paste0("N", 8:10)
), type = "layer")
get_groups(net)
```

set_layout

Set Layout in Cograph Network

Description

Sets the layout coordinates in a `cograph_network` object. Updates the x and y columns in the nodes data frame.

Usage

```
set_layout(x, layout_df)
```

Arguments

`x` A `cograph_network` object.

`layout_df` A data frame with `x` and `y` columns, or a matrix with 2 columns.

Value

The modified `cograph_network` object.

See Also

[as_cograph](#), [get_nodes](#), [sn_layout](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
layout <- data.frame(x = c(0, 1, 0.5), y = c(0, 0, 1))
net <- set_layout(net, layout)
get_nodes(net)
```

set_nodes

Set Nodes in Cograph Network

Description

Replaces the nodes data frame in a `cograph_network` object.

Usage

```
set_nodes(x, nodes_df)
```

Arguments

`x` A `cograph_network` object.

`nodes_df` A data frame with node information (`id`, `label` columns expected).

Value

The modified `cograph_network` object.

See Also

[as_cograph](#), [get_nodes](#), [set_edges](#)

Examples

```
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)
new_nodes <- data.frame(id = 1:3, label = c("A", "B", "C"))
net <- set_nodes(net, new_nodes)
get_labels(net)
```

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object
<code>from</code>	Character or numeric node identifier(s) for the source. If NULL (default), compute distances from all nodes.
<code>to</code>	Character or numeric node identifier(s) for the target. If NULL (default), compute distances to all nodes.
<code>weights</code>	Edge weight handling: NULL (default) auto-detects from edge attributes, NA forces unweighted distances, or a numeric vector of custom weights.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

Details

Uses `igraph::distances()` internally. For weighted networks, edge weights are used as distances by default. Pass `weights = NA` to ignore weights and treat all edges as having unit distance.

Note: `igraph::distances()` with `weights = NULL` automatically uses edge weight attributes if present. To force unweighted computation, pass `weights = NA` explicitly.

`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

```
# All-pairs distances
adj <- matrix(c(
  0, 1, 0, 0,
  1, 0, 1, 0,
  0, 1, 0, 1,
  0, 0, 1, 0
), 4, 4)
rownames(adj) <- colnames(adj) <- LETTERS[1:4]
cograph::shortest_paths(adj)

# Single source to all
cograph::shortest_paths(adj, from = "A")

# Point-to-point
cograph::shortest_paths(adj, from = "A", to = "D")
```

simmelian_strength	<i>Simmelian Strength (Triangle Count per Edge)</i>
--------------------	---

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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>top</code>	Integer or NULL. Return only the top N edges. Default NULL.
<code>directed</code>	Logical or NULL. Default NULL (auto-detect).
<code>digits</code>	Integer or NULL. Round numeric columns. Default NULL.
<code>...</code>	Additional arguments passed to edge centrality .

Value

A data frame sorted by triangles (descending) with columns: from, to, weight (if weighted), triangles.

See Also

[edge centrality](#), [neighborhood overlap](#)

Examples

```
k4 <- matrix(1, 4, 4); diag(k4) <- 0
rownames(k4) <- colnames(k4) <- c("A", "B", "C", "D")
cograph::simmelian_strength(k4)
```

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

<code>x</code>	Network input (matrix, cograph_network, igraph, tna object).
<code>remove_loops</code>	Logical. Remove self-loops (diagonal entries)?
<code>remove_multiple</code>	Logical. Merge duplicate edges? No-op for matrix/tna inputs (see Details).
<code>edge_attr_comb</code>	How to combine weights of duplicate edges: "sum", "mean", "max", "min", "first", or a custom function. Ignored for matrix/tna inputs.
<code>...</code>	Additional arguments (currently unused).

Details

The extent of simplification depends on the input representation:

- `matrix` and `tna`: edges are stored as an $n \times n$ weight matrix. Each cell (i, j) is unique by construction, so duplicate-edge merging is a no-op regardless of `remove_multiple` / `edge_attr_comb`;

only self-loops (the diagonal) can be removed. Convert to `cograph_network` or `igraph` first if you need true duplicate aggregation.

- `cograph_network`: duplicate edges in the edge-list are merged via `aggregate_duplicate_edges()` using `edge_attr_comb`.
- `igraph`: delegates to `igraph::simplify()`.

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.
# Matrix with self-loops
mat <- matrix(c(0.5, 0.3, 0, 0.3, 0.2, 0.4, 0, 0.4, 0.1), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
cograph::simplify(mat)

# Edge list with duplicates
edges <- data.frame(from = c(1, 1, 2), to = c(2, 2, 3), weight = c(0.3, 0.7, 0.5))
net <- cograph(edges, layout = NULL)
cograph::simplify(net)
cograph::simplify(net, edge_attr_comb = "sum")
```

sn_edges

Set Edge Aesthetics

Description

Customize the visual appearance of edges in a network plot.

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 = 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. Positive values are passed to renderers; in splot(), edges below the cutoff are faded while width scaling remains continuous.
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 = 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:

- **Single value:** Applied to all edges
- **Vector:** Per-edge values (must match edge count)
- **"weight":** Special value for width and color that auto-maps from edge weights

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 with absolute weight values, respecting the maximum parameter if set.

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:

Visualize uncertainty by drawing a wider, semi-transparent edge behind:

- ci: Vector of CI widths (0-1 scale)
- ci_scale: Width multiplier (default 2)
- ci_alpha: Transparency (default 0.15)

Value

Modified `cograph_network` object that can be piped to further customization functions or plotting functions.

See Also

[sn_nodes](#) for node customization, [cograph](#) for network creation, [splot](#) and [soplot](#) for plotting, [sn_layout](#) for layout algorithms, [sn_theme](#) for visual themes

Examples

```
adj <- matrix(c(0, 1, -0.5, 1, 0, 1, -0.5, 1, 0), nrow = 3)
cograph(adj) |>
  sn_edges(width = "weight", color = "weight") |>
  splot()

# Custom positive/negative colors with labels
cograph(adj) |>
  sn_edges(color = "weight",
            edge_positive_color = "darkblue",
            edge_negative_color = "darkred",
            labels = TRUE) |>
  splot()
```

sn_ggplot

Convert Network to ggplot2

Description

Convert a Cograph network visualization to a ggplot2 object for further customization and composability.

Usage

```
sn_ggplot(network, title = NULL)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
title	Optional plot title.

Value

A ggplot2 object.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
# With cograph()
p <- cograph(adj) |> sn_ggplot()
print(p)

# Direct matrix input
p <- adj |> sn_ggplot()

# Further customization
p + ggplot2::labs(title = "My Network")
```


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 style). Good general-purpose layout. Default.

oval/ellipse Nodes arranged around an ellipse.

circle Nodes arranged in a circle. Good for small networks or when structure is less important.

groups Circular layout with grouped nodes clustered together.

grid Nodes in a regular grid.

random Random positions. Useful as starting point.

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.

You can also pass igraph layout functions directly or use full names like "layout_with_kk".

Value

Modified cograph_network object.

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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |> sn_layout("circle") |> splot()

# Custom coordinates
coords <- matrix(c(0, 0, 1, 0, 0.5, 1), ncol = 2, byrow = TRUE)
cograph(adj) |> sn_layout(coords) |> splot()
```

sn_nodes

Set Node Aesthetics

Description

Customize the visual appearance of nodes in a network plot.

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 <code>cograph_network</code> 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. Options: "circle", "square", "triangle", "diamond", "pentagon", "hexagon", "ellipse", "heart", "star", "pie", "donut", "cross", "rectangle", or any custom SVG shape registered with <code>register_svg_shape()</code> .
node_svg	Custom SVG for node shape: path to SVG file OR inline SVG string. Overrides shape parameter when provided.
svg_preserve_aspect	Logical: maintain SVG aspect ratio? Default TRUE.
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 transparency (0-1).
label_size	Label text size.
label_color	Label text color.
label_position	Label position: "center", "above", "below", "left", "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. 0.1 = 10% filled, 0.5 = 50% filled, 1.0 = fully filled ring. Can be a single value (all nodes) or vector (per-node values).
donut_values	Deprecated. Use donut_fill for simple fill proportion. Still works for backwards compatibility.
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). Default: "maroon" fill, "gray90" background when shape="donut".
donut_colors	Deprecated. Use donut_color instead.
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: custom format function (overrides digits).
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:

All aesthetic parameters can be specified as:

- **Single value:** Applied to all nodes (e.g., `fill = "blue"`)
- **Vector:** Per-node values, recycled if shorter than node count
- **Column name:** String referencing a column in the node data frame

Parameters are validated for correct length; providing a vector with length other than 1 or `n_nodes` will produce a warning about recycling.

Donut Charts:

Donut charts are ideal for showing a single proportion (0-1) per node:

- Set `donut_fill` to a numeric value or vector (0 = empty, 1 = full)
- Use `donut_color` to set fill color(s)
- Use `donut_shape` for non-circular donuts ("square", "hexagon", etc.)
- Enable `donut_show_value = TRUE` to display the value in the center

Value

Modified `cograph_network` object that can be piped to further customization functions 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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |>
  sn_nodes(size = 0.08, fill = "steelblue", shape = "circle") |>
  splot()

# Per-node customization: vectors of length n
cograph(adj) |>
  sn_nodes(size = c(0.08, 0.06, 0.1),
            fill = c("#E41A1C", "#377EB8", "#4DAF4A"),
            shape = c("circle", "square", "triangle")) |>
  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 or function.
target	What to apply the palette to: "nodes", "edges", or "both".
by	Variable to map colors to (for nodes: column name or "group").

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.

You can also pass a custom palette function that takes `n` and returns `n` colors.

Value

Modified cograph_network object.

