Package 'cograph'

July 10, 2026

Title Analysis and Visualization of Complex Networks

Version 2.4.4

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, Matrix, R6, stats, utils

Suggests backbone, brainGraph, centiserve, colorspace, digest, dplyr, gifski, gridExtra, grImport2, igraph, influenceR, jsonlite, knitr, Nestimate, network, qgraph, RColorBrewer, reticulate, rmarkdown, rsvg, scales, 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 2026-07-10 16:10:02 UTC

Contents

`abbrev_label` *Abbreviate Labels*

Description

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

Usage

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

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

Arguments

`label`	Character vector of labels to abbreviate.
`abbrev`	Abbreviation control: • NULL: No abbreviation (return labels unchanged) • Integer: Maximum character length (truncate + ellipsis) • "auto": 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")
```

`aggregate_layers` *Aggregate Layers*

Description

Combines multiple network layers into a single network.

Usage

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

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

Arguments

`layers`	List of adjacency matrices
`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 *Aggregate Edge Weights*

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

w	Numeric vector of edge weights
method	Aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", "geomean"
n_possible	Number of possible edges (for density calculation)

Value

Single aggregated value

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
```

`assortativity` *Degree Assortativity Coefficient*

Description

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

Usage

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

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
`type`	Character string specifying which degree correlation to compute. One of `"out-in"` (default for directed), `"in-in"`, `"out-out"`, `"in-out"`, or `"degree"` (for undirected). Ignored for undirected networks.
`digits`	Integer or NULL. Round result to this many decimal places. Default NULL (no rounding).
`...`	Additional arguments passed to `to_igraph`.

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.

For directed networks, different degree combinations (in/out) at source and target ends can be specified via the `type` parameter.

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

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.
...	Additional arguments passed to to_igraph.

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.

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

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 statnet network object • A qgraph object • A tna object • An existing cograph_network object (returned as-is)
directed	Logical. Force directed interpretation. NULL for auto-detect.
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 (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 n×n 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 tna::group_model(cluster_data(...))).
method	Aggregation method for macro weights (default "sum").
type	Transition type (default "tna").
directed	Logical; whether the network is directed (default TRUE).

Value

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

An mcml object.

An mcml object. When clusters is provided, macro$data contains the cluster assignments and macro$weights is NULL (the 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 silently excluded. 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 `$weights` (k x k transition matrix), `$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 `$weights` (n_i x n_i matrix), `$inits` (initial distribution), and `$labels` (node labels). Clusters with single nodes or zero-row nodes are excluded (tna requires positive row sums).

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

```
mat <- matrix(runif(36), 6, 6); diag(mat) <- 0
rownames(mat) <- colnames(mat) <- LETTERS[1:6]
clusters <- list(G1 = c("A","B"), G2 = c("C","D"), G3 = c("E","F"))
cs <- csum(mat, clusters, type = "tna")
tna_models <- as_tna(cs)
names(tna_models)          # "macro", "G1", "G2", "G3"
splot(tna_models$macro)    # cograph renderer avoids tna's plot deps
```

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,
  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,
  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 available measure.

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 every measure. Any custom vector of valid measure names is also accepted. **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).

`mode` 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.

`normalized` 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 igraph.

`weighted` Logical. Use edge weights if available. Default TRUE.

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

`loops` Logical. If TRUE (default), keep self-loops. Set to FALSE to remove them before calculation.

`simplify` How to combine multiple edges between the same node pair. Options: "sum" (default), "mean", "max", "min", or FALSE/"none" to keep multiple edges.

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

`sort_by` Character or NULL. Column name to sort results by (descending order). Default NULL (original node order).

`cutoff` Maximum path length to consider for betweenness, closeness, and harmonic centrality. Default -1 (no limit). Set to a positive value for faster computation on large networks at the cost of accuracy.

`invert_weights` 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.

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

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

`personalized` Named numeric vector for personalized PageRank. Default NULL (standard PageRank). Values should sum to 1.

`transitivity_type`
Type of transitivity to calculate: "local" (default), "global", "undirected", "localundirected", "barrat" (weighted), "weighted", or "onnela". The first six dispatch to `igraph::transitivity()`; `"onnela"` computes the Onnela / Holme weighted clustering coefficient on the symmetrized matrix (`wcc(x + t(x))`) and matches `tna::centralities(., "Clustering")` byte-for-byte. Auto-set to `"onnela"` when `tna_network = TRUE` and the user did not pass an explicit value.

`isolates` How to handle isolate nodes in transitivity calculation: "nan" (default) returns NaN, "zero" returns 0.

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

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

k	Path length parameter for geodesic k-path centrality. Default 3.
states	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).
decay_parameter	Numeric. Decay parameter for decay and generalized closeness centrality. Default 0.5. Must be between 0 and 1.
dmnc_epsilon	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.
membership	Integer vector of community assignments (one per node) for community-aware measures: participation, within_module_z, gateway, 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 satisfy $w \cdot \rho(W) \leq 1$ for solvability. Default 0.5. Only used when "hubbell" is in measures.
tna_network	Logical or NULL. Umbrella switch that forces tna-style conventions across all measures. NULL (default) auto-detects from the input class — TRUE iff x is a tna or related sequence-network object. TRUE forces tna conventions even on raw matrices: invert_weights = TRUE, loops = FALSE, diffusion_method = "power_series", transitivity_type = "onnela". FALSE suppresses all tna defaults even for tna inputs, giving the cograph defaults verbatim. Precedence: any arg the user passes explicitly always wins over tna_network.
psych_network	Logical or NULL. Switch for signed psychometric network conventions. NULL (default) auto-detects TRUE when a signed weighted network is evaluated with expected-influence measures. When TRUE, normalized expected influence is divided by the maximum absolute expected-influence value, preserving sign and bounding the result from -1 to 1. FALSE keeps the generic cograph normalization convention.
...	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/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 (diam + 1 - d) normalized by n-1.

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

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

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

dangalchev Dangalchev closeness (alias for residual closeness).

generalized_closeness Generalized closeness. Sum of alpha^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 / sum of distances.

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

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

communicability Total communicability. Row sums of matrix exponential.

communicability_betweenness Communicability betweenness. Fraction of communicability through each node.

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

stress Stress centrality. Number of shortest paths through node.

flow_betweenness Flow betweenness. Max-flow based betweenness.

lobby Lobby index (h-index of neighborhood).

entropy Graph entropy centrality. Entropy change on node removal.

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

clusterrank ClusterRank. Clustering coefficient times neighbor degree sum.

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

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

mnc Maximum Neighborhood Component size.

dmnc Density of Maximum Neighborhood Component.

topological_coefficient Topological coefficient. Shared neighbor ratio.

bridging Bridging centrality. Betweenness times bridging coefficient.

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

effective_size Burt's effective size. Degree minus redundancy.

diversity Diversity centrality. Shannon entropy of edge weight distribution.

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

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

integration Integration centrality. Distance-based influence.

expected Expected centrality. Sum of neighbor degrees.

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

salsa SALSA authority scores (directed graphs only).

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

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

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

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

Value

A data frame with columns:

- `node`: Node labels/names
- One column per measure, with mode suffix for directional measures (e.g., `degree_in`, `closeness_all`)

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_alpha *Alpha (Katz) Centrality*

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_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_local_bridging for the local 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_brokerage_coordinator *Gould-Fernandez Brokerage — Coordinator Role*

Description

Coordinator brokerage (w_I): count of open directed 2-paths $A \to V \to A$ 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_g
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

Gould-Fernandez Brokerage — Gatekeeper Role

Description

Gatekeeper brokerage (b_OI): count of open directed 2-paths $A \to V \to B$ 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 & Fernandez (1989).

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 $A \to V \to A$ 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_O"]`. Directed-only.

Value

Named integer vector of itinerant role counts.

References

Gould & Fernandez (1989).

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 $A \to V \to B$ 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_O"]`. Directed-only.

Value

Named integer vector of liaison role counts.

References

Gould & Fernandez (1989).

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 $A \to V \to B$ 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

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_IO"]. Directed-only.

Value

Named integer vector of representative role counts.

References

Gould & Fernandez (1989).

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_centroid *Centroid Value*

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

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 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

`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 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_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_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_coreness *K-Core Decomposition (Coreness)*

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 *Decay Centrality*

Description

Sum of delta^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

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. Default 0.5.
...	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 *Degree Centrality*

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

`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 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_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

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

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 $\mathrm{rowSums}(P + P^2 + \ldots + P^n)$ on the weighted matrix (with `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_diversity *Diversity Centrality*

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_dmnc *Density of Maximum Neighborhood Component (DMNC)*

Description

Edge count divided by max component size^1.5 in the neighborhood subgraph.

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.

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_eccentricity *Eccentricity*

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

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 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_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(adj)
```

---

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_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_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 alpha^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_gilschmidt *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_gilschmidt(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_gilschmidt(adj)
```

centrality_harary *Harary Centrality*

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 *Harmonic Centrality*

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_hubbell *Hubbell Centrality*

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

x	Network input (matrix, igraph, network, cograph_network, tna object).
hubbell_weight	Attenuation factor w. Default 0.5. If $w \cdot \rho(W) \geq 1$, the function returns NA with a warning.
...	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_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 $IC_i = 1/(C_{ii} + (\mathrm{tr}(C) - 2R_i)/n)$ 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_katz` *Katz Centrality*

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_kreach *Geodesic K-Path Centrality*

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 *Local Average Connectivity (LAC)*

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

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 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

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	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_leverage *Leverage Centrality*

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_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_load *Load Centrality*

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 *Lobby Index (H-Index of Neighborhood)*

Description

Largest k such that the node's closed neighborhood contains at least k nodes with degree >= k. Network analogue of the h-index.

Usage

```
centrality_lobby(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 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_local_bridging *Local Bridging Centrality*

Description

(1/degree) times bridging coefficient. Local measure of inter-community connectivity.

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_markov *Markov Centrality*

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_mnc *Maximum Neighborhood Component (MNC)*

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_pagerank *PageRank Centrality*

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., Voss, N., Sasse, N., & Wingender, E. (2008). Topology of mammalian transcription networks. *Genome Informatics*, 18, 193-204.

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.

$$\text{domain}(v) = |\{u \neq v : u \to^* v\}|$$

Usage

```
centrality_prestige_domain(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"). Directed-only; returns NA with a warning on undirected input.