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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |> sn_palette("viridis") |> splot()

# Apply to edges
cograph(adj) |> sn_palette("colorblind", target = "edges") |> 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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- cograph(adj)
sn_save(net, 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 <code>cograph_network</code> object.
filename	Output filename. Format is detected from the extension by <code>ggplot2::ggsave()</code> .
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 <code>ggsave</code> .

Value

The output filename, invisibly.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)  
net <- cograph(adj)  
sn_save_ggplot(net, file.path(tempdir(), "network.pdf"))
```

sn_theme	<i>Apply Theme to Network</i>
----------	-------------------------------

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.
...	Additional theme parameters to override.

Details**Available Themes:**

classic Default theme with white background, blue nodes, gray edges.

dark Dark background with light nodes. Good 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

Modified cograph_network object.

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

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj) |> sn_theme("dark") |> splot()

# Override a theme property
cograph(adj) |> sn_theme("classic", background = "lightgray") |> splot()
```

soplot

Plot Cograph Network

Description

Main plotting function for Cograph networks. Renders the network visualization using grid graphics. Accepts all node and edge aesthetic parameters.

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 <code>cograph_network</code> object, matrix, data.frame, or <code>igraph</code> object. Matrices and other inputs are auto-converted.
title	Optional plot title.
title_size	Title font size.
margins	Plot margins as <code>c(bottom, left, top, right)</code> .
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". <code>igraph</code> (2-letter): "kk" (Kamada-Kawai), "fr" (Fruchterman-Reingold), "drl", "mds", "ni" (nicely), "tr" (tree), etc. Can also pass a coordinate matrix or <code>igraph</code> layout function directly.
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 $\text{abs}(\text{weight}) < \text{threshold}$ 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". Default inherits from node_shape.
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	Base edge size for weight scaling. NULL (default) uses adaptive sizing based on network size: $15 * \exp(-n_nodes/90) + 1$. Larger values = thicker edges.
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	Two-tier cutoff for edge width scaling. NULL (default) = auto 75th percentile. 0 = disabled. Positive number = manual threshold.
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".
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.

<code>edge_negative_color</code>	Color for negative edge weights.
<code>negative_color</code>	Deprecated. Use <code>edge_negative_color</code> instead.
<code>edge_duplicates</code>	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.
<code>edge_labels</code>	Edge labels. Can be TRUE to show weights, or a vector.
<code>edge_label_size</code>	Edge label text size.
<code>edge_label_color</code>	Edge label text color.
<code>edge_label_position</code>	Position along edge (0 = source, 0.5 = middle, 1 = target).
<code>edge_label_offset</code>	Perpendicular offset from edge line.
<code>edge_label_bg</code>	Background color for edge labels (default "white"). Set to NA for transparent.
<code>edge_label_fontface</code>	Font face: "plain", "bold", "italic", "bold.italic".
<code>edge_label_border</code>	Border style: NULL, "rect", "rounded", "circle".
<code>edge_label_border_color</code>	Border color for label border.
<code>edge_label_underline</code>	Logical. Underline the label text?
<code>bidirectional</code>	Logical. Show arrows at both ends of edges?
<code>loop_rotation</code>	Angle in radians for self-loop direction (default: $\pi/2$ = top).
<code>curve_shape</code>	Spline tension for curved edges (-1 to 1, default: 0).
<code>curve_pivot</code>	Pivot position along edge for curve control point (0-1, default: 0.5).
<code>curves</code>	Curve mode: TRUE (default) = single edges straight, reciprocal edges curve as ellipse (two opposing curves); FALSE = all straight; "force" = all curved.
<code>node_names</code>	Alternative names for legend (separate from display labels).
<code>legend</code>	Logical. Show legend?
<code>legend_position</code>	Legend position: "topright", "topleft", "bottomright", "bottomleft".
<code>scaling</code>	Scaling mode: "default" for qgraph-matched scaling where <code>node_size=6</code> looks similar to <code>qgraph vsize=6</code> , or "legacy" to preserve pre-v2.0 behavior.
<code>weight_digits</code>	Number of decimal places to round edge weights to before plotting. Edges that round to zero are automatically removed. Default 2. Set NULL to disable rounding.

Details

soplot vs splot:

soplot() uses grid graphics while splot() uses base R graphics. Both accept the same parameters and produce visually similar output. Choose based on:

- **soplot**: Better for integration with ggplot2, combining plots, and publication-quality vector graphics.
- **splot**: Better for large networks (faster rendering), interactive exploration, and traditional R workflows.

Edge Curve Behavior:

Edge curving is controlled by the curves and curvature parameters:

curves = FALSE All edges are straight lines.

curves = TRUE (Default) Reciprocal edge pairs (A->B and B->A) curve in opposite directions to form a visual ellipse. Single edges remain straight.

curves = "force" All edges curve inward toward the network center.

Weight Scaling Modes (edge_scale_mode):

Controls how edge weights map to visual widths:

linear Width proportional to weight. Best for similar-magnitude weights.

log Logarithmic scaling. Best for weights spanning orders of magnitude.

sqrt Square root scaling. Moderate compression for skewed data.

rank Rank-based scaling. Equal visual spacing regardless of values.

Donut Visualization:

The donut system visualizes proportions (0-1) as filled rings around nodes:

donut_fill Proportion filled (0-1). Can be scalar or per-node vector.

donut_color Fill color. Single color, c(fill, bg), or per-node vector.

donut_shape Base shape: "circle", "square", "hexagon", etc.

donut_show_value Show numeric value in center.

Value

The updated cograph_network object, invisibly. Called primarily for the side effect of drawing.

The updated cograph_network object, invisibly. Called primarily for the side effect of drawing.

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

```

adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
# With cograph()
cograph(adj) |> soplplot()

# Direct matrix input with all options
adj |> soplplot(
  layout = "circle",
  node_fill = "steelblue",
  node_size = 0.08,
  edge_width = 2
)
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
sn_render(mat)

```

spanning_tree

*Minimum or Maximum Spanning Tree***Description**

Prim's algorithm on each connected component, so a disconnected network yields a spanning forest.

Usage

```

spanning_tree(
  x,
  weights = c("weight", "none"),
  maximum = FALSE,
  keep_format = FALSE,
  directed = NULL
)

```

Arguments

<code>x</code>	Network input.
<code>weights</code>	"weight" (default) uses the edge weights as costs; "none" treats every edge as cost 1.
<code>maximum</code>	Logical. Find the maximum spanning tree instead of the minimum. Default FALSE. Set TRUE when the weights are similarities.
<code>keep_format</code>	Logical. Return the input format when TRUE.
<code>directed</code>	Logical or NULL. Directedness to read the input with; 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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

spanning_tree(adj)
spanning_tree(adj, maximum = TRUE)
```

split_components	<i>Split a Network into Its Connected Components</i>
------------------	--

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. Drop components smaller than this. Default 1 (keep all, including isolated nodes).
keep_format	Logical. Return each component in the input format.
directed	Logical or NULL. If NULL (default), auto-detect.

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")`.

See Also

[select_component](#), [remove_isolates](#)

Examples

```
adj <- matrix(0, 5, 5, dimnames = list(LETTERS[1:5], LETTERS[1:5]))
adj["A", "B"] <- adj["B", "A"] <- 1
adj["C", "D"] <- adj["D", "C"] <- 1

parts <- split_components(adj)
length(parts)
```

```
splot.group_tna_permutation
```

Plot Group Permutation Test Results

Description

Visualizes all pairwise permutation test results from a `group_tna` object. Creates a multi-panel plot with one panel per comparison.

Usage

```
splot.group_tna_permutation(x, ...)

plot_group_permutation(x, i = NULL, combined = TRUE, ...)
```

Arguments

<code>x</code>	A <code>group_tna_permutation</code> object (from <code>tna::permutation_test</code> on <code>group_tna</code>).
<code>...</code>	Additional arguments passed to <code>plot_permutation()</code> .
<code>i</code>	Index or name of specific comparison to plot. <code>NULL</code> for all.
<code>combined</code>	Logical: when <code>TRUE</code> (default), lay out panels in an internal grid via <code>graphics::par(mfrow=...)</code> . Set to <code>FALSE</code> to draw each panel into a layout the caller has already configured (e.g. via <code>panel_layout()</code>). Ignored when <code>i</code> selects a single panel.

Value

When `i` is supplied, invisibly returns the `plot_permutation()` result for the selected panel (a `cograph_network`). Otherwise invisibly returns `NULL` after drawing all panels.

Examples

```
# Mock a group_tna_permutation object
d1 <- matrix(c(0, .2, -.1, -.2, 0, .1, .1, -.1, 0), 3, 3)
rownames(d1) <- colnames(d1) <- c("A", "B", "C")
d1_sig <- d1; d1_sig[abs(d1) < 0.15] <- 0
perm1 <- list(edges = list(diffs_true = d1, diffs_sig = d1_sig, stats = NULL))
attr(perm1, "labels") <- c("A", "B", "C")
class(perm1) <- c("tna_permutation", "list")
gperm <- list("G1 vs. G2" = perm1)
```

```
class(gperm) <- c("group_tna_permutation", "list")
plot_group_permutation(gperm)
```

splot.net_bootstrap *Plot Nestimate Bootstrap Results*

Description

Visualizes `net_bootstrap` objects from the Nestimate package. Mirrors `splot.tna_bootstrap` but adapts to Nestimate's field layout: weights live under `$original$weights`, `directed` is not always `TRUE`, and there are no `donut/inits`.

Plots the original `tna` model with nodes colored by community membership. The original model is retrieved from `attr(x, "tna")`, which `tna::communities()` sets automatically. Uses `walktrap` if present in `x$assignments`; otherwise falls back to the first available algorithm column.

Plots the original network with nodes colored by community membership. The network is retrieved from `attr(x, "network")`, which `detect_communities()` / `.wrap_communities()` sets automatically.

Applies TNA-compatible styling defaults before delegating to `splot()`: directed networks get oval layout, colored nodes, and sized arrows; undirected networks get spring layout with no arrows or dashes. All parameters can be overridden by the caller.

Visualizes `boot_glasso` objects from the Nestimate package. Plots a partial-correlation network with edge inclusion probabilities mapped to edge transparency.

Plot a `wtna_mixed` object either as a single overlaid network or as two separate group panels.

Visualizes `net_permutation` objects from the Nestimate package. Differs from `plot_permutation`: `p_values` and `effect_size` are already `p` $\times$ `p` matrices (no edge-name parsing needed), and `directed` comes from `x$x$directed`.

Network visualization using base R graphics (similar to `qgraph`).

Creates a network visualization using base R graphics functions (`polygon`, `lines`, `xspline`, etc.) instead of grid graphics. This provides better performance for large networks and uses the same `snake_case` parameter names as `soplot()` for consistency.

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_online = 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 <code>group_tna</code> object (list of <code>tna</code> objects from <code>tna</code> package). Use parameter <code>i</code> to select a specific group, or omit to plot all groups.
<code>display</code>	Display mode: "styled" (default), "significant", or "full".
<code>show_ci</code>	Logical: overlay CI bounds on edge labels? Default FALSE.
<code>show_stars</code>	Logical: show significance stars? Default TRUE.
<code>inherit_style</code>	Logical: inherit labels/layout/colors from network? Default TRUE.
<code>...</code>	Additional arguments passed to layout functions. One ride-along worth calling out: <code>combined</code> (default TRUE). When <code>x</code> is a multi-panel input (a <code>group_tna</code> , <code>group_tna_bootstrap</code> , <code>group_tna_permutation</code> , <code>net_permutation_group</code> , or any class routed to a <code>splot.*</code> method that draws multiple panels such as <code>splot.net_mlvar</code> with <code>type = "all"</code>), <code>combined = FALSE</code> skips the internal <code>graphics::par(mfrow = ...)</code> grid so the caller can drive layout explicitly via <code>panel_layout()</code> or <code>graphics::layout()</code> . For single-network inputs (a single <code>tna</code> , <code>netobject</code> , <code>matrix</code> , etc.) <code>combined</code> has no effect — there is no panel grid to gate.
<code>type</code>	Character. "overlay" (default) renders both networks on a single canvas via <code>plot_mixed_network</code> — co-occurrence as straight undirected edges, transitions as curved directed arrows. "group" plots each component as a separate panel.
<code>combined</code>	Logical: when <code>type = "all"</code> , controls whether the three panels are arranged in an internal 1 x 3 grid (TRUE, default) or drawn into a layout the caller has already configured (FALSE — pair with <code>panel_layout()</code>). Ignored for single-network types.
<code>use_thresholded</code>	Logical: use <code>\$thresholded_pcor</code> ? If FALSE, uses <code>\$original_pcor</code> . Default TRUE.
<code>show_inclusion</code>	Logical: scale edge alpha by inclusion probability? Default TRUE.
<code>inclusion_threshold</code>	Numeric: minimum inclusion probability to show an edge. Default NULL, which uses $1 - x\$alpha$ (i.e. the complement of the alpha level, falling back to $1 - 0.05$ when <code>\$alpha</code> is absent).
<code>edge_positive_color</code>	Color for positive weights.
<code>edge_negative_color</code>	Color for negative weights.
<code>show_nonsig</code>	Logical: show non-significant edges? Default FALSE.
<code>show_effect</code>	Logical: show effect size in parentheses? Default FALSE.
<code>edge_nonsig_color</code>	Color for non-significant edges. Default "#888888".
<code>edge_nonsig_style</code>	Line style for non-significant edges. Default 2L.
<code>layout</code>	Layout algorithm: "oval" (default), "circle", "spring", "groups", "target" (qgraph-style focal-node BFS levels; node of interest via <code>target</code>), "sqqr" (Start/End transition flow; start/ end/jitter), or a matrix of x,y coordinates, or an <code>igraph</code> layout function. Also supports <code>igraph</code> 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 <code>edge_cutoff</code> 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	Reserved for future curve scaling; currently 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., <code>splot.tna_disparity</code>). 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_online	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 (indexed at each drawn edge automatically — the safe form when minimum/threshold filter 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=".

<code>edge_label_stars</code>	Stars for labels: character vector, TRUE (compute from p), or numeric (treated as p-values).
<code>weight_digits</code>	Number of decimal places to round edge weights to before plotting. Edges that round to zero are automatically removed. Default 2. Set NULL to disable rounding.
<code>threshold</code>	Minimum absolute weight to display.
<code>minimum</code>	Alias for threshold (qgraph compatibility). Uses max of threshold and minimum.
<code>maximum</code>	Maximum weight for scaling. NULL for auto.
<code>positive_color</code>	Deprecated. Use <code>edge_positive_color</code> instead.
<code>negative_color</code>	Deprecated. Use <code>edge_negative_color</code> instead.
<code>edge_duplicates</code>	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.
<code>title</code>	Plot title.
<code>title_size</code>	Title font size.
<code>margins</code>	Margins as <code>c(bottom, left, top, right)</code> .
<code>background</code>	Background color.
<code>rescale</code>	Logical: rescale layout to -1 to 1 range?
<code>layout_scale</code>	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.
<code>layout_margin</code>	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.
<code>aspect</code>	Logical: maintain aspect ratio?
<code>use_pch</code>	Logical: use points() for simple circles (faster). Default FALSE.
<code>usePCH</code>	Deprecated. Use <code>use_pch</code> instead.
<code>scaling</code>	Scaling mode: "default" for qgraph-matched scaling where <code>node_size=6</code> looks similar to qgraph <code>vsize=6</code> , or "legacy" to preserve pre-v2.0 behavior.
<code>align_panels</code>	Logical. If TRUE, forces a uniform symmetric plot box (<code>c(-layout_scale, layout_scale)</code> on each axis) so two networks plotted side-by-side in a <code>par(mfrow)</code> grid render at identical absolute scales — useful for bootstrap panels, comparison grids with networks of different node counts, or any case where visual-size parity across panels matters more than canvas fill. Default FALSE uses dynamic, layout-driven bounds (the pre-2.1.x behavior) which renders tighter on the canvas. The fixed box is only applied when the layout is being rescaled, so <code>align_panels = TRUE</code> has no effect under <code>rescale = FALSE</code> . The per-node loop-reservation pad in <code>compute_plot_limits</code> runs regardless, so networks with different self-loop patterns stay centered consistently in either mode.
<code>legend</code>	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 <i>x</i> is a tna object, FALSE otherwise. Can be used with any input type (matrix, igraph, co-graph_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_mlv. 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 <i>x</i> is a group_tna object. If NULL (default), plots all groups in a grid. If specified (e.g., <i>i</i> = 1 or <i>i</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

Edge Curve Behavior:

Edge curving is controlled by three parameters that interact:

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.

curve_scale Not currently used; reserved for future scaling.

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 dispatch cograph already performs for it — 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 "wtma_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

Invisibly returns the `cograph_network` object built by `splot()`. Called for the side effect of drawing.

Invisibly, the `splot` result: a `cograph_network` object.

Invisibly, the `splot` result: a `cograph_network` object.

Invisibly returns `x`.

Invisibly returns the `cograph_network` object built by `splot()`. Called for the side effect of drawing.

Invisibly returns the `cograph_network` object built by `splot()`. Called for the side effect of drawing.

Invisibly returns `x`.

Invisibly returns the `cograph_network` object built by `splot()`, or `NULL` when there is no edge to draw.

Invisibly returns the `cograph_network` object.

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

```
# Basic directed network
adj <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1,
               0, 0, 0, 1, 0, 0, 0, 0), 4, 4, byrow = TRUE)
splot(adj, layout = "circle", labels = c("A", "B", "C", "D"))

# Abbreviate long labels to a fixed maximum length
splot(adj, layout = "circle",
      labels = c("Orientation", "Planning", "Reading", "Submission"),
      label_abbrev = 4)

# Weighted network with signed edges
w_adj <- matrix(c(0, .5, -.3, 0, .8, 0, .4, -.2,
                  0, 0, 0, .6, 0, 0, 0, 0), 4, 4, byrow = TRUE)
splot(w_adj, edge_positive_color = "darkgreen", edge_negative_color = "red")
```

splot.tna_disparity	<i>Plot Disparity Results with splot</i>
---------------------	--

Description

Plot Disparity Results with `splot`

Usage

```
splot.tna_disparity(
  x,
  show = c("styled", "backbone", "full"),
  edge_style_sig = 1,
  edge_style_nonsig = 2,
  alpha_nonsig = 0.3,
  ...
)
```

Arguments

x A tna_disparity object.
show What to display: "styled" (default), "backbone", "full".
edge_style_sig Line style for backbone edges. Default 1 (solid).
edge_style_nonsig Line style for non-backbone edges. Default 2 (dashed).
alpha_nonsig Alpha for non-backbone edges. Default 0.3.
... Additional arguments passed to splot.

Value

Invisibly returns the value from the underlying `splot` call. Called primarily for the side effect of producing a plot.

Examples

```

mat <- matrix(c(0.0, 0.5, 0.1, 0.0, 0.3, 0.0, 0.4, 0.1,
               0.1, 0.2, 0.0, 0.5, 0.0, 0.1, 0.3, 0.0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("A", "B", "C", "D")
disp <- disparity_filter(cograph(mat), level = 0.05)
splot(disp)
splot(disp, show = "backbone")

```

splot.tna_permutation *Plot Permutation Test Results*

Description

Visualizes permutation test results with styling to distinguish significant from non-significant edge differences. Works with tna_permutation objects from the tna package.

Usage