Value

Named numeric vector of domain prestige values in $\{0, 1, \ldots, N-1\}$.

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 `(counts > 0) * gdist` element-wise and then sums to get the denominator. For any pair where `gdist = Inf` (unreachable), R evaluates `FALSE * Inf = 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_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

x	Network input (matrix, igraph, network, cograph_network, tna object).
...	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_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

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

Details

- Directed unweighted: $LRC(v) = |\{u : u \neq v, v \to u\}|/(N-1)$.
- 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_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_salsa *SALSA Authority Centrality*

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 *Semi-Local Centrality*

Description

Triple-nested neighborhood computation measuring 4-hop local influence.

Usage

```
centrality_semilocal(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 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_strength *Strength Centrality (Weighted Degree)*

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 *Stress Centrality*

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_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 `"onnela"`. `"onnela"` computes the Onnela / Holme weighted clustering coefficient on the symmetrized matrix and matches `tna::centralities(., "Clustering")` byte-for-byte. Auto-set to `"onnela"` 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_voterank *VoteRank Centrality*

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_wiener *Wiener Index Centrality*

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))
```

---

centralization *Centralization index*

Description

Computes Freeman's centralization for degree, betweenness, closeness, or eigenvector centrality.

Usage

```
centralization(
  x,
  measure = c("degree", "betweenness", "closeness", "eigenvector"),
  directed = NULL,
  mode = "all",
  ...
)
```

Arguments

x	Network input
measure	One of "degree", "betweenness", "closeness", "eigenvector"
directed	Logical or NULL
mode	"all", "in", or "out"
...	Additional arguments passed to to_igraph()

Value

Numeric scalar in $[0, 1]$

Examples

```
star <- matrix(0, 5, 5)
star[1, 2:5] <- 1; star[2:5, 1] <- 1
cograph::centralization(star, "degree")
```

`cluster_quality` *Cluster Quality Metrics*

Description

Computes per-cluster and global quality metrics for network partitioning. Supports both binary and weighted networks.

Usage

```
cluster_quality(x, clusters, weighted = TRUE, directed = TRUE)

cqual(x, clusters, weighted = TRUE, directed = TRUE)
```

Arguments

`x`	Adjacency matrix
`clusters`	Cluster specification (list or membership vector)
`weighted`	Logical; if TRUE, use edge weights; if FALSE, binarize
`directed`	Logical; if TRUE, treat as directed network

Value

A `cluster_quality` object with:

`per_cluster`	Data frame with per-cluster metrics
`global`	List of global metrics (modularity, coverage)

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

`x`	Network input: adjacency matrix, igraph object, or cograph_network.
`communities`	A communities object (from `communities` or igraph) or a membership vector (integer vector where `communities[i]` is the community of node i).
`n_random`	Number of random networks to generate for the null distribution. Default 100.
`method`	Null model type: **"configuration"** Preserves degree sequence (default). More stringent test. **"gnm"** Erdos-Renyi model with same number of edges. Tests against random baseline.
`null`	Which null question to answer. Default `"detect"`: **"detect"** Null is the modularity of the *best partition found by community detection* 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 `communities` membership *evaluated on each null graph*. Answers "does the supplied partition itself explain more structure than it would on similar random graphs?" — the conservative test.

`seed` 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: (observed - null_mean) / 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 sn_layout() 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")
```

CographLayout *CographLayout R6 Class*

Description

Class for managing layout algorithms and computing node positions.

Value

A CographLayout R6 object.

Methods

Public methods:

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()
```

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)
```

`CographNetwork` *CographNetwork R6 Class*

Description

Core class representing a network for visualization. Stores nodes, edges, layout coordinates, and aesthetic mappings.

Value

A `CographNetwork` R6 object.

Active bindings

`n_nodes` Number of nodes in the network.

`n_edges` Number of edges in the network.

`is_directed` Whether the network is directed.

`has_weights` 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.

Method `set_edges()`**:** Set edges data frame.

Usage:

```
CographNetwork$set_edges(edges)
```

Arguments:

`edges` Data frame with edge information.

Method `set_directed()`**:** Set directed flag.

Usage:

```
CographNetwork$set_directed(directed)
```

Arguments:

`directed` Logical.

Method `set_weights()`**:** Set edge weights.

Usage:

```
CographNetwork$set_weights(weights)
```

Arguments:

`weights` Numeric vector of weights.

Method `set_layout_coords()`**:** Set layout coordinates.

Usage:

```
CographNetwork$set_layout_coords(coords)
```

Arguments:

`coords` Matrix or data frame with x, y columns.

Method `set_node_aes()`**:** Set node aesthetics.

Usage:

```
CographNetwork$set_node_aes(aes)
```

Arguments:

`aes` List of aesthetic parameters.

Method `set_edge_aes()`**:** Set edge aesthetics.

Usage:

```
CographNetwork$set_edge_aes(aes)
```

Arguments:

`aes` List of aesthetic parameters.

Method `set_theme()`**:** Set theme.

Usage:

```
CographNetwork$set_theme(theme)
```

Arguments:

`theme` CographTheme object or theme name.

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.).

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.

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()
```

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.

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()
```

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` *Color Nodes by Community*

Description

Generate colors for nodes based on community membership. Designed for direct use with `splot()` `node_fill` parameter.

Usage

```
color_communities(x, method = "louvain", palette = NULL, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`method`	Community detection algorithm. See `detect_communities` for available methods. Default `"louvain"`.
`palette`	Color palette to use. Can be: • `NULL` (default): Uses a colorblind-friendly palette • A character vector of colors • A function that takes n and returns n colors • A palette name: "rainbow", "colorblind", "pastel", "viridis"
`...`	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 *Community Detection*

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(E² n)
leading_eigenvector	Networks with dominant structure	O(n²)
spinglass	Small networks, allows negative weights	O(n³)
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` *Consensus Community Detection*

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).
n_runs	Number of times to run the algorithm. Default 100.
threshold	Co-occurrence threshold for consensus. Default 0.5. Nodes that appear together in >= threshold proportion of runs are placed in the same community.
seed	Optional seed for reproducibility. If provided, the RNG state is initialized once before repeated runs.
...	Additional arguments passed to the community detection method.

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 >= threshold)
5. Run walktrap on the consensus graph to get final communities

Value

A cograph_communities object with consensus membership.

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.
...	Additional arguments passed to to_igraph

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.
`...`	Additional arguments passed to `to_igraph`

Value

A `cograph_communities` object with optional dendrogram

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` *Fluid Communities Detection*

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.
`...`	Additional arguments passed to `to_igraph`

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` *Infomap Community Detection*

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

`x`	Network input
`weights`	Edge weights for transitions. NULL uses network weights, NA for unweighted.
`v.weights`	Vertex weights (teleportation weights).
`nb.trials`	Number of optimization trials. Default 10.
`modularity`	Logical; calculate modularity? Default TRUE.
`seed`	Random seed for reproducibility. Default NULL.
`...`	Additional arguments passed to `to_igraph`

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.
...	Additional arguments passed to to_igraph

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

x	Network input
weights	Edge weights. NULL uses network weights, NA for unweighted.
steps	Maximum number of splits. Default -1 (until modularity decreases).
start	Starting community structure (membership vector).
options	ARPACK options list. Default uses igraph::arpack_defaults().
callback	Optional callback function called after each split.
extra	Extra argument passed to callback.
env	Environment for callback evaluation.
...	Additional arguments passed to to_igraph

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 *Leiden Community Detection*

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.
...	Additional arguments passed to to_igraph

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 *Louvain Community Detection*

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.
...	Additional arguments passed to to_igraph

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 *Optimal Community Detection*

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.
...	Additional arguments passed to to_igraph

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 *Get Community Sizes*

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 *Spinglass Community Detection*

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

x	Network input
weights	Edge weights. NULL uses network weights, NA for unweighted.
vertex	Vertex to find community for (single community mode). NULL for full partitioning.
spins	Number of spins (maximum communities). Default 25.
parupdate	Parallel update mode. Default FALSE.
start.temp	Starting temperature. Default 1.
stop.temp	Stopping temperature. Default 0.01.
cool.fact	Cooling factor. Default 0.99.
update.rule	Update rule: "config" (default), "random", or "simple".
gamma	Gamma parameter for modularity. Default 1.
implementation	"orig" (default) or "neg" (for negative weights).
gamma.minus	Gamma for negative weights in "neg" implementation.
seed	Random seed for reproducibility. Default NULL.
...	Additional arguments passed to to_igraph

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` *Walktrap Community Detection*

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.
...	Additional arguments passed to to_igraph

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 *Compare Community Structures*

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")
}
```

---

core_periphery *Detect Core-Periphery Structure*

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

x	Network input: matrix, igraph, network, cograph_network, or tna object
method	Character string; either "continuous" (default, Borgatti-Everett model) or "discrete" (binary core/periphery assignment).
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

iter	Integer; maximum number of iterations for the continuous algorithm. Default 100.
digits	Integer or NULL. Round numeric outputs to this many decimal places. Default NULL (no rounding).
...	Additional arguments passed to to_igraph

Details

Continuous method (Borgatti-Everett): Finds a coreness vector c (values 0-1) that maximizes the correlation between the adjacency matrix and the ideal rank-1 pattern matrix (the outer product of the coreness vector with itself). The algorithm initializes from eigenvector centrality and iteratively refines via power iteration until convergence.

Discrete method: Produces a binary core (1) / periphery (0) assignment. Starts from the continuous solution, thresholds at the median, then greedily swaps node assignments to maximize fitness. Discrete fitness is defined by high density within the core and low density within the periphery.

Value

A data frame with class "cograph_core_periphery" and columns node, role, and coreness. Fitness, core density, periphery density, and the original network are stored as attributes.

References

Borgatti, S.P. & Everett, M.G. (2000). Models of core/periphery structures. *Social Networks*, 21(4), 375-395. doi:10.1016/S03788733(99)000192

See Also

centrality, network_summary

Examples