```

splot.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,
  ...
)

```

Arguments

<code>x</code>	A <code>tna_permutation</code> object (from <code>tna::permutation_test</code>).
<code>...</code>	Additional arguments passed to <code>splot()</code> .
<code>show_nonsig</code>	Logical: show non-significant edges? Default FALSE (only significant shown).
<code>edge_positive_color</code>	Color for positive differences ($x > y$). Default "#009900" (green).
<code>edge_negative_color</code>	Color for negative differences ($x < y$). Default "#C62828" (red).
<code>edge_nonsig_color</code>	Color for non-significant edges. Default "#888888" (grey).
<code>edge_nonsig_style</code>	Line style for non-significant edges (2=dashed). Default 2.
<code>show_stars</code>	Logical: show significance stars (*, **, ***) on edges? Default TRUE.
<code>show_effect</code>	Logical: show effect size in parentheses for significant edges? Default FALSE.
<code>edge_nonsig_alpha</code>	Alpha for non-significant edges. Default 0.4.

Details

The function expects a `tna_permutation` object containing:

- `edges$diffs_true`: Matrix of actual edge differences ($x - y$)
- `edges$diffs_sig`: Matrix of significant differences only
- `edges$stats`: Data frame with `edge_name`, `diff_true`, `effect_size`, `p_value`

Edge styling:

- Significant positive: solid green, bold labels with stars
- Significant negative: solid red, bold labels with stars
- Non-significant (when `show_nonsig=TRUE`): dashed grey, plain labels, lower alpha

Value

Invisibly returns the `cograph_network` object built by `splot()`, or NULL when no edge survives the significance filter. Called for the side effect of drawing.

Examples

```
# Mock a tna_permutation object with synthetic data
diffs <- matrix(c(0, .15, -.1, -.2, 0, .05, .1, -.05, 0), 3, 3)
rownames(diffs) <- colnames(diffs) <- c("A", "B", "C")
diffs_sig <- diffs; diffs_sig[abs(diffs) < 0.1] <- 0
perm <- list(edges = list(
  diffs_true = diffs, diffs_sig = diffs_sig,
  stats = data.frame(
    edge_name = c("A -> B", "A -> C", "B -> A", "B -> C", "C -> A", "C -> B"),
    diff_true = c(.15, -.1, -.2, .05, .1, -.05),
```

```

    effect_size = c(2.1,-1.5,-2.8,.4,1.2,-.3),
    p_value      = c(.01,.04,.001,.3,.02,.5)))
attr(perm, "level") <- 0.05
attr(perm, "labels") <- c("A", "B", "C")
class(perm) <- c("tna_permutation", "list")
plot_permutation(perm)

```

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`), which may represent self-directed actions. These can be removed with `subset(student_interactions, from != to)`.

Because interactions repeat, this edge list naturally represents 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

```
# Load and build network
data(student_interactions)
head(student_interactions)

# Remove self-loops and build a network
e1 <- subset(student_interactions, from != to)
n_edges(as_cograph(e1))
n_nodes(as_cograph(e1))
```

subgraphs

Extract Specific Motif Instances (Subgraphs)

Description

Convenience wrapper for `motifs(x, named_nodes = TRUE, ...)`. Returns one row per concrete node-triple and MAN type. At individual level, observed counts sessions/units exhibiting that combination, so one triple can occupy multiple rows when its type differs across units. The same MAN type can also appear in many rows, each with its own z / p . For per-triple significance use `plot(., type = "significance")` or `plot(., type = "triads")`; the per-type plots ("types", "patterns") deliberately drop the significance decoration here, because aggregating per type requires a rule (median? max- $|z|$?) that isn't pinned and would be misleading by default.

Usage

```
subgraphs(...)
```

Arguments

... Arguments forwarded to `motifs()`. See `?motifs` for the full parameter list (`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 uses a canonical representative of the row's MAN isomorphism class. Concrete labels identify the participating nodes; their positions in that representative diagram do not encode the nodes' observed source/sink roles.

Value

A `cograph_motif_result` object with `named_nodes = TRUE`. Contains `$results` (data frame with columns `triad`, `node1`, `node2`, `node3`, `observed`, `type`, and when `significance = TRUE` also `expected`, `z`, `p`, `sig`), `$type_summary`, `$level`, `$n_units`, and `$params`. At individual level, each result row is a node-triple and MAN-type combination, and observed counts sessions/units exhibiting it. In instance mode, `$type_summary` is built via `table(results$type)` so it counts how many node-triples fall under each MAN type.

See Also`motifs()`

Other motifs: `extract_motifs()`, `extract_triads()`, `get_edge_list()`, `motif_census()`, `motifs()`, `plot.cograph_motif_analysis()`, `plot.cograph_motifs()`, `triad_census()`

Examples

```
mat <- matrix(c(0,3,2,0, 0,0,5,1, 0,0,0,4, 2,0,0,0), 4, 4, byrow = TRUE)
rownames(mat) <- colnames(mat) <- c("Plan", "Execute", "Monitor", "Adapt")
subgraphs(mat, significance = FALSE)
```

summarize_clusters	<i>Build MCML from Raw Transition Data</i>
--------------------	--

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 actual transitions. Unlike `csum` which aggregates a pre-computed weight matrix, this function works from the original transition data to produce the TRUE Markov chain over cluster states.

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.
- 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`.
- cluster_summary** Returns as-is.
- 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: <code>list(A = c("N1", "N2"), B = c("N3", "N4"))</code> data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: <code>data.frame(node = c("N1", "N2", "N3"), group = c("A", "A", "B"))</code> membership vector Character or numeric vector. Node names are extracted from the data. Example: <code>c("A", "A", "B", "B")</code> 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 <code>cograph_network\$nodes</code> or <code>\$node_groups</code> (same logic as <code>csum</code>).
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". Default "tna".
directed	Logical. Treat as directed network? Default TRUE.
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

```
# Edge list with clusters
edges <- data.frame(
  from = c("A", "A", "B", "C", "C", "D"),
  to   = c("B", "C", "A", "D", "D", "A"),
  weight = c(1, 2, 1, 3, 1, 2)
)
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"))
cs <- summarize_clusters(edges, clusters)
cs$macro$weights

# Sequence data with clusters
seqs <- data.frame(
  T1 = c("A", "C", "B"),
  T2 = c("B", "D", "A"),
  T3 = c("C", "C", "D"),
```

```
T4 = c("D", "A", "C")
)
cs <- summarize_clusters(seqs, clusters, type = "raw")
cs$macro$weights
```

summarize_network	<i>Summarize Network by Clusters</i>
-------------------	--------------------------------------

Description

Creates a summary network where each cluster becomes a single node. Edge weights are aggregated from the original network using the specified method. Returns a `cograph_network` object ready for plotting.

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 <code>cograph_network</code> .
cluster_list	Cluster specification: • Named list of node vectors (e.g., <code>list(A = c("n1", "n2"), B = c("n3", "n4"))</code>) • String column name from nodes data (e.g., "clusters", "groups") • NULL to auto-detect from common column names
method	Aggregation method for edge weights: "sum", "mean", "max", "min", "median", "density", "geomean". Default "sum".
directed	Logical. Treat network as directed. Default TRUE.

Value

A cograph_network object with:

- One node per cluster (named by cluster)
- Edge weights = aggregated between-cluster weights
- nodes\$size = cluster sizes (number of original nodes)

See [summarize_network](#).

See Also

[csum](#), [plot_mcml](#)

Examples

```
# Create a network with clusters
mat <- matrix(runif(100), 10, 10)
diag(mat) <- 0
rownames(mat) <- colnames(mat) <- LETTERS[1:10]

# Define clusters
clusters <- list(
  Group1 = c("A", "B", "C"),
  Group2 = c("D", "E", "F"),
  Group3 = c("G", "H", "I", "J")
)

# Create summary network
summary_net <- summarize_network(mat, clusters)
splot(summary_net)

# With cograph_network (auto-detect clusters column)
Net <- cograph(mat)
Net$nodes$clusters <- rep(c("A", "B", "C"), c(3, 3, 4))
summary_net <- summarize_network(Net) # Auto-detects 'clusters'
```

summary.cograph_network

Summary of cograph_network Object

Description

Summary of cograph_network Object

Usage

```
## S3 method for class 'cograph_network'
summary(object, ...)
```

Arguments

object A cograph_network object.
... Ignored.

Value

A list with network summary information (invisibly), containing elements n_nodes, n_edges, directed, weighted, and has_layout.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- cograph(adj)
summary(net)
```

supra_adjacency	<i>Supra-Adjacency Matrix</i>
-----------------	-------------------------------

Description

Builds the supra-adjacency matrix for multilayer networks. Diagonal blocks = intra-layer, off-diagonal = inter-layer.

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 (scalar or L x L matrix)
coupling Coupling type: "diagonal", "full", or "custom"
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 $\text{choose}(L, 2)$ 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 (legacy chain layout): entry i is the coupling for (i, i+1). Non-adjacent pairs use $\text{omega}[a,b] * I$.

If no entry matches a pair and no legacy chain layout applies, a warning is emitted and the diagonal default $\text{omega}[a,b] * I$ is used (previously this happened silently).

Value

A supra-adjacency matrix of dimension $(NL) \times (NL)$ with class `c("supra_adjacency", "matrix")`. Diagonal $N \times N$ 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

```
nodes <- c("A", "B", "C")
l1 <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3, dimnames = list(nodes, nodes))
l2 <- matrix(c(0, 1, 1, 1, 0, 0, 1, 0, 0), 3, 3, dimnames = list(nodes, nodes))
layers <- list(L1 = l1, L2 = l2)

# 3 nodes x 2 layers gives a 6 x 6 supra-adjacency matrix.
s <- supra_adjacency(layers, omega = 0.5)
dim(s)
s
```

supra_interlayer	<i>Extract Inter-Layer Block</i>
------------------	----------------------------------

Description

Extract Inter-Layer Block

Usage

```
supra_interlayer(x, from, to)

extract_interlayer(x, from, to)
```

Arguments

x	Supra-adjacency matrix
from	Source layer index
to	Target layer index

Value

Inter-layer adjacency matrix

Examples

```
L1 <- matrix(c(0,.5,.3,.5,0,.4,.3,.4,0), 3, 3)
L2 <- matrix(c(0,.2,.6,.2,0,.1,.6,.1,0), 3, 3)
S <- supra_adjacency(list(L1 = L1, L2 = L2), omega = 0.5)
supra_interlayer(S, 1, 2)
L1 <- matrix(c(0,.5,.3,.5,0,.4,.3,.4,0), 3, 3)
L2 <- matrix(c(0,.2,.6,.2,0,.1,.6,.1,0), 3, 3)
S <- supra_adjacency(list(L1 = L1, L2 = L2), omega = 0.5)
extract_interlayer(S, 1, 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
layer	Layer index to extract

Value

Intra-layer adjacency matrix

Examples

```
L1 <- matrix(c(0,.5,.3,.5,0,.4,.3,.4,0), 3, 3)
L2 <- matrix(c(0,.2,.6,.2,0,.1,.6,.1,0), 3, 3)
S <- supra_adjacency(list(L1 = L1, L2 = L2), omega = 0.5)
supra_layer(S, 1)
L1 <- matrix(c(0,.5,.3,.5,0,.4,.3,.4,0), 3, 3)
L2 <- matrix(c(0,.2,.6,.2,0,.1,.6,.1,0), 3, 3)
S <- supra_adjacency(list(L1 = L1, L2 = L2), omega = 0.5)
extract_layer(S, 2)
```

symmetrize

Symmetrize a Directed Network

Description

Combines each pair of opposite arcs into one undirected edge. The result is an undirected network, so measures that branch on directedness see the change.

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 rather than being compared against the zero that stands for the missing arc. That distinction matters for signed networks, where comparing a negative weight against a structural zero would delete the edge. Use "mutual" when an edge should survive only if it was reciprocated.

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

```
adj <- matrix(c(0, .5, 0,
               .2, 0, .7,
               0, .1, 0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

symmetrize(adj, method = "max")
symmetrize(adj, method = "mean")
symmetrize(adj, method = "mutual")
```

 themes-builtin

Built-in Themes

Description

Pre-defined themes for network visualization.

Value

A `CographTheme` object.

Examples

```
theme_cograph_classic()
theme_cograph_dark()
```

`theme_cograph_classic` *Classic Theme*

Description

Traditional network visualization style with blue nodes and gray edges.

Usage

```
theme_cograph_classic()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_classic()
```

`theme_cograph_colorblind`
Colorblind-friendly Theme

Description

Theme using colors distinguishable by people with color vision deficiency.

Usage

```
theme_cograph_colorblind()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_colorblind()
```

theme_cograph_dark	<i>Dark Theme</i>
--------------------	-------------------

Description

Dark background theme for presentations.

Usage

```
theme_cograph_dark()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_dark()
```

theme_cograph_gray	<i>Grayscale Theme</i>
--------------------	------------------------

Description

Black and white theme suitable for print.

Usage

```
theme_cograph_gray()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_gray()
```

theme_cograph_minimal *Minimal Theme*

Description

Clean, minimal style with thin borders.

Usage

```
theme_cograph_minimal()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_minimal()
```

theme_cograph_nature *Nature Theme*

Description

Earth tones theme inspired by nature.

Usage

```
theme_cograph_nature()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_nature()
```

theme_cograph_viridis	<i>Viridis Theme</i>
-----------------------	----------------------

Description

Theme using viridis color palette.

Usage

```
theme_cograph_viridis()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_viridis()
```

threshold_edges	<i>Threshold Edges by Weight, Count, Proportion or Density</i>
-----------------	--

Description

Keeps the edges that satisfy every criterion supplied. This is the network equivalent of qgraph's minimum/cut arguments and of tna::prune(), except that it returns a network rather than a plot setting, so the thresholded network can be analysed, not only drawn.

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: <code>cograph_network</code> , <code>matrix</code> , <code>igraph</code> , <code>network</code> , <code>tna</code> , 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	Integer. Keep this many edges, the strongest first.
absolute	Logical. Compare <code>abs(weight)</code> rather than the signed weight. Default <code>TRUE</code> , which is what correlation and partial-correlation networks need. <code>minimum/maximum</code> and the ranking used by <code>proportion</code> , <code>density</code> and <code>top</code> both follow this flag.
keep_isolates	Logical. Keep nodes that end up with no edges? Default <code>TRUE</code> . Set <code>FALSE</code> , or call <code>remove_isolates()</code> , to drop them.
keep_format	Logical. Return the input format when <code>TRUE</code> .
directed	Logical or <code>NULL</code> . If <code>NULL</code> (default), auto-detect.

Details

When several criteria are given they 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.

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. This is deliberate: breaking ties on edge order would make the result depend on how the network was built.

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. An out-of-range `minimum`, `maximum`, `proportion`, `density` or `top` raises 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

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

threshold_edges(adj, minimum = 0.5)
threshold_edges(adj, top = 2)
threshold_edges(adj, density = 0.5)
```

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

Any further edge columns the network carries (for example session or time from temporal edge lists) are *not* included; use [get_edges](#), which returns the edge table whole. An undirected network contributes one row per unordered pair, not two.

See Also

[to_df](#), [to_igraph](#), [as_cograph](#)

Examples

```
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")

# Convert to edge list
to_data_frame(adj)

# Use alias
to_df(adj)
```

to_directed	<i>Convert an Undirected Network to Directed</i>
-------------	--

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.
- keep_format Logical. Return the input format when TRUE.
- directed Logical or NULL. Directedness to read the input with.

Value

A directed `cograph_network`, or the input format when `keep_format = TRUE`.

See Also

[to_undirected](#), [reverse_edges](#)

Examples

```
adj <- matrix(c(0, 1, 0,
               1, 0, 1,
               0, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

to_directed(adj)
to_directed(adj, mode = "arbitrary")
```

to_igraph	<i>Convert Network to igraph Object</i>
-----------	---

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

```
# From matrix
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
g <- to_igraph(adj)

# Force directed
g_dir <- to_igraph(adj, directed = TRUE)
```

to_matrix	<i>Convert Network to Adjacency Matrix</i>
-----------	--

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

```
# From matrix
adj <- matrix(c(0, .5, .8, 0,
               .5, 0, .3, .6,
               .8, .3, 0, .4,
               0, .6, .4, 0), 4, 4, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C", "D")
to_matrix(adj)

# From cograph_network
net <- as_cograph(adj)
to_matrix(net)

# From igraph (weighted graph)
```

```

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::graph_from_adjacency_matrix(adj, mode = "undirected", weighted = TRUE)
  to_matrix(g)
}

```

to_network	<i>Convert Network to statnet network Object</i>
------------	--

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

```

if (requireNamespace("network", quietly = TRUE)) {
  adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
  rownames(adj) <- colnames(adj) <- c("A", "B", "C")
  net <- to_network(adj)
}

```

to_undirected

*Convert a Directed Network to Undirected***Description**

Collapses each pair of opposite arcs into one undirected edge. The counterpart of `igraph::as_undirected()` and `tidygraph`'s `to_undirected()`.

Usage

```
to_undirected(
  x,
  method = c("max", "sum", "mean", "min", "mutual"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

<code>x</code>	Network input.
<code>method</code>	How to combine <code>w[i, j]</code> and <code>w[j, i]</code> : "max" (default), "sum", "mean", "min", or "mutual" (keep only reciprocated pairs, taking the minimum weight).
<code>keep_format</code>	Logical. Return the input format when TRUE.
<code>directed</code>	Logical or NULL. Directedness to read the input with.

Value

An undirected `cograph_network`, or the input format when `keep_format = TRUE`. 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.

See Also

[to_directed](#), [symmetrize](#)

Examples

```
adj <- matrix(c(0, .5, 0,
               .2, 0, .7,
               0, 0, 0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")

to_undirected(adj, method = "sum")
to_undirected(adj, method = "mutual")
```

triad_census	<i>Triad Census</i>
--------------	---------------------

Description

Count the 16 types of triads in a directed network using MAN notation.

Usage

```
triad_census(x)
```

Arguments

x A matrix, igraph object, or cograph_network

Details

Triad census is defined only for directed networks. Matrix input is built as directed; existing igraph and cograph inputs must already be directed.

MAN notation describes triads by:

- M: number of Mutual (reciprocal) edges
- A: number of Asymmetric edges
- N: number of Null (absent) edges

The 16 triad types are: 003, 012, 102, 021D, 021U, 021C, 111D, 111U, 030T, 030C, 201, 120D, 120U, 120C, 210, 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\(\)](#), [plot.cograph_motif_analysis\(\)](#), [plot.cograph_motifs\(\)](#), [subgraphs\(\)](#)

Examples

```
set.seed(1)
mat <- matrix(sample(0:1, 100, replace = TRUE), 10, 10)
diag(mat) <- 0
# igraph and sna also export triad_census(); qualify the call.
cograph::triad_census(mat)
```

trophic_incoherence	<i>Trophic Incoherence Parameter</i>
---------------------	--------------------------------------

Description

The trophic incoherence parameter q is a measure of 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

Low values ($q \approx 0$) indicate a perfectly coherent network (e.g., a pure food web where every edge goes up one level). High values indicate an incoherent network with many level-skipping or downward edges. Johnson et al. 2014 showed that low- q food webs are dynamically more stable.

Matches `networkx.trophic_incoherence_parameter` at machine epsilon. Directed-only; requires at least one basal node (node with no incoming edges) for trophic levels to be well-defined.

Value

A single numeric value (NA_real_ for empty edge sets or undirected input).

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

```
# Small directed 3-node chain: 1 -> 2 -> 3 (perfectly coherent, q = 0)
adj <- matrix(c(0,1,0, 0,0,1, 0,0,0), 3, 3, byrow = TRUE)
rownames(adj) <- colnames(adj) <- c("A", "B", "C")
trophic_incoherence(adj)
```

unregister_svg_shape *Unregister SVG Shape*

Description

Remove a custom SVG shape from the registry.

Usage