```
# Core-periphery 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$assignment
```

csum *Cluster Summary Statistics*

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 x 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	method="sum", type="tna"	Preserves total flow, normalizes to probabilities
Transition matrix	method="mean", type="tna"	Avoids cluster size bias
Frequencies	method="sum", type="raw"	Keep raw counts for analysis
Correlation matrix	method="mean", type="raw"	Average correlations

Value

A `cluster_summary` object (S3 class) containing:

macro A tna object representing the macro (cluster-level) network:

- **weights** k x k matrix of cluster-to-cluster weights, where k is the number of clusters. Row i, column j contains the aggregated weight from cluster i to cluster j. 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

x	Network input: matrix, igraph, network, cograph_network, or tna object.
mode	For directed networks: "all", "in", or "out". Default "all".
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
loops	Logical. If TRUE (default), keep self-loops. Set FALSE to remove them.
simplify	How to combine multiple edges between the same node pair. Options: "sum" (default), "mean", "max", "min", or FALSE/"none" to keep multiple edges.
cumulative	Logical. If TRUE, show CCDF (complementary cumulative distribution: P(degree >= k)) instead of frequency. Default FALSE.
breaks	Bin specification passed to hist. Can be a numeric vector of breakpoints, a single number giving the number of bins, or a character string naming an algorithm (e.g. "Sturges", "FD", "scott"). Overrides bins and bin_width. Default NULL (auto-detect).
bins	Integer. Approximate number of bins. Overrides bin_width. Default NULL.
bin_width	Numeric. Width of each bin. Default NULL (auto: 1 when the degree range is ≤ 50, otherwise Freedman-Diaconis).
normalize	Logical. If TRUE, the y-axis shows proportions (bars sum to 1) instead of counts. Default FALSE.
log	Character. Axis log-scaling: "" (none, default), "x", "y", or "xy". Histogram plots apply y-axis log scaling for "y" or "xy"; cumulative plots support x, y, and xy scaling, with "xy" producing a log-log CCDF (standard for power-law inspection).
main	Character. Plot title. Default "Degree Distribution".
xlab	Character. X-axis label. Default "Degree".
ylab	Character. Y-axis label. Default auto-chosen based on normalize and cumulative.
col	Character. Bar/line fill color. Default "steelblue".
border	Character. Bar border color. Default "white".
...	Additional graphical arguments passed to barplot (histogram) or plot (cumulative).

Value

Invisibly returns a list with components:

degree Named numeric vector of per-node degrees.

table Table of degree frequencies.

breaks Breakpoints used for the histogram (non-cumulative only).

counts Bin counts (non-cumulative only).

proportions Bin proportions (non-cumulative only).

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 *Detect Communities in a Network*

Description

Detects communities (clusters) in a network using various community detection algorithms. Returns a data frame with node-community assignments.

Usage

```
detect_communities(x, method = "louvain", directed = NULL, weights = TRUE)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
method	Community detection algorithm to use. One of: • "louvain": Louvain 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 data frame with columns:

- node: Node labels/names
- community: Integer community membership

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 *Disparity Filter*

Description

Extracts the statistically significant backbone of a weighted network using the disparity filter method (Serrano, Boguna, & Vespignani, 2009).

Usage

```
disparity_filter(x, level = 0.05, ...)

## Default S3 method:
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'matrix'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'tna'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'cograph_network'
disparity_filter(x, level = 0.05, ...)

## S3 method for class 'igraph'
disparity_filter(x, level = 0.05, ...)
```

Arguments

x	A weight matrix, tna object, cograph_network, or igraph object.
level	Significance level (default 0.05). Lower values result in a sparser backbone (fewer edges retained).
...	Additional arguments (currently unused).

Details

The disparity filter identifies edges that carry a disproportionate fraction of a node's total weight, based on a null model where weights are distributed uniformly at random.

For each node i with degree k_i, and each edge (i, j) with normalized weight $p_{ij} = w_{ij}/s_i$ (where s_i is the node's strength), the p-value is:

$$p = (1 - p_{ij})^{(k_i - 1)}$$

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` *Dispersion (Backstrom-Kleinberg 2014)*

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 `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 (names are the other endpoints).
- A data frame with columns `from`, `to`, `dispersion` when neither `u` nor `v` is given (one row per ordered edge).

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

`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.
`...`	Additional arguments passed to `to_igraph`.

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

x	Network input (matrix, igraph, network, cograph_network, tna object)
measures	Which measures to calculate. Default "all" calculates all available edge measures. Options: "betweenness", "weight", "overlap", "simmelian", "reciprocity".
weighted	Logical. Use edge weights if available. Default TRUE.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
cutoff	Maximum path length for betweenness. Default -1 (no limit).
invert_weights	Logical or NULL. Invert weights for path-based measures? Default NULL (auto-detect: TRUE for tna objects, FALSE otherwise).
alpha	Numeric. Exponent for weight inversion. Default 1.
digits	Integer or NULL. Round numeric columns. Default NULL.
sort_by	Character or NULL. Column to sort by (descending). Default NULL.
...	Additional arguments passed to to_igraph

Details

Edge measures available:

betweenness Number of shortest paths passing through the edge.

weight Original edge weight.

overlap Jaccard neighborhood overlap of edge endpoints.

simmelian Number of triangles the edge participates in.

reciprocity Whether the reverse edge exists (directed only). Adds columns: reciprocated, reverse_weight, weight_ratio.

Value

A data frame with columns from, to, and one column per requested measure.

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` *Edge Reciprocity*

Description

Convenience wrapper around `edge_centrality` that returns only reciprocity information for directed networks.

Usage

```
edge_reciprocity(x, top = NULL, directed = NULL, digits = NULL, ...)
```

Arguments

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`top`	Integer or NULL. Return only the top N edges. Default NULL.
`directed`	Logical or NULL. Default NULL (auto-detect).
`digits`	Integer or NULL. Round numeric columns. Default NULL.
`...`	Additional arguments passed to `edge_centrality`.

Value

A data frame with columns: `from`, `to`, `weight`, `reciprocated`, `reverse_weight`, `weight_ratio`.

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 *Ego-Network Metrics*

Description

Extracts the ego network of each requested node (the node, its neighbours 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 neighbourhood order defining the ego network. 1 (default) is the standard ego network (ego + direct neighbours). 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 neighbourhood: "all" (default), "out", or "in".
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry.
...	Additional arguments passed to to_igraph.

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 ~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).

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 significance test. Default 100.
seed	Random seed for reproducibility.
n	Number of motif rows to print.
...	Passed to methods; currently unused.

Value

A `cograph_motif_analysis` object (list) containing:

results Data frame with triad, type, observed count, and (if significance=TRUE) expected, z-score, p-value

type_summary Summary counts by motif type

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(tna::group_regulation)
# 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` *Extract Triads with Node Labels*

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 = FALSE,
  keep_format = FALSE,
  directed = NULL
)

subset_edges(
  x,
  ...,
  .keep_isolates = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input: cograph_network, matrix, igraph, network, or tna object.
`...`	Filter expressions using any edge column (e.g., `weight > 0.5`, `weight > mean(weight)`, `abs(weight) > 0.3`).
`.keep_isolates`	Logical. Keep nodes with no remaining edges? Default FALSE.
`keep_format`	Logical. If TRUE, matrix, igraph, and statnet network inputs are returned in that format. Default FALSE returns cograph_network (universal format).
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected. Only used for non-cograph_network inputs.

Value

A cograph_network object with filtered edges. If `keep_format = TRUE`, matrix, igraph, and statnet network inputs are converted back to that type.

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
)

subset_nodes(
  x,
  ...,
  .keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = 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` Examples: `degree >= 3`, `label %in% c("A", "B")`, `pagerank > 0.1 & degree >= 2`.
.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 cograph_network (universal format).
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected. Only used for non-cograph_network inputs.

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

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`distributions`	Character vector of distributions to fit. Options: `"power_law"`, `"exponential"`, `"poisson"`, `"geometric"`. Default `NULL` fits all four.
`mode`	For directed networks: `"all"`, `"in"`, or `"out"`. Determines which degree to extract. Default `"all"`.
`directed`	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
`xmin`	Minimum degree to include in fitting. For power-law, NULL triggers automatic estimation (Clauset et al. 2009 via igraph). For other distributions, NULL defaults to 1.
`...`	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: $p = 1/(1+\bar{k})$.

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 qgraph::qgraph()
engine	Which cograph renderer to use: "splot" or "soplot". Default: "splot".
plot	Logical. If TRUE (default), immediately plot using the chosen engine.
weight_digits	Number of decimal places to round edge weights to. Default 2. Edges that round to zero are removed unless show_zero_edges = TRUE.
show_zero_edges	Logical. If TRUE, keep edges even if their weight rounds to zero. 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., minimum) or cograph names (e.g., threshold).

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
- asize -> arrow_size (scaled by 0.3x)

Graph properties:

- minimum -> threshold
- maximum -> maximum
- groups -> groups
- directed -> directed
- posCol/negCol -> edge_positive_color/edge_negative_color

Pie/Donut:

- pie values -> donut_fill with donut_inner_ratio=0.8

- `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 tna::tna()
engine	Which cograph renderer to use: "splot" or "soplot". Default: "splot".
plot	Logical. If TRUE (default), immediately plot using the chosen engine.
weight_digits	Number of decimal places to round edge weights to. Default 2. Edges that round to zero are removed unless show_zero_edges = TRUE.
show_zero_edges	Logical. If TRUE, keep edges even if their weight rounds to zero. Default: FALSE.
...	Additional parameters passed to the plotting engine (e.g., layout, node_fill, donut_color).

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

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(tna::group_regulation)
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 columns: from, to, weight.

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(tna::group_regulation)

# 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 *Get Labels from Cograph Network*

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 *Get a Registered Layout*

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 *Get Nodes from Cograph Network*

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 *Get a Registered Shape*

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 *Get a Registered Theme*

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` *Compare Network Robustness (ggplot2)*

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

`...`	Named arguments: network names as names, network objects as values.
`networks`	Named list of networks (alternative to ...).
`measures`	Attack strategies to compare. Default c("betweenness", "degree", "random").
`strategy`	Character string; "sequential" (default) recalculates centrality after each removal, "static" uses initial centrality ranking throughout.
`colors`	Named vector of colors for measures.
`title`	Overall title. Default NULL.
`n_iter`	Iterations for random. Default 1000.
`seed`	Random seed. Default NULL.
`type`	Removal type. Default "vertex".
`ncol`	Columns in facet. Default NULL (auto).
`free_y`	If TRUE, allow different y-axis scales per facet. Default FALSE.

Value

A 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-Yahalom-Elovici iterative algorithm, which produces results that diverge from the textbook Everett-Borgatti / Puzis 2008 "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., Yahalom, R., & Elovici, Y. (2008). Augmentative data collection for betweenness centrality. In *Advances in Social Networks Analysis and Mining* (pp. 196-200). IEEE.