```
unregister_svg_shape(name)
```

Arguments

name Shape name to remove.

Value

Invisible TRUE if removed, FALSE if not found.

Examples

```
# Attempt to unregister a non-existent shape (returns FALSE)
unregister_svg_shape("nonexistent")
```

verify_with_igraph *Verify Against igraph*

Description

Confirms numerical match with igraph's contract_vertices + simplify.

Usage

```
verify_with_igraph(x, clusters, method = "sum", type = "raw")

verify_igraph(x, clusters, method = "sum", type = "raw")
```

Arguments

<code>x</code>	Adjacency matrix
<code>clusters</code>	Cluster specification (see <code>csum</code>)
<code>method</code>	Aggregation method. Default "sum".
<code>type</code>	Normalization type. Defaults to "raw" for igraph compatibility.

Value

A list with components `our_result` (cograph's macro weight matrix), `igraph_result` (igraph's `contract()` + `simplify()` matrix), `matches` (logical: do the off-diagonals 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

```
if (requireNamespace("igraph", quietly = TRUE)) {
  mat <- matrix(runif(100), 10, 10)
  diag(mat) <- 0
  rownames(mat) <- colnames(mat) <- LETTERS[1:10]
  clusters <- c(1,1,1,2,2,2,3,3,3,3)
  verify_igraph(mat, 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

<code>x</code>	Network input: matrix, igraph, network, cograph_network, or tna object.
<code>directed</code>	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
<code>normalized</code>	Logical. If TRUE (default), return the proportional drop. If FALSE, return the raw efficiency difference.
<code>weighted</code>	Logical. If TRUE, honor edge weights when computing shortest paths (Dijkstra); distance is $1/\text{weight}^\alpha$ per the usual qgraph/tna convention when <code>invert_weights = TRUE</code> . If FALSE (default, matches prior behavior), all edges are treated as unit length.
<code>invert_weights</code>	Logical. If TRUE (default) and weights are present, invert weights to distances via $1/\text{weight}^\alpha$ so that higher weight = shorter path (matches <code>centrality()</code> 's default). Ignored when <code>weighted = FALSE</code> .
<code>alpha</code>	Weight-to-distance exponent (default 1).
<code>digits</code>	Integer or NULL. Round scores to this many decimal places. Default NULL (no rounding).
<code>...</code>	Currently unused; <code>directed</code> is already an explicit argument above and to_igraph accepts no others.

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:

$$E_{global} = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{1}{d(i, j)}$$

$E_{global \setminus i}$ is computed on the reduced graph but keeps the *original* $n(n-1)$ denominator, so vulnerability is bounded below by zero (Latora & Marchiori 2007); re-normalizing by $(n-1)(n-2)$ would let a node removal appear to raise efficiency.

Nodes with high vulnerability are critical to the network's communication efficiency. Removing them causes the greatest drop in global efficiency.

Performance note: This function computes all-pairs shortest paths once for the full graph and once per node removal, giving $O(n)$ calls to the shortest-path algorithm. 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 scores, sorted descending.

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.

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](https://doi.org/10.1088/13672630/9/6/188)

See Also

[network_global_efficiency](#), [robustness](#), [centrality](#)

Examples

```
# Star network: hub is most vulnerable
star <- matrix(c(0,1,1,1, 1,0,0,0, 1,0,0,0, 1,0,0,0), 4, 4)
rownames(star) <- colnames(star) <- c("hub", "a", "b", "c")
cograph::vulnerability(star)

# Complete graph: all nodes equally vulnerable
k4 <- matrix(1, 4, 4); diag(k4) <- 0
rownames(k4) <- colnames(k4) <- c("A", "B", "C", "D")
cograph::vulnerability(k4)
```

Index

- * **datasets**
 - hai_datasets, 294
 - regulation_net, 441
 - student_interactions, 519
- * **motifs**
 - extract_motifs, 269
 - extract_triads, 273
 - get_edge_list, 285
 - motif_census, 319
 - motifs, 313
 - plot.cograph_motif_analysis, 351
 - plot.cograph_motifs, 349
 - subgraphs, 520
 - triad_census, 541
- abbrev_label, 10
- add_edges, 11, 13, 27, 335, 442
- add_nodes, 12, 12, 335, 444
- aggregate_layers, 13, 335
- aggregate_weights, 14, 398
- ai_coding (hai_datasets), 294
- ai_detailed (hai_datasets), 294
- as.data.frame, 15, 16
- as.data.frame.cograph_motif_result, 15
- as.data.frame.cograph_network, 16, 285, 335
- as_cograph, 20, 21, 284, 285, 287–290, 298, 299, 335, 336, 339, 340, 468, 470, 535, 537–539
- as_mcml, 22
- as_tna, 23, 23, 254, 256, 397, 522
- assortativity, 17, 20
- assortativity_attribute, 18, 19
- barplot, 257, 354, 369, 381
- binarize, 25, 249, 334, 338, 534
- bind_networks, 12, 26, 335
- bootstrap, 260
- centrality, 18, 28, 45–54, 56–82, 84–88, 90–115, 117–128, 130–141, 143–151, 153–159, 161–163, 165, 166, 168–187, 189–202, 204–212, 252, 264, 267, 268, 276, 279, 294, 309, 310, 322, 365–370, 376, 449, 451, 461, 464, 465, 475, 542, 546
- centrality_access_information, 31, 45
- centrality_adaptive_leadersrank, 35, 46
- centrality_alpha, 48
- centrality_authority, 49, 190
- centrality_average_distance, 49
- centrality_barycenter, 50
- centrality_beta_measure, 35, 51
- centrality_betweenness, 52, 74, 106, 137, 144, 173, 188, 195
- centrality_bottleneck, 53
- centrality_bridging, 53, 83, 147, 184, 200
- centrality_bridging_capital, 35, 54
- centrality_bridging_coefficient, 153
- centrality_bridging_coefficient (centrality_truss), 199
- centrality_brokerage_coordinator, 31, 55, 57–60
- centrality_brokerage_gatekeeper, 56, 56
- centrality_brokerage_itinerant, 56, 57
- centrality_brokerage_liaison, 56, 58
- centrality_brokerage_representative, 56, 59
- centrality_cda, 35, 60
- centrality_centroid, 61
- centrality_closeness, 62, 75, 114, 140, 178
- centrality_closeness_vitality, 63, 170
- centrality_clusterrank, 64
- centrality_coleman_theil, 64
- centrality_comm_centrality (centrality_community_based), 67
- centrality_communicability, 66
- centrality_communicability_betweenness,

- 66
- centrality_community_based, [32](#), [67](#)
- centrality_community_hub_bridge, [31](#), [68](#), [69](#)
- centrality_community_mediator
(centrality_community_based), [67](#)
- centrality_constraint, [70](#), [92](#)
- centrality_controlrank, [43](#), [71](#)
- centrality_coreness, [72](#), [112](#), [152](#), [197](#), [200](#), [207](#)
- centrality_cross_clique, [73](#), [157](#)
- centrality_current_flow_betweenness, [74](#), [187](#), [188](#), [204](#)
- centrality_current_flow_closeness, [74](#), [127](#)
- centrality_dangalchev, [75](#), [184](#)
- centrality_decay, [76](#), [108](#)
- centrality_degree, [77](#), [165](#), [194](#), [205](#), [310](#)
- centrality_degree_discount, [31](#), [34](#), [78](#)
- centrality_delta_betweenness
(centrality_length_scaled_betweenness), [136](#)
- centrality_delta_closeness
(centrality_length_scaled_betweenness), [136](#)
- centrality_diffusion, [79](#), [81](#)
- centrality_diffusion_centrality, [35](#), [80](#), [89](#), [90](#)
- centrality_dil, [42](#), [81](#), [202](#), [203](#)
- centrality_distance_entropy, [31](#), [84](#), [149](#)
- centrality_diversity, [85](#)
- centrality_dkgm, [36](#), [42](#), [86](#), [116](#)
- centrality_dmnc, [87](#), [134](#), [161](#)
- centrality_dynamical_importance, [41](#), [88](#)
- centrality_dynamics_sensitive, [35](#), [89](#)
- centrality_eccentricity, [91](#)
- centrality_effective_size, [92](#), [118](#)
- centrality_ego_betweenness
(centrality_length_scaled_betweenness), [136](#)
- centrality_ehcc, [42](#), [92](#), [116](#)
- centrality_eigenvector, [48](#), [93](#), [130](#), [170](#), [174](#)
- centrality_enrenew
(centrality_wvoterank), [211](#)
- centrality_entropy, [94](#), [132](#)
- centrality_entropy_variation, [31](#), [95](#)
- centrality_epc
(centrality_local_efficiency), [150](#)
- centrality_exogenous, [35](#), [96](#)
- centrality_expected, [98](#), [99](#)
- centrality_expected_force, [98](#), [161](#)
- centrality_expected_influence_1, [100](#), [101](#)
- centrality_expected_influence_2, [100](#), [101](#)
- centrality_extended_coreness, [102](#), [167](#)
- centrality_extended_gravity, [103](#)
- centrality_extended_local_bridging, [104](#), [146](#)
- centrality_extended_mixed_gravity, [105](#), [160](#)
- centrality_flow_betweenness, [106](#)
- centrality_flow_coefficient
(centrality_heatmap), [116](#)
- centrality_fragmentation
(centrality_local_efficiency), [150](#)
- centrality_fuzzy_local_dimension
(centrality_local_dimension_fixed), [149](#)
- centrality_gateway, [107](#)
- centrality_generalized_closeness, [107](#)
- centrality_geodesic_kpath, [152](#)
- centrality_geodesic_kpath
(centrality_weighted_kshell), [206](#)
- centrality_gilschmidt, [108](#)
- centrality_global_structure, [109](#), [119](#), [125](#)
- centrality_godfather
(centrality_truss), [199](#)
- centrality_graph_regularization, [35](#), [110](#)
- centrality_gravity, [32](#), [104](#), [111](#), [137](#)
- centrality_harary, [113](#)
- centrality_harmonic, [62](#), [108](#), [113](#), [137](#), [181](#)
- centrality_hcc, [36](#), [42](#), [83](#), [92](#), [93](#), [114](#), [140](#)
- centrality_heatmap, [32](#), [116](#)
- centrality_hide_information
(centrality_access_information), [45](#)
- centrality_hub (centrality_authority),