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 (project + session_id); columns T1, T2, ... 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. The first columns identify the session; the remaining columns (T1, T2, ...) hold the sequential action codes, with `NA` indicating the session ended before that time step. 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)
head(coding[, 1:6])
dim(coding)
```

---

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 *Check if Network is Directed*

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 *Check if Network is TNA-based*

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(tna::group_regulation)
net_tna <- as_cograph(model)
is_tna_network(net_tna)  # TRUE
```

k_shortest_paths *Find K Shortest Loopless Paths (Yen's Algorithm)*

Description

Computes up to k shortest loopless paths between two nodes using Yen's algorithm. Each path is a sequence of distinct nodes from source to target.

Usage

```
k_shortest_paths(x, from, to, k = 3, weights = NULL, directed = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
from	Character or numeric node identifier for the source node.
to	Character or numeric node identifier for the target node.
k	Integer; number of shortest paths to find. Default 3.
weights	Edge weight handling: NULL (default) auto-detects from edge attributes, NA forces unweighted distances, or a numeric vector of custom weights.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
...	Additional arguments passed to to_igraph

Details

Yen's algorithm finds the k shortest loopless (simple) paths in a graph. It works by:

1. Finding the shortest path via Dijkstra's algorithm
2. For each subsequent path, systematically exploring deviations from previously found paths by temporarily removing edges, finding spur paths, and selecting the shortest candidate

The algorithm may return fewer than k paths if fewer distinct loopless paths exist between the two nodes.

Value

A list with class "cograph_k_paths" containing:

paths List of up to k character vectors, each containing node names in path order

distances Numeric vector of path lengths (sum of edge weights or hop count)

from Source node name

to Target node name

k Number of paths requested

References

Yen, J.Y. (1971). Finding the K shortest loopless paths in a network. *Management Science*, 17(11), 712-716. doi:10.1287/mnsc.17.11.712

See Also

shortest_paths

Examples

```
# 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` *Degree Correlation Between Layers*

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 *Layer Similarity*

Description

Computes similarity between two network layers.

Usage

```
layer_similarity(
  A1,
  A2,
  method = c("jaccard", "overlap", "hamming", "cosine", "pearson")
)

lsim(A1, A2, method = c("jaccard", "overlap", "hamming", "cosine", "pearson"))
```

Arguments

A1	First adjacency matrix
A2	Second adjacency matrix
method	Similarity method: "jaccard", "overlap", "hamming", "cosine", "pearson"

Value

Numeric similarity value

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 *Pairwise Layer Similarities*

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	List of adjacency matrices (one per layer)
method	Similarity method

Value

Symmetric matrix of pairwise similarities

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 *Circular Layout*

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 *Group-based Layout*

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` *Saqr Layout (Start/End transition flow)*

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 *Fruchterman-Reingold Spring Layout*

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 *Target Layout (focal-node, topological)*

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_layouts *List Available Layouts*

Description

List Available Layouts

Usage

```
list_layouts()
```

Value

Character vector of registered layout names.

Examples

```
list_layouts()
```

list_palettes *List Available Color Palettes*

Description

Returns the names of all registered color palettes.

Usage

```
list_palettes()
```

Value

Character vector of palette names.

Examples

```
list_palettes()
```

list_shapes *List Available Shapes*

Description

List Available Shapes

Usage

```
list_shapes()
```

Value

Character vector of registered shape names.

Examples

```
list_shapes()
```

list_svg_shapes *List Registered SVG Shapes*

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 *List Available Themes*

Description

List Available Themes

Usage

```
list_themes()
```

Value

Character vector of registered theme names.

Examples

```
list_themes()
```

membership *Get Community Membership*

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 *Network Motif Analysis*

Description

Two modes of motif 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,
  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). `subgraphs()` 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 (session_id, session, actor, user, participant). 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`.

`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. Default 1000.

`min_count` Inclusive minimum count to keep a row — rows with `count >= min_count` are retained. In instance mode (`named_nodes = TRUE`) 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 (`named_nodes = FALSE`) 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), "expected", or "percent".

`edge_threshold` Threshold for "expected" or "percent" methods. 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). 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 graphics::par(mfrow=...). Set to FALSE to draw into a layout the caller has already configured (e.g. via panel_layout()).

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.

Value

A `cograph_motif_result` object (a list) with:

results Data frame of results. Census mode (`named_nodes = FALSE`): one row per 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 with columns `triad`, `type`, `observed`, and when `significance = TRUE` also `expected`, `z`, `p`, `sig`.

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_analy`
`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(tna::group_regulation)
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

`x`	A matrix, igraph object, or cograph_network
`size`	Motif size: 3 (triads) or 4 (tetrads). Default 3.
`n_random`	Number of random networks for null model. Default 100.
`method`	Null model method: "configuration" (preserves degree) or "gnm" (preserves edge count). Default "configuration".
`directed`	Logical. Treat as directed? Default auto-detected.
`seed`	Random seed for reproducibility
`...`	Passed to methods; currently unused.

Value

A `cograph_motifs` data frame with motif count, null-model mean, null-model standard deviation, z-score, p-value, and significance columns. The motif size, directed flag, null-model method, and number of random networks are stored as attributes.

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)
```

---

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 *Bridge Edges*

Description

Finds edges whose removal would disconnect the network. These are critical edges for network connectivity.

Usage

```
network_bridges(x, count_only = FALSE, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
count_only	Logical. If TRUE, return only the count. Default FALSE.
...	Additional arguments passed to to_igraph

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 *Largest Clique Size*

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
...	Additional arguments passed to to_igraph

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 *Cut Vertices (Articulation Points)*

Description

Finds nodes whose removal would disconnect the network. These are critical nodes for network connectivity.

Usage

```
network_cut_vertices(x, count_only = FALSE, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
count_only	Logical. If TRUE, return only the count. Default FALSE.
...	Additional arguments passed to to_igraph

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 *Network Girth (Shortest Cycle Length)*

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
...	Additional arguments passed to to_igraph

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 *Global Efficiency*

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. Consider edge direction? Default TRUE for directed graphs.
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.
...	Additional arguments passed to to_igraph

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)  # ~0.83
```

network_local_efficiency *Local Efficiency*

Description

Computes the average local efficiency across all nodes. Local efficiency of a node is the global efficiency of its neighborhood subgraph (excluding the node itself). Measures fault tolerance and local integration.

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.
...	Additional arguments passed to to_igraph

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
```

network_radius *Network Radius*

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. Consider edge direction? Default TRUE for directed graphs.
...	Additional arguments passed to to_igraph

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 *Rich Club Coefficient*

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

x	Network input: matrix, igraph, network, cograph_network, or tna object
k	Degree threshold. Only nodes with degree > k are included. If NULL, uses median degree.
normalized	Logical. Normalize by random graph expectation? Default FALSE.
n_random	Number of random graphs for normalization. Default 10.
...	Additional arguments passed to to_igraph

Value

Numeric: rich club coefficient (> 1 indicates rich club effect when normalized)

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 *Small-World Coefficient (Sigma)*

Description

Computes the small-world coefficient sigma, defined as: sigma = (C / C_rand) / (L / L_rand) where C is clustering coefficient, L is mean path length, and _rand are values from equivalent random graphs.

Usage

```
network_small_world(x, n_random = 10, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object
n_random	Number of random graphs for comparison. Default 10.
...	Additional arguments passed to to_igraph

Details

Values > 1 indicate small-world properties. Typically small-world networks have sigma » 1.

Value

Numeric: small-world coefficient sigma

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` *Network-Level Summary Statistics*

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.

`extended`	Logical. If TRUE, include additional structural metrics (girth, radius, clique size, cut vertices, bridges, efficiency). Default FALSE.
`digits`	Integer. Round numeric results to this many decimal places. Default 3.
`...`	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

hub_score Maximum hub score (HITS algorithm)

authority_score Maximum authority score (HITS algorithm)

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
`...`	Additional arguments passed to `to_igraph`

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
```

---

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
```

n_communities *Get Number of Communities*

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 *Get Number of Edges*

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 *Get Number of Nodes*

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 *Overlay Community Blobs on a Network Plot*

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

x	A network object passed to splot: tna, matrix, igraph, or cograph_network.
communities	Community assignments in any format: a method name (e.g., "walktrap", "louvain"), a numeric or factor membership vector (e.g., c(1, 1, 2, 2, 3)), a named list of character vectors, a cograph_communities object, or a tna_communities object.
blob_colors	Character vector of fill colors for blobs. Recycled if shorter than the number of communities.
blob_alpha	Numeric. Fill transparency (0-1).
blob_linewidth	Numeric. Border line width.
blob_line_alpha	Numeric. Border line transparency (0-1).
...	Additional arguments passed to splot.

Value

The splot result (invisibly).

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")))
```

```
model <- tna::tna(tna::group_regulation)
comm <- cograph::communities(model$weights, method = "infomap")
overlay_communities(model, comm)
```

---

palettes *Color Palettes*

Description

Built-in color palettes for network visualization.

Examples

```
palette_blues(5)
palette_reds(5)
```

---

palette_blues *Blues Palette*

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 *Colorblind-friendly Palette*

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 *Diverging Palette*

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 *Pastel Palette*

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 *Rainbow Palette*

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 *Reds Palette*

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 *Viridis Palette*

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 *Configure a custom multi-panel layout*

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

`spec`	Either a length-2 integer vector `c(nrow, ncol)` for a uniform grid, or a numeric matrix of panel positions to pass to `graphics::layout()`.
`mar`	Numeric vector of length 4 giving panel margins. Default `c(2, 2, 3, 1)` matches cograph's multi-panel margin convention.
`widths, heights`	Optional numeric vectors of column widths and row heights. Only valid when `spec` is a matrix; passed straight to `graphics::layout()`. Supplying them with a uniform-grid `spec` is an error, since `par(mfrow=...)` 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

x	A cograph_communities object
network	The original network (required if not stored)
...	Additional arguments passed to splot

Value

Invisibly returns the plot

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

x	A cograph_degree_fit object from fit_degree_distribution.
which	Character vector of distribution names to display. Default NULL shows all fitted distributions.
log	Character string for log-scale axes: "" (default), "y", or "xy". Values containing "x" are accepted for compatibility but only filter non-positive fitted curve values.
cols	Named or unnamed character vector of colors for distribution curves. Default uses a built-in palette.
lwd	Line width for fitted curves. Default 2.
main	Plot title. Default "Degree Distribution Fit".
...	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

x	A cograph_motifs object from motif_census()
type	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 |z-score|.
show_nonsig	Show non-significant motifs? Default FALSE.
top_n	Show only top N motifs by |z-score|. Default NULL (all).
colors	Three-element color vector for under-represented, neutral, and over-represented motifs. Default c("#2166AC", "#999999", "#B2182B") (blue/gray/red).
combined	Logical: when TRUE (default) and type = "network", 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()). Has no effect for type = "bar" or type = "heatmap".
...	Additional arguments passed to plotting functions

Value

A ggplot2 object (invisibly)

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 showing over- and under-represented types. 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

x	A cograph_network object.
...	Additional arguments passed to sn_render.

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 *Plot Bootstrap Results*

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 plot.

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 *Plot Disparity Filter Result*

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 *Plot Alluvial Diagram*

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.

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.

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 colour; 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: inherits from the object ($ci_level or $alpha), 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-colour vector for grouped radial layouts.
sig_color	Colour for significant CI bars and points. Default "#2C6E8A" (teal-blue).
cr_color	Colour for the consistency range bar (interval = "cr" or "both"). Default "#D4829A".
nonsig_color	Colour for non-significant edges. Default "#CCCCCC".
ring_color	Colour for the reference rings (radial layout only). Default "#C8C8C8".
median_color	Colour 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 colour for edge labels (radial layout only). NULL (default) inherits the edge colour (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)
d    <- tna::prepare_data(
  tna::group_regulation_long,
  actor = "Actor", time = "Time", action = "Action"
)
Mod  <- tna::tna(d)
boot <- tna::bootstrap(Mod, iter = 50)
plot_bootstrap_forest(boot, n_top = 8)
```

`plot_centrality` *Plot Centrality*

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 *Chord Diagram*

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

x	A weight matrix, cograph_network, CographNetwork, tna, igraph, or list-like object with a matrix weights component.
directed	Logical. If NULL (default), auto-detected from matrix symmetry.
segment_colors	Colors for the outer ring segments. NULL uses a built-in vibrant palette.
segment_border_color	Border color for segments.
segment_border_width	Border width for segments.
segment_pad	Gap between segments in radians.
segment_width	Radial thickness of the outer ring as a fraction of the radius.
chord_color_by	How to color chords: "source" (default), "target", or a color vector of length matching the number of non-zero edges.
chord_alpha	Alpha transparency for chords.
chord_border	Border color for chords. NA for no border.
self_loop	Logical. Currently accepted for API compatibility; the current matrix preparation preserves self-loop chords.
labels	Node labels. NULL uses row names, FALSE suppresses labels.
label_size	Text size multiplier for labels.
label_color	Color for labels.
label_offset	Radial offset of labels beyond the outer ring.
label_threshold	Hide labels for nodes whose flow fraction is below this value.
threshold	Minimum absolute weight to show a chord.
ticks	Logical. Draw tick marks along the outer ring to indicate magnitude?
tick_interval	Spacing between ticks in the same units as the weight matrix. NULL (default) auto-selects a nice interval.
tick_labels	Logical. Show numeric labels at major ticks?
tick_size	Text size multiplier for tick labels.
tick_color	Color for tick marks and labels.
start_angle	Starting angle in radians (default pi/2, top).
clockwise	Logical. Lay out segments clockwise?
title	Optional plot title.
title_size	Text size multiplier for the title.
background	Background color for the plot. NULL (default) uses the current device background.
...	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)

if (requireNamespace("tna", quietly = TRUE)) {
  # TNA transition network
  model <- tna::tna(tna::group_regulation)
  plot_chord(model, 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. NULL for all pairs.
j	Index/name of second group when x is group_tna. NULL for all pairs.
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 TRUE if inits available.
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 graphics::par(mfrow=...). Set to FALSE to draw into a layout the caller has already configured (e.g. via panel_layout()). 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 tna_comparison object (from tna::compare()) is detected automatically and its $difference_matrix is used.
...	Additional arguments passed to splot().

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 difference matrix and inits difference.

See Also

plot_compare, the deprecated alias of this function.

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

Visualises 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

x	A boot_glasso object with $boot_edges and $edge_diff_p.
...	Currently unused.
alpha	Significance threshold. Default: inherits from object.
layout	"linear" (default), "circular", "chord", or "tile". The chord layout places all edge names on a unit circle and connects significant pairs with bezier arcs; arc width and colour encode the mean bootstrap difference. The tile layout draws the pairwise-difference matrix.
show_nonsig	Include non-significant pairs? Default FALSE.

nonzero_only	If TRUE, restrict to edges that are non-zero in the original network (identified via $original_pcor). Useful for EBICglasso results where many edges are regularised to exactly zero. Default FALSE.
sort_by	"estimate" (default), "significance", or "name" (linear only).
n_top	Restrict to top N pairs by absolute difference.
pos_color	Colour when edge1 > edge2. Default crimson.
neg_color	Colour when edge1 < edge2. Default teal.
nonsig_color	Colour for non-significant pairs.
ring_color	Ring colour (circular/chord). Default light grey.
label_size	Text size. Default 2.3.
label_color	Fixed label colour (NULL = inherit).
point_size	Size of estimate square (linear/circular).
r_inner	Inner ring radius (circular). Default 0.38.
r_outer	Outer ring radius (circular). Default 0.72.
title	Plot title.
subtitle	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)
```

---

plot_edge_weights *Plot Edge Weight Distribution*

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

x	Network input: matrix, igraph, network, cograph_network, or tna.
normalize	Logical. Show proportions. Default FALSE.
bins	Integer or NULL. Number of bins. Default NULL (auto).
log	Character. Log scaling. Default "".
directed	Logical or NULL. Default NULL (auto-detect).
col	Fill color. Default "steelblue".
border	Border color. Default "white".
main	Title. Default "Edge Weight Distribution".
xlab	X-axis label. Default "Weight".
...	Additional arguments passed to barplot.

Value

Invisibly 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

x	Network input: matrix, CographNetwork, cograph_network, tna, igraph, group_tna, or a list-like object with a $weights matrix.
cluster_list	Optional list of character vectors defining node clusters. Creates a block-structured heatmap with clusters along diagonal.
cluster_spacing	Gap size between clusters (in cell units). Default 0.
show_legend	Logical: display color legend? Default TRUE.

`legend_position`
: Position: "right" (default), "left", "top", "bottom", "none".

`legend_title`
: Title for legend. Default "Weight".

`colors`
: Color palette: vector of colors for gradient, or a palette name ("viridis", "heat", "blues", "reds", "greens", "diverging"). Default "viridis".

`limits`
: Numeric vector c(min, max) for color scale. NULL for auto.

`midpoint`
: Midpoint for diverging scales. NULL for auto (0 if data spans neg/pos).

`na_color`
: Color for NA values. Default "grey90".

`show_values`
: Logical: display values in cells? Default FALSE.

`value_size`
: Text size for cell values. Default 2.5.

`value_color`
: Color for cell value text. Default "black".

`value_fontface`
: Font face for values: "plain", "bold", "italic", "bold.italic". Default "plain".

`value_fontfamily`
: Font family for values: "sans", "serif", "mono". Default "sans".

`value_halo`
: Halo color behind value labels for readability on dark cells. Set to a color (e.g., "white") to enable, or NULL (default) to disable.

`value_digits`
: Decimal places for values. Default 2.

`show_diagonal`
: Logical: show diagonal values? Default TRUE.

`diagonal_color`
: Accepted for API compatibility; diagonal cells currently use the active fill scale unless hidden with `show_diagonal = FALSE`.

`cluster_labels`
: Logical: show cluster/layer labels? Default TRUE.

`cluster_borders`
: Logical: draw borders around clusters? Default TRUE.

`border_color`
: Color for cluster borders. Default "black".

`border_width`
: Width of cluster borders. Default 0.5.

`row_labels`
: Row labels. NULL for auto (rownames or indices).

`col_labels`
: Column labels. NULL for auto (colnames or indices).

`show_axis_labels`
: Logical: show axis tick labels? Default TRUE.

`axis_text_size`
: Size of axis labels. Default 8.

`axis_text_angle`
: 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 `abs(value) < threshold` 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 colour 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 *Plot Heterogeneous TNA Network (Multi-Group Layout)*

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,
  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,
  extend_lines = FALSE,
  scale = 1,
  nodes = NULL,
  label_abbrev = NULL,
  ...
)
```

Arguments

x	A tna object, weight matrix, or cograph_network.
node_list	Node groups can be specified as: • A list of character vectors (node names per group) • A string column name from nodes data (e.g., "groups") • NULL to auto-detect from columns named: groups, cluster, community, etc. • NULL with community specified for algorithmic detection
community	Community detection method to use for auto-grouping. If specified, overrides node_list. See detect_communities for available methods: "louvain", "walktrap", "fast_greedy", "label_prop", "infomap", "leiden".
layout	Layout type: "auto" (default), "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.

`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 result from tplot().

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,
  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	How to assign nodes to clusters. Accepts: • 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. Ignored when x is a cluster_summary.
mode	What values to display on edges: "weights" (default) Shows raw aggregated edge values. Useful when absolute magnitudes (e.g., total co-occurrences) matter. "tna" Row-normalizes the summary matrix so each row sums to 1, producing transition probabilities. Automatically enables edge_labels and summary_edge_labels unless you explicitly set them to FALSE.
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 (node_donut, curved_edges) override the preset when supplied.
layer_spacing	Vertical distance between the bottom and top layers. NULL (default) auto-calculates a gap that prevents overlap based on cluster positions and shell sizes. Increase for more vertical separation; decrease to make the plot more compact.
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 x is a cluster_summary.

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:

  `"inits"` (default) The cluster's share of the initial state distribution (`cs$macro$inits[i]`). 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 (`bw[i, i] / rowSums(bw)[i]`). 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 edge_positive_color, negative in edge_negative_color.

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 four 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.