- 49
- centrality_hubbell, 31, 118
- centrality_hybrid_global_structure, 109, 119
- centrality_iec, 42, 82, 120, 202, 203
- centrality_iira, 36, 42, 122, 129
- centrality_improved_closeness, 124
- centrality_improved_global_structure, 125
- centrality_incloseness (centrality_closeness), 62
- centrality_indegree (centrality_degree), 77
- centrality_ineccentricity (centrality_eccentricity), 91
- centrality_information, 31, 126
- centrality_inharmonic (centrality_harmonic), 113
- centrality_instrength (centrality_strength), 194
- centrality_integration, 127
- centrality_ira, 36, 42, 122, 123, 128
- centrality_katz, 31, 119, 130
- centrality_ked, 42, 131, 140
- centrality_kpath (centrality_local_efficiency), 150
- centrality_kreach, 112, 133, 205, 207
- centrality_lac, 133
- centrality_laplacian, 134
- centrality_leaderrank, 135, 208
- centrality_length_scaled_betweenness, 32, 136
- centrality_leverage, 137
- centrality_lhc, 36, 42, 82, 83, 138, 202, 203
- centrality_lin, 140
- centrality_linerank, 35, 141
- centrality_lnc, 42, 132, 142
- centrality_load, 52, 144
- centrality_lobby, 47, 145
- centrality_local_bridging, 146, 147
- centrality_local_dimension, 31, 84, 148, 149, 150, 153
- centrality_local_dimension_fixed, 32, 149
- centrality_local_efficiency, 32, 150
- centrality_local_entropy (centrality_heatmap), 116
- centrality_local_information_dimension, 31, 149, 150, 152
- centrality_local_volume_dimension (centrality_local_dimension_fixed), 149
- centrality_localized_bridging, 54, 104, 146, 147
- centrality_malatya, 153
- centrality_map_equation, 36, 154
- centrality_markov, 121, 122, 156
- centrality_mcc, 41, 157
- centrality_mcgm, 36, 87, 158
- centrality_mdd (centrality_truss), 199
- centrality_mixed_gravity, 86, 87, 105, 159
- centrality_mnc, 88, 160
- centrality_modified_expected_force, 99, 161
- centrality_modularity_vitality, 31, 70, 162
- centrality_ncvoterank, 31, 34, 163, 212
- centrality_neighbor_distance, 36, 42, 132, 140, 144, 165
- centrality_neighborhood_connectivity, 31, 164
- centrality_ninl, 35, 167
- centrality_node_contraction, 32, 169
- centrality_node_contraction_improved (centrality_node_contraction), 169
- centrality_outcloseness (centrality_closeness), 62
- centrality_outdegree (centrality_degree), 77
- centrality_outeccentricity (centrality_eccentricity), 91
- centrality_outharmonic (centrality_harmonic), 113
- centrality_outstrength (centrality_strength), 194
- centrality_pagerank, 94, 130, 135, 170, 203
- centrality_pairwisedis, 31, 171, 174
- centrality_participation, 68, 70, 107, 163, 172, 210
- centrality_percolation, 172
- centrality_power, 173
- centrality_prestige_domain, 174, 176

- centrality_prestige_domain_proximity, 175
- centrality_proximal_betweenness, 35, 176
- centrality_radiality, 177
- centrality_random_walk, 122, 178, 188
- centrality_random_walk_decay, 35, 179
- centrality_reaching_local, 31, 174, 176, 180, 437, 438
- centrality_redundancy (centrality_heatmap), 116
- centrality_relative_entropy, 37, 43, 181
- centrality_renewed_coreness (centrality_weighted_kshell), 206
- centrality_residual_closeness, 75, 184
- centrality_resistance_curvature, 41, 185
- centrality_rsp_betweenness, 37, 43, 121, 186
- centrality_rumor, 31, 188
- centrality_s_core (centrality_local_efficiency), 150
- centrality_s_shell, 31, 34, 196, 207
- centrality_salsa, 190
- centrality_semilocal, 144, 167, 190, 200
- centrality_shapley_game1, 31, 191
- centrality_shapley_game2, 34
- centrality_shapley_game2 (centrality_shapley_game1), 191
- centrality_shapley_game3, 34
- centrality_shapley_game3 (centrality_shapley_game1), 191
- centrality_single_discount (centrality_degree_discount), 78
- centrality_spectralrank, 36, 192
- centrality_strength, 77, 100, 194
- centrality_stress, 195
- centrality_subgraph, 66, 196, 269
- centrality_support (centrality_truss), 199
- centrality_topological_coefficient, 198
- centrality_transitivity, 118, 198
- centrality_truss, 35, 41, 199
- centrality_trust_pagerank, 36, 37, 42, 201
- centrality_two_way_rw, 32, 204
- centrality_volume, 35, 205
- centrality_voterank, 79, 163, 164, 206, 212
- centrality_voterank_plus (centrality_wvoterank), 211
- centrality_weighted_h_index (centrality_heatmap), 116
- centrality_weighted_kshell, 32, 152, 206
- centrality_weighted_leaderrank, 35, 208
- centrality_wiener, 209
- centrality_within_module_z, 163, 172, 210
- centrality_wvoterank, 32, 211
- centrality_x_degree, 212
- centralization, 213
- cluster_quality, 214, 215, 217
- cluster_significance, 215, 217
- cnet (summarize_network), 523
- coding (hai_datasets), 294
- coding_detailed (hai_datasets), 294
- cograph, 217, 281, 283, 479, 482, 485, 486, 489, 497, 516
- CographLayout, 219
- CographNetwork, 220
- CographTheme, 225
- color_communities, 228
- com_consensus (community_consensus), 231
- com_eb (community_edge_betweenness), 233
- com_fg (community_fast_greedy), 234
- com_fl (community_fluid), 235
- com_im (community_infomap), 236
- com_ld (community_leiden), 241
- com_le (community_leading_eigenvector), 239
- com_lp (community_label_propagation), 238
- com_lv (community_louvain), 242
- com_op (community_optimal), 243
- com_sg (community_spinglass), 245
- com_wt (community_walktrap), 247
- communities, 216, 217, 229, 232
- community_consensus, 231
- community_edge_betweenness, 230, 233
- community_fast_greedy, 230, 234
- community_fluid, 230, 235
- community_infomap, 230, 236

- community_label_propagation, 230, 238
- community_leading_eigenvector, 230, 239
- community_leiden, 230, 241
- community_louvain, 230, 232, 242
- community_optimal, 230, 243
- community_sizes, 244
- community_spring, 230, 245
- community_walktrap, 230, 247
- compare_communities, 248
- complement_network, 249, 335
- contract_nodes, 250, 335
- core_periphery, 251, 348
- cqual (cluster_quality), 214
- csig (cluster_significance), 215
- csum, 24, 25, 214, 253, 391, 396–398, 411, 521, 522, 524, 544
- degree_distribution, 256, 279, 376
- detect_communities, 20, 69, 162, 163, 228, 234–238, 240, 242–244, 246, 247, 251, 258, 335, 386, 398, 402, 409, 469
- disparity_filter, 259, 335, 499, 534
- dispersion, 261
- dyad_census, 262
- edge_betweenness (edge centrality), 264
- edge centrality, 264, 266, 322, 323, 472, 473
- edge_reciprocity, 263, 266
- ego_networks, 267, 335
- estrada_index, 268
- extract_interlayer (supra_interlayer), 526
- extract_layer (supra_layer), 527
- extract_motifs, 269, 274, 286, 318, 320, 350, 352, 521, 541
- extract_motifs(), 274, 286, 318, 320, 351, 352
- extract_triads, 272, 273, 286, 318, 320, 350, 352, 521, 541
- extract_triads(), 272
- filter_edges, 274, 275, 277, 334, 442, 443, 458, 475, 534
- filter_nodes, 275, 276, 334, 443, 444, 465
- fit_degree_distribution, 277, 349, 434
- from_qgraph, 218, 279, 283, 497, 516
- from_tna, 218, 281, 282, 497, 516
- get_data, 284
- get_edge_list, 272, 274, 285, 318, 320, 350, 352, 521, 541
- get_edges, 16, 21, 22, 284, 289, 468, 535
- get_groups, 286, 469
- get_labels, 21, 287
- get_layout, 287
- get_meta, 284, 288, 290
- get_nodes, 16, 21, 22, 285, 287, 289, 336, 340, 470
- get_shape, 289
- get_source, 288, 290
- get_theme, 291
- ggplot_robustness, 291
- group centrality, 292
- hai_datasets, 294
- hist, 257, 349
- homophily (assortativity_attribute), 19
- htna (plot_htna), 384
- human_ai (hai_datasets), 294
- human_ai_detailed (hai_datasets), 294
- invert_weights, 296, 334, 338
- is_bipartite, 297, 436
- is_directed, 22, 298
- is_tna_network, 298
- k_shortest_paths, 299, 472
- label_abbrev (abbrev_label), 10
- lagg (aggregate_layers), 13
- layer_degree_correlation, 301
- layer_similarity, 301, 303
- layer_similarity_matrix, 302
- layout_circle, 303
- layout_groups, 304
- layout_oval, 305
- layout_sqr, 306
- layout_spring, 307
- layout_target, 308
- ldegcor (layer_degree_correlation), 301
- list_centralities, 30, 32, 44, 83, 93, 116, 122, 123, 129, 132, 140, 144, 157, 167, 184, 188, 200, 203, 276, 309, 464, 465
- list_layouts, 310
- list_palettes, 311, 486
- list_shapes, 311

- `list_svg_shapes`, 312
- `list_themes`, 312, 489
- `lsim(layer_similarity)`, 301
- `lsim_matrix(layer_similarity_matrix)`, 302
- `membership`, 313
- `mlna(plot_mlna)`, 400
- `motif_census`, 272, 274, 286, 318, 319, 350, 352, 521, 541
- `motif_census()`, 271–274, 317, 318, 350, 352, 541
- `motifs`, 15, 272, 274, 286, 313, 320, 350, 352, 408, 520, 521, 541
- `motifs()`, 271, 272, 274, 320, 521, 541
- `mtna(plot_mtna)`, 408
- `mutate_edges`, 16, 320, 322, 335
- `mutate_nodes`, 13, 321, 321, 335
- `n_communities`, 338
- `n_edges`, 21, 22, 285, 339, 340
- `n_nodes`, 21, 22, 289, 336, 339, 339
- `neighborhood_overlap`, 268, 322, 473
- `network_bridges`, 323
- `network_clique_size`, 324
- `network_cut_vertices`, 325
- `network_girth`, 325
- `network_global_efficiency`, 326, 546
- `network_local_efficiency`, 152, 327
- `network_radius`, 328
- `network_rich_club`, 329
- `network_small_world`, 330
- `network_summary`, 18, 252, 263, 331, 376, 472
- `network_vertex_connectivity`, 333
- `network_wrangling`, 334
- `nodes`, 336
- `normalize_weights`, 26, 296, 334, 337, 529
- `overlay_communities`, 340
- `palette_blues`, 341
- `palette_colorblind`, 342
- `palette_diverging`, 342
- `palette_pastel`, 343
- `palette_rainbow`, 343
- `palette_reds`, 344
- `palette_viridis`, 344
- `palettes`, 341
- `panel_layout`, 317, 345, 350, 352, 357, 378, 412–415, 500, 506
- `plot`, 257, 354, 369, 376
- `plot.cograph_cluster_significance`, 346
- `plot.cograph_communities`, 347
- `plot.cograph_core_periphery`, 347
- `plot.cograph_degree_fit`, 348
- `plot.cograph_motif_analysis`, 272, 274, 286, 318, 320, 350, 351, 521, 541
- `plot.cograph_motif_result(motifs)`, 313
- `plot.cograph_motifs`, 272, 274, 286, 318, 320, 349, 352, 521, 541
- `plot.cograph_motifs()`, 320
- `plot.cograph_network`, 353
- `plot.cograph_rich_club`, 353
- `plot.cograph_vulnerability`, 354
- `plot.net_bootstrap_group`
(`plot_net_bootstrap_group`), 414
- `plot.netobject_group`
(`plot_netobject_group`), 411
- `plot.netobject_ml(plot_netobject_ml)`, 412
- `plot.tna_bootstrap`, 355
- `plot.tna_disparity`, 357
- `plot_alluvial`, 357, 429
- `plot_bootstrap_forest`, 361
- `plot_centrality`, 365
- `plot_centrality_compare`, 367
- `plot_centrality_distribution`, 368
- `plot_centrality_heatmap`, 370
- `plot_chord`, 371
- `plot_compare`, 373, 378
- `plot_comparison_heatmap`, 374
- `plot_degree_correlation`, 375
- `plot_difference`, 27, 373, 374, 376
- `plot_edge_diff_forest`, 379
- `plot_edge_weights`, 380
- `plot_group_permutation`
(`splot.group_tna_permutation`), 500
- `plot_heatmap`, 382, 436
- `plot_htna`, 384
- `plot_mcml`, 25, 256, 389, 411, 522, 524
- `plot_mixed_network`, 399, 506
- `plot_ml_heatmap`, 404
- `plot_mlna`, 396, 398, 400, 403, 423
- `plot_motifs`, 406
- `plot_mtna`, 256, 396, 398, 408, 411
- `plot_net_bootstrap_group`, 414
- `plot_net_stability`, 415

- plot_netobject_group, 411
- plot_netobject_ml, 412
- plot_network_evolution, 413, 423
- plot_permutation, 500
- plot_permutation
 - (splot.tna_permutation), 517
- plot_robustness, 416, 453
- plot_simplicial, 417
- plot_temporal, 421
- plot_tna, 411, 423
- plot_trajectories, 360, 426
- plot_transitions, 360, 429, 429
- print.cograph_communities, 433
- print.cograph_degree_fit, 434
- print.cograph_motif_analysis
 - (extract_motifs), 269
- print.cograph_motif_result(motifs), 313
- print.cograph_motifs(motif_census), 319
- print.cograph_network, 435
- project_bipartite, 435

- reaching_global, 181, 437
- register_layout, 438
- register_shape, 439
- register_svg_shape, 439
- register_theme, 440
- regulation_net, 441
- remove_edges, 12, 335, 442
- remove_isolates, 275, 335, 442, 443, 444, 458, 499, 534
- remove_nodes, 13, 335, 444
- rename_nodes, 335, 444, 446
- render-ggplot, 445
- render-grid, 446
- reorder_nodes, 335, 445, 446
- reverse_edges, 335, 447, 536
- rich_club, 329, 448, 451
- rich_club_local, 449, 450
- robustness, 171, 417, 449, 451, 453, 546
- robustness_auc, 453, 453
- robustness_summary, 454

- select_bridges, 455, 458
- select_component, 334, 456, 462, 465, 466, 499
- select_edges, 321, 334, 455, 457, 459, 460, 467
- select_edges_between, 459, 460
- select_edges_involving, 459, 460

- select_k_core, 334, 461
- select_neighbors, 268, 334, 456, 462, 465
- select_nodes, 276, 321, 322, 334, 446, 455, 456, 458, 461, 462, 463, 466
- select_top, 334, 465, 465, 467
- select_top_edges, 458, 466
- set_edges, 21, 467, 470
- set_groups, 286, 468
- set_layout, 21, 469
- set_nodes, 21, 445, 468, 470
- shortest_paths, 296, 300, 335, 471
- simmelian_strength, 323, 472
- simplify, 335, 473
- sn_edges, 218, 475, 482, 485, 489, 497, 516
- sn_ggplot, 446, 480
- sn_layout, 218, 470, 479, 481, 485, 497, 516
- sn_nodes, 218, 479, 482, 482, 486, 489, 497, 516
- sn_palette, 218, 486, 489
- sn_render(soplot), 489
- sn_save, 487
- sn_save_ggplot, 487
- sn_theme, 218, 479, 482, 485, 486, 488, 497, 516
- soplot, 218, 281, 283, 446, 479, 482, 485, 486, 489, 489, 516
- spanning_tree, 335, 498
- split_components, 251, 334, 443, 499
- splot, 21, 22, 218, 228, 275, 277, 281, 283, 286, 340, 347, 348, 354, 356, 357, 388, 396, 412, 414, 415, 469, 479, 482, 485, 486, 489, 497, 515–518, 522
- splot(splot.net_bootstrap), 501
- splot.group_tna_permutation, 500
- splot.net_bootstrap, 501
- splot.tna_bootstrap
 - (plot.tna_bootstrap), 355
- splot.tna_disparity, 516
- splot.tna_permutation, 517
- student_interactions, 519
- subgraphs, 15, 272, 274, 286, 318, 320, 350, 352, 408, 520, 541
- subgraphs(), 272, 274, 318
- subset_edges, 275
- subset_edges(filter_edges), 274
- subset_nodes, 277
- subset_nodes(filter_nodes), 276

`summarize_clusters`, [23](#), [250](#), [251](#), [335](#), [521](#)
`summarize_network`, [438](#), [523](#), [524](#)
`summary.cograph_network`, [524](#)
`supra(supra_adjacency)`, [525](#)
`supra_adjacency`, [525](#)
`supra_interlayer`, [526](#), [526](#)
`supra_layer`, [526](#), [527](#)
`symmetrize`, [334](#), [338](#), [528](#), [540](#)

`theme_cograph_classic`, [530](#)
`theme_cograph_colorblind`, [530](#)
`theme_cograph_dark`, [531](#)
`theme_cograph_gray`, [531](#)
`theme_cograph_minimal`, [532](#)
`theme_cograph_nature`, [532](#)
`theme_cograph_viridis`, [533](#)
`themes-builtin`, [529](#)
`threshold_edges`, [26](#), [334](#), [499](#), [533](#)
`tna::tna()`, [285](#)
`to_cograph(as_cograph)`, [20](#)
`to_data_frame`, [535](#), [537](#)
`to_df`, [16](#), [285](#), [335](#), [535](#), [538](#), [539](#)
`to_df(to_data_frame)`, [535](#)
`to_directed`, [335](#), [447](#), [536](#), [540](#)
`to_igraph`, [17](#), [19](#), [234–238](#), [240](#), [242–244](#),
 [246](#), [247](#), [251](#), [263](#), [267](#), [299](#),
 [323–330](#), [333](#), [335](#), [448](#), [450](#), [452](#),
 [471](#), [535](#), [537](#), [538](#), [539](#), [545](#)
`to_matrix`, [335](#), [538](#), [539](#)
`to_network`, [335](#), [538](#), [539](#)
`to_undirected`, [249](#), [335](#), [447](#), [529](#), [536](#), [540](#)
`tplot`, [389](#)
`tplot(plot_tna)`, [423](#)
`triad_census`, [262](#), [263](#), [272](#), [274](#), [286](#), [318](#),
 [320](#), [350](#), [352](#), [521](#), [541](#)
`trophic_incoherence`, [542](#)

`unregister_svg_shape`, [543](#)

`verify_igraph(verify_with_igraph)`, [543](#)
`verify_with_igraph`, [543](#)
`vulnerability`, [544](#)

`wagg(aggregate_weights)`, [14](#)