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	`spacing`, `skew_angle`
Cluster shell appearance	`shape_size`, `shell_alpha`, `shell_border_width`, `colors`
Detail nodes	`node_size`, `node_shape`, `node_border_color`
Detail labels	`show_labels`, `label_size`, `label_abbrev`, `label_color`, `label_position`
Summary nodes	`summary_size`, `summary_border_color`, `summary_border_width`
Summary labels	`summary_labels`, `summary_label_size`, `summary_label_color`, `summary_label_positior`
Within-cluster edges	`edge_width_range`, `edge_alpha`, `edge_labels`
Between-cluster edges	`between_edge_width_range`, `between_edge_alpha`
Summary edges	`summary_edge_width_range`, `summary_edge_alpha`, `summary_edge_labels`, `summary_arrc`
Directed vs undirected	`directed`
Inter-layer lines	`inter_layer_alpha`
Top-layer layout	`top_layer_scale`, `inter_layer_gap`
Title / legend	`title`, `subtitle`, `legend`, `legend_position`

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

```
mat <- matrix(runif(36), 6, 6); diag(mat) <- 0
colnames(mat) <- rownames(mat) <- LETTERS[1:6]
clusters <- list(C1 = c("A","B"), C2 = c("C","D"), C3 = c("E","F"))
plot_mcml(mat, clusters)
```

```
cs <- csum(mat, clusters)
plot_mcml(cs, mode = "tna", edge_labels = TRUE)
```

`plot_mixed_network` *Plot Mixed Network*

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 `abs(value) < threshold` 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 splot().

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", "walktrap", "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 = 2.5,
  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,
  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. Default 2.5.
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.
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 *Plot a motif/subgraph result*

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

`x`	A `cograph_motif_result` object from `motifs()` or `subgraphs()`.
`type`	Plot type:
	`"triads"` Network diagrams of specific node triples (instance mode) or falls back to patterns (census mode). 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.
...	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

x	A tna object, weight matrix, or cograph_network.
cluster_list	Clusters can be specified as: • A list of character vectors (node names per cluster) • A string column name from nodes data (e.g., "groups") • NULL with `community` specified for auto-detection • NULL with a cograph_network that has a common cluster/group column
community	Community detection method to use for auto-clustering. If specified, overrides `cluster_list`. See `detect_communities` for available methods.
layout	How to arrange the clusters: "circle" (default), "grid", "horizontal", "vertical".
spacing	Distance between cluster centers. Default 4.
shape_size	Size of each cluster shape (shell radius). Default 1.8.
node_spacing	Radius for node placement within shapes (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(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 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_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 `shapes` to control cluster shell shapes.
...	Additional parameters passed to plot_tna().

Value

Invisibly returns a cluster_summary object for summary mode, or the plot_tna result otherwise.

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 graphics::par(mfrow=...). Set to FALSE to draw each panel into the active device without altering par(), e.g. when laying panels out yourself with panel_layout().
...	Additional arguments passed to splot().

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")
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 *Plot a Multilevel Nestimate netobject*

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

x	A netobject_ml object with $between and $within networks.
layout	Character: layout algorithm. Default "oval" (deterministic).
common_scale	Logical: use the same maximum weight for both panels? Default TRUE.
titles	Character vector of length 2: panel titles. Default c("Between-person", "Within-person").

combined	Logical: when TRUE (default), draws both panels in an internal 1 x 2 grid. Set to FALSE to render into a layout the caller already configured (e.g. via panel_layout()).
...	Additional arguments passed to splot().

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 graphics::par(mfrow=...). Set to FALSE to draw into a layout the caller has already configured (e.g. via panel_layout()).
...	Additional arguments passed to splot.

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 graphics::par(mfrow=...). Set to FALSE to draw into a layout the caller already configured (e.g. via panel_layout()).
...	Additional arguments passed to splot.net_bootstrap (e.g. display = "significant", show_stars = FALSE).

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))
grp <- Nestimate::cluster_network(seqs, k = 2)
gbs <- Nestimate::bootstrap_network(grp, iter = 10)
plot_net_bootstrap_group(gbs)
```

---

plot_net_stability *Plot Centrality Stability Results*

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 *Plot Network Robustness*

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)
}
```

---

plot_simplicial *Simplicial Complex Visualization*

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,
  ...
)
```

Arguments

x	A network object: tna, netobject, matrix, igraph, cograph_network, net_hon, or net_hypa. 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.

`method` Pathway source when auto-building from a `tna`/`netobject`: `"hon"` (default, higher-order network), `"hypa"` (anomalous paths via hypergeometric null), or `"rules"` (association-rule itemsets via `Nestimate::association_rules`; rules are rendered as single-colored blobs because itemsets are undirected).

`max_pathways` Maximum number of pathways to display. HON pathways are ranked by count, HYPA by anomaly ratio. `NULL` shows all. Default `10`.

`pathway_index` Optional positive integer vector selecting ranked pathways after extraction and ranking, before `max_pathways` is applied. For example, `2` plots the second-ranked pathway and `2:4` plots pathways ranked second through fourth.

`anomaly` HYPA anomaly type to display when plotting a `net_hypa` object or auto-building HYPA pathways via `method = "hypa"`. One of `"all"`, `"over"`, or `"under"`. Default `"all"`. Ignored (with a warning) for non-HYPA inputs such as `net_hon`, `net_association_rules`, `net_link_prediction`, character pathway vectors, or `method = "hon"` / `"rules"`, which have no anomaly concept.

`layout` `"circle"` (default) or a coordinate matrix.

`labels` Display labels. `NULL` uses state names.

`node_color` Source node fill color.

`target_color` Target node fill color.

`ring_color` Donut ring color.

`node_size` Node point size.

`label_size` Label text size.

`label_color` Label text color (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 `target_label_color` overrides for targets.

`target_label_color`
Target-node label color. `NULL` (default) reuses `label_color`.

`label_halo` Logical. Draw a contrasting halo behind each label so it stays readable on any fill — node disc, blob, or the white canvas. Default `TRUE`. The halo is the only reliable way to keep, e.g., white labels legible when `node_color` is also light.

`label_halo_color`
Halo color. `NULL` (default) auto-picks black or white based on the luminance of `label_color`, so a white label gets a dark halo and vice versa.

`label_halo_width`
Halo thickness in plot units. Default `0.035`; raise for chunkier outlines, lower for subtler ones, or set to `0` to disable without touching `label_halo`.

`label_halo_alpha`
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.
...	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

A ggplot object (or combined grid if dismantled), invisibly.

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 *Temporal Network Prism (3D Glass Box)*

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",
  node_shape = 21,
  node_border = "gray30",
  edge_color = "#E41A1C",
  edge_width = 1.5,
  edge_alpha = 0.35,
  plane_color = "gray92",
  plane_alpha = 0.2,
  plane_border = "gray60",
  plane_lty = 2,
  box = TRUE,
  box_color = "gray40",
  connections = FALSE,
  connection_color = "gray50",
  connection_alpha = 0.15,
  minimum = 0,
  show_labels = FALSE,
  label_size = 0.4,
  title = NULL,
  angle = c(1, 0.7),
  seed = 42,
  ...
)
```

Arguments

x	An edge list data frame with columns from, to, and a time column, OR a cograph_network (reads time from stored data), OR a named list of network objects.
time	Character. Name of the time column.
slices	Integer or NULL. Number of equal-width time bins. Default NULL uses unique time values.
cumulative	Logical. If TRUE, edges accumulate. Default FALSE.
labels	Character vector of layer labels. Default auto.
layout	Character or matrix. Character values currently use a shared Fruchterman-Reingold/spring layout; a matrix supplies shared coordinates. Default "spring".
node_size	Numeric. Node size. Default 2.5.
node_color	Character or vector. Node fill color. A single color applies to all layers, or a vector of length n_layers for per-layer colors. Default "steelblue".
node_shape	Integer. Point shape (pch). Default 21 (filled circle).
node_border	Character. Node border color. Default "gray30".
edge_color	Character or vector. Edge color (single or per-layer). Default "#E41A1C".
edge_width	Numeric. Base edge width. Actual width scales by weight. Default 1.5.

edge_alpha	Numeric. Edge transparency (0-1). Default 0.35.
plane_color	Character or vector. Plane fill color (single or per-layer). Default "gray92".
plane_alpha	Numeric. Plane fill transparency (0-1). Default 0.2.
plane_border	Character. Plane border color. Default "gray60".
plane_lty	Integer. Plane border line type. Default 2 (dashed).
box	Logical. Draw 3D bounding box. Default TRUE.
box_color	Character. Box edge color. Default "gray40".
connections	Logical. Draw lines connecting same nodes across planes. Default FALSE.
connection_color	Character. Default "gray50".
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.)
mar	Plot margins (numeric vector of length 4)
cut	Edge emphasis threshold
edge.label.position	Position of edge labels along edge (0-1)
edge.label.cex	Edge label size multiplier
edge.color	Edge colors
vsize	Node size
pie	Pie/donut fill values (e.g., initial probabilities)
pieColor	Pie/donut segment colors
lty	Line type for edges (1=solid, 2=dashed, 3=dotted)
directed	Logical, is the graph directed?
minimum	Minimum edge weight to display
posCol	Color for positive edges
negCol	Color for negative edges
arrowAngle	Arrow head angle in radians. Default pi/6 (30 degrees).
title	Plot title
...	Additional arguments passed to splot()

Value

Invisibly returns the cograph_network object from splot().

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 = rainbow(5), vsize = 10)

m <- matrix(runif(25), 5, 5)
tplot(m)
```

---

plot_trajectories *Plot Individual Trajectories*

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

x	Data frame with one column per time point and one row per individual trajectory.
from_title	Title for the left column. Default "From". For multi-step, use a vector of titles (e.g., c("T1", "T2", "T3", "T4")).
title	Optional plot title. Applied via ggplot2::labs(title = title).
from_colors	Colors for left-side nodes. Default uses palette.
flow_color_by	Color trajectory lines by state. Supports "source", "destination", "first", "last", or NULL. Default "first".
node_width	Width of node rectangles (0-1 scale). Default 0.08.
node_border	Border color for nodes. Default NA (no border).
node_spacing	Vertical spacing between nodes (0-1 scale). Default 0.02.
label_size	Size of node labels. Default 3.5.
label_position	Position of node labels: "beside" (default), "inside", "above", "below", "outside". Applied to first and last columns. See mid_label_position for middle columns.
mid_label_position	Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as label_position. Default NULL uses label_position 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.

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 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", "outside". Applied to first and last columns. See `mid_label_position` for middle columns.

`mid_label_position`
: Position of labels for intermediate (middle) columns in individual-tracking plots. Same options as `label_position`. Default NULL uses `label_position` 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 *Print cograph_network Object*

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 *Project Bipartite Network to One-Mode*

Description

Projects a two-mode (bipartite/incidence) network into a one-mode adjacency matrix. Row-mode projection yields a matrix of shared-column connections among row nodes; column-mode projection does the converse.

Usage

```
project_bipartite(x, mode = "rows", method = "sum", ...)
```

Arguments

x	An incidence matrix (rows = type 1 nodes, columns = type 2 nodes) where non-zero entries indicate connections. Can also be a data.frame with columns type1, type2, and optionally weight.
mode	Character. "rows" (default) projects onto row nodes (result: n_rows x n_rows). "columns" projects onto column nodes (result: n_cols x n_cols).
method	Character. Projection method: "sum" Weighted projection: A %*% t(A) (rows) or t(A) %*% A (columns). Edge weight equals sum of shared connection-weight products. "binary" Co-occurrence count: binarize A first, then compute overlap. Edge weight equals number of shared connections. "jaccard" Jaccard similarity: shared / (total_i + total_j - shared) for each pair. "cosine" Cosine similarity: dot product of row (or column) vectors divided by the product of their norms. "newman" Newman's weighted projection (Newman 2001): each shared affiliation contributes 1 / (d_k - 1) where d_k is the degree of the shared node. Gives more weight to connections through exclusive affiliations.
...	Additional arguments (currently unused).

Details

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 with row and column names preserved from the input. 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 CographNetwork object and return a matrix with x, y columns.

Value

Invisible NULL.

Examples

```
# Register a simple random layout
register_layout("random", function(network, ...) {
  n <- network$n_nodes
  cbind(x = runif(n), y = runif(n))
})
```

`register_shape` *Register a Custom Shape*

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
register_shape("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 *Register Custom SVG Shape*

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 *Register a Custom Theme*

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"
))
```

`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, 0, 1, 1, 1, 0), nrow = 3)
p <- sn_ggplot(adj)
```

`render-grid` *Grid Rendering*

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)
```

`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

`x`	Network input: matrix, igraph, network, cograph_network, or tna object.
`rich`	Character. Prominence definition: `"k"` (degree, default) or `"s"` (strength / weighted degree).
`weighted`	Logical. If TRUE (default), compute the weighted rich club coefficient. If FALSE, compute the unweighted version (density among rich nodes).
`normalized`	Logical. If TRUE (default), normalize against degree-preserving random graphs and include confidence intervals.
`n_random`	Integer. Number of random graphs for normalization. Default 100.
`directed`	Logical or NULL. Default NULL (auto-detect).
`seed`	Integer or NULL. Random seed for reproducibility. Default NULL.
`digits`	Integer or NULL. Round numeric output. Default NULL.
`...`	Additional arguments passed to `to_igraph`.

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"` and columns: `threshold`, `n_rich`, `phi`, and if normalized: `phi_norm`, `phi_rand`, `ci_lo`, `ci_hi`.

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 *Local Rich Club Score*

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".
...	Additional arguments passed to to_igraph.

Details

For each node i: $r_i = \bar{w}_{i \to rich} / \bar{w}_i$

Value

A data frame with columns node and score, sorted by score descending. Values > 1 indicate the node directs disproportionately strong ties to prominent nodes.

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"
strategy	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.
n_iter	Integer; number of iterations for random analysis. Default: 1000 (matching brainGraph convention)
mode	For directed networks: "all", "in", or "out". Default "all".
seed	Random seed for reproducibility. Default NULL.
...	Additional arguments passed to to_igraph

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 columns:

removed_pct Fraction of vertices/edges removed (0 to 1)

comp_size Size of largest component after removal

comp_pct Ratio of component size to original maximum

measure Attack strategy used

type Type of analysis (vertex or edge removal)

References

Albert, R., Jeong, H., & Barabasi, A.L. (2000). Error and attack tolerance of complex networks. *Nature*, 406, 378-381. doi:10.1038/35019019

See Also

plot_robustness, robustness_auc

Examples

```
# 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 *Calculate Area Under Robustness Curve (AUC)*

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 *Summary of Robustness Analysis*

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

Data frame with AUC and critical points for each measure.

Examples

```
g <- igraph::sample_pa(30, m = 2, directed = FALSE)
robustness_summary(x = g, measures = c("degree", "random"), n_iter = 10)
```

`select_bridges` *Select Bridge Edges*

Description

Select edges whose removal would disconnect the graph.

Usage

```
select_bridges(
  x,
  ...,
  .keep_isolates = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`...`	Additional filter expressions.
`.keep_isolates`	Keep nodes with no edges? Default FALSE.
`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` *Select Connected Component*

Description

Select nodes belonging to a specific connected component.

Usage

```
select_component(
  x,
  which = "largest",
  ...,
  .keep_edges = c("internal", "none"),
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

`x`	Network input.
`which`	Component selection: `"largest"` (default) The largest connected component **Integer** Component by ID **Character** Component containing the named node
`...`	Additional filter expressions to apply after component selection.
`.keep_edges`	How to handle edges. Default "internal".
`keep_format`	Logical. Keep input format? Default FALSE.
`directed`	Logical or NULL. Auto-detect if NULL.

Value

A cograph_network with nodes in the selected component.

See Also

`select_nodes`, `select_neighbors`

Examples

```
# 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 = FALSE,
  keep_format = FALSE,
  directed = 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, is_bridge, is_mutual, same_community, from_label, to_label

top	Integer. Select top N edges by a metric.
by	Character. Metric for top selection. Default "weight". Options: "weight", "abs_weight", "edge_betweenness".
involving	Character or integer. Select edges involving these nodes (by name or index). An edge is selected if either endpoint matches.
between	List of two character/integer vectors. Select edges between two node sets. Example: between = list(c("A", "B"), c("C", "D")).
bridges_only	Logical. Select only bridge edges (edges whose removal disconnects the graph). Default FALSE.
mutual_only	Logical. For directed networks, select only mutual (reciprocated) edges. Default FALSE.
community	Character. Community detection method for same_community variable. One of "louvain", "walktrap", "fast_greedy", "label_prop", "infomap", "leiden". Default "louvain".
.keep_isolates	Logical. Keep nodes with no remaining edges? Default FALSE.
keep_format	Logical. If TRUE, matrix, igraph, and statnet network inputs are returned in that format. Default FALSE returns cograph_network.
directed	Logical or NULL. If NULL (default), auto-detect.

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.

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 *Select Edges Between Node Sets*

Description

Select edges connecting two specified node sets.

Usage

```
select_edges_between(
  x,
  set1,
  set2,
  ...,
  .keep_isolates = FALSE,
  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 with no edges? Default FALSE.
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 = FALSE,
  keep_format = FALSE,
  directed = NULL
)
```

Arguments

x	Network input.
nodes	Character or integer. Node names or indices.
...	Additional filter expressions.
.keep_isolates	Keep nodes with no edges? Default FALSE.
keep_format	Keep input format? Default FALSE.
directed	Auto-detect if NULL.

Value

A cograph_network with edges involving the specified nodes.

See Also

select_edges, select_edges_between

Examples

```
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_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
)
```

Arguments

x	Network input: cograph_network, matrix, igraph, network, or tna object.

`...` Filter expressions using node columns, centrality measures, or global context variables. Centrality measures are computed lazily (only those actually referenced). Available variables:

Node columns All columns 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`

Global context `component`, `component_size`, `is_largest_component`, `neighborhood_size`, `k_core`, `is_articulation`, `is_bridge_endpoint`

`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 cograph_network.

`directed` Logical or NULL. If NULL (default), auto-detect.

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` and `closeness` will return NA with a warning (igraph cannot compute these with negative weights).

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. One of: "degree", "indegree", "outdegree", "strength", "instrength", "outstrength", "betweenness", "closeness", "eigenvector", "pagerank", "hub", "authority", "coreness". 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 *Select Top N Edges*

Description

Select the top N edges ranked by weight or another metric.

Usage

```
select_top_edges(
  x,
  n,
  by = "weight",
  ...,
  .keep_isolates = FALSE,
  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", "abs_weight", "edge_betweenness". Default "weight".
...	Additional filter expressions.

.keep_isolates	Keep nodes with no edges? Default FALSE.
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 *Set Edges in Cograph Network*

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

x	Network input: matrix, igraph, network, cograph_network, or tna object
from	Character or numeric node identifier(s) for the source. If NULL (default), compute distances from all nodes.
to	Character or numeric node identifier(s) for the target. If NULL (default), compute distances to all nodes.
weights	Edge weight handling: NULL (default) auto-detects from edge attributes, NA forces unweighted distances, or a numeric vector of custom weights.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.
...	Additional arguments passed to to_igraph

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.

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 *Simmelian Strength (Triangle Count per Edge)*

Description

Convenience wrapper around edge_centrality that returns only the triangle count per edge, sorted descending.

Usage

```
simmelian_strength(x, top = NULL, directed = NULL, digits = NULL, ...)
```

Arguments

x	Network input: matrix, igraph, network, cograph_network, or tna object.
top	Integer or NULL. Return only the top N edges. Default NULL.
directed	Logical or NULL. Default NULL (auto-detect).
digits	Integer or NULL. Round numeric columns. Default NULL.
...	Additional arguments passed to edge_centrality.

Value

A data frame sorted by triangles (descending) with columns: from, to, weight (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

x	Network input (matrix, cograph_network, igraph, tna object).
remove_loops	Logical. Remove self-loops (diagonal entries)?
remove_multiple	Logical. Merge duplicate edges? No-op for matrix/tna inputs (see Details).
edge_attr_comb	How to combine weights of duplicate edges: "sum", "mean", "max", "min", "first", or a custom function. Ignored for matrix/tna inputs.
...	Additional arguments (currently unused).

Details

The extent of simplification depends on the input representation:

- matrix and tna: edges are stored as an n x n weight matrix. Each cell (i, j) is unique by construction, so duplicate-edge merging 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 as the input.

See Also

filter_edges for conditional edge removal, centrality which has its own simplify parameter

Examples

```
# 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")
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)
simplify(net)
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 or a CographLayout object.
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 cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
size	Node size. Can be a single value, vector (per-node), or column name.
shape	Node shape. Options: "circle", "square", "triangle", "diamond", "pentagon", "hexagon", "ellipse", "heart", "star", "pie", "donut", "cross", "rectangle", or any custom SVG shape registered with register_svg_shape().
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: "lightgray" 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 "circle".

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 extension.
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 cograph_network object.
filename	Output filename.
width	Width in inches.
height	Height in inches.
dpi	Resolution for raster formats.
title	Optional plot title.
...	Additional arguments passed to ggsave.

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 *Apply Theme to Network*

Description

Apply a visual theme to the network.

Usage

```
sn_theme(network, theme, ...)
```

Arguments

network	A cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
theme	Theme name (string) or CographTheme object.
...	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 cograph_network object, matrix, data.frame, or igraph object. Matrices and other inputs are auto-converted.
title	Optional plot title.
title_size	Title font size.
margins	Plot margins as c(bottom, left, top, right).
layout_margin	Margin around the network layout (proportion of viewport). Default 0.15.
newpage	Logical. Start a new graphics page? Default TRUE.
background	Background color for the plot. Default "white".
layout	Layout algorithm. Built-in: "circle", "spring", "groups", "grid", "random", "star", "bipartite". igraph (2-letter): "kk" (Kamada-Kawai), "fr" (Fruchterman-Reingold), "drl", "mds", "ni" (nicely), "tr" (tree), etc. Can also pass a coordinate matrix or igraph layout function directly.
theme	Theme name: "classic", "dark", "minimal", etc.

seed	Random seed for deterministic layouts. Default 42. Set NULL for random.
labels	Node labels. Can be a character vector to set custom labels.
threshold	Minimum absolute edge weight to display. Edges with abs(weight) < threshold are hidden. Similar to qgraph's threshold.
maximum	Maximum edge weight for width scaling. Weights above this are capped. Similar to qgraph's maximum parameter.
node_size	Node size.
node_shape	Node shape: "circle", "square", "triangle", "diamond", "ellipse", "heart", "star", "pie", "donut", "cross".
node_fill	Node fill color.
node_border_color	Node border color.
node_border_width	Node border width.
node_alpha	Node transparency (0-1).
label_size	Node label text size.
label_color	Node label text color.
label_position	Label position: "center", "above", "below", "left", "right".
show_labels	Logical. Show node labels?
pie_values	For pie/donut/donut_pie nodes: list or matrix of values for segments. For donut with single value (0-1), shows that proportion filled.
pie_colors	For pie/donut/donut_pie nodes: colors for pie segments.
pie_border_width	Border width for pie chart segments.
donut_values	For donut_pie nodes: vector of values (0-1) for outer ring proportion.
donut_border_width	Border width for donut ring.
donut_inner_ratio	For donut nodes: inner radius ratio (0-1). Default 0.5.
donut_bg_color	For donut nodes: background color for unfilled portion.
donut_show_value	For donut nodes: show value in center? Default FALSE.
donut_value_size	For donut nodes: font size for center value.
donut_value_color	For donut nodes: color for center value text.
donut_fill	Numeric value (0-1) for donut fill proportion. This is the simplified API for creating donut charts. Can be a single value or vector per node.
donut_color	Fill color(s) for the donut ring. Simplified API: single color for fill, or c(fill, background) for both.
donut_colors	Deprecated. Use donut_color instead.

donut_shape Base shape for donut: "circle", "square", "hexagon", "triangle", "diamond", "pentagon". 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.

edge_negative_color
: Color for negative edge weights.

negative_color
: Deprecated. Use edge_negative_color instead.

edge_duplicates
: How to handle duplicate edges in undirected networks. NULL (default) = stop with error listing duplicates. Options: "sum", "mean", "first", "max", "min", or a custom aggregation function.

edge_labels
: Edge labels. Can be TRUE to show weights, or a vector.

edge_label_size
: Edge label text size.

edge_label_color
: Edge label text color.

edge_label_position
: Position along edge (0 = source, 0.5 = middle, 1 = target).

edge_label_offset
: Perpendicular offset from edge line.

edge_label_bg
: Background color for edge labels (default "white"). Set to NA for transparent.

edge_label_fontface
: Font face: "plain", "bold", "italic", "bold.italic".

edge_label_border
: Border style: NULL, "rect", "rounded", "circle".

edge_label_border_color
: Border color for label border.

edge_label_underline
: Logical. Underline the label text?

bidirectional
: Logical. Show arrows at both ends of edges?

loop_rotation
: Angle in radians for self-loop direction (default: pi/2 = top).

curve_shape
: Spline tension for curved edges (-1 to 1, default: 0).

curve_pivot
: Pivot position along edge for curve control point (0-1, default: 0.5).

curves
: Curve mode: TRUE (default) = single edges straight, reciprocal edges curve as ellipse (two opposing curves); FALSE = all straight; "force" = all curved.

node_names
: Alternative names for legend (separate from display labels).

legend
: Logical. Show legend?

legend_position
: Legend position: "topright", "topleft", "bottomright", "bottomleft".

scaling
: Scaling mode: "default" for qgraph-matched scaling where node_size=6 looks similar to qgraph vsize=6, or "legacy" to preserve pre-v2.0 behavior.

weight_digits
: 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) |> soplot()

# Direct matrix input with all options
adj |> soplot(
  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)
```

---

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

x	A group_tna_permutation object (from tna::permutation_test on group_tna).
...	Additional arguments passed to plot_permutation().
i	Index or name of specific comparison to plot. NULL for all.
combined	Logical: when TRUE (default), lay out panels in an internal grid via `graphics::par(mfrow=...)`. Set to FALSE to draw each panel into a layout the caller has already configured (e.g. via `panel_layout()`). Ignored when `i` selects a single panel.

Value

When `i` is supplied, returns the selected permutation plot. 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, coloured 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×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_size = NULL,
  label_color = "black",
  label_position = "center",
  label_fontface = "plain",
  label_fontfamily = "sans",
  label_hjust = 0.5,
  label_vjust = 0.5,
  label_angle = 0,
  pie_values = NULL,
  pie_colors = NULL,
  pie_border_width = NULL,
  donut_fill = NULL,
  donut_values = NULL,
  donut_color = NULL,
  donut_colors = NULL,
  donut_border_color = NULL,
  donut_border_width = NULL,
  donut_inner_border_color = NULL,
  donut_inner_border_width = NULL,
  donut_outer_border_color = NULL,
  donut_line_type = "solid",
  donut_border_lty = NULL,
  donut_inner_ratio = 0.8,
  donut_bg_color = "gray90",
  donut_shape = "circle",
  donut_show_value = FALSE,
  donut_value_size = 0.8,
  donut_value_color = "black",
  donut_value_fontface = "bold",
  donut_value_fontfamily = "sans",
  donut_value_digits = 2,
  donut_value_prefix = "",
  donut_value_suffix = "",
  donut_empty = TRUE,
  donut2_values = NULL,
```

```
  donut2_colors = NULL,
  donut2_inner_ratio = 0.4,
  edge_color = NULL,
  edge_width = NULL,
  edge_size = NULL,
  esize = NULL,
  edge_width_range = c(0.1, 4),
  edge_scale_mode = "linear",
  edge_cutoff = NULL,
  cut = NULL,
  edge_alpha = 0.8,
  edge_labels = FALSE,
  edge_label_size = 0.8,
  edge_label_color = "gray30",
  edge_label_bg = NA,
  edge_label_position = 0.5,
  edge_label_offset = 0,
  edge_label_fontface = "plain",
  edge_label_shadow = FALSE,
  edge_label_shadow_color = "gray40",
  edge_label_shadow_offset = 0.5,
  edge_label_shadow_alpha = 0.5,
  edge_label_halo = TRUE,
  edge_style = 1,
  curvature = 0,
  curve_scale = TRUE,
  curve_shape = 0,
  curve_pivot = 0.5,
  curves = TRUE,
  arrow_size = 1,
  arrow_angle = pi/6,
  show_arrows = TRUE,
  bidirectional = FALSE,
  loop_rotation = NULL,
  show = NULL,
  edge_start_style = "solid",
  edge_start_length = 0.15,
  edge_start_dot_density = "12",
  edge_ci = NULL,
  edge_ci_scale = 2,
  edge_ci_alpha = 0.15,
  edge_ci_color = NA,
  edge_ci_style = 2,
  edge_ci_arrows = FALSE,
  edge_priority = NULL,
  edge_label_style = "none",
  edge_label_template = NULL,
  edge_label_digits = 2,
```

```
edge_label_oneline = TRUE,
edge_label_ci_format = "bracket",
edge_label_leading_zero = TRUE,
edge_ci_lower = NULL,
edge_ci_upper = NULL,
edge_label_p = NULL,
edge_label_p_diff = NULL,
edge_label_p_digits = 3,
edge_label_p_prefix = "p=",
edge_label_stars = NULL,
weight_digits = 2,
threshold = 0,
minimum = 0,
maximum = NULL,
edge_positive_color = "#2E7D32",
positive_color = NULL,
edge_negative_color = "#C62828",
negative_color = NULL,
edge_duplicates = NULL,
title = NULL,
title_size = 1.2,
margins = c(0.1, 0.1, 0.1, 0.1),
background = "white",
rescale = TRUE,
layout_scale = 1,
layout_margin = 0.15,
aspect = TRUE,
use_pch = FALSE,
usePCH = NULL,
scaling = "default",
align_panels = FALSE,
legend = FALSE,
legend_position = "topright",
legend_size = 0.8,
legend_edge_colors = TRUE,
legend_node_sizes = FALSE,
groups = NULL,
node_names = NULL,
tna_styling = NULL,
psych_styling = NULL,
predictability = NULL,
i = NULL,
filetype = "default",
filename = file.path(tempdir(), "splot"),
width = 7,
height = 7,
res = 600,
...
```

)

Arguments

x	Network input. Can be: • A square numeric matrix (adjacency/weight matrix) • A data frame with edge list (from, to, optional weight columns) • An igraph object • A CographNetwork or cograph_network object • A tna object (from tna package) • A group_tna object (list of tna objects from tna package). Use parameter i to select a specific group, or omit to plot all groups.
display	Display mode: "styled" (default), "significant", or "full".
show_ci	Logical: overlay CI bounds on edge labels? Default FALSE.
show_stars	Logical: show significance stars? Default TRUE.
inherit_style	Logical: inherit labels/layout/colors from network? Default TRUE.
...	Additional arguments passed to layout functions. One ride-along worth calling out: combined (default TRUE). When x is a multi-panel input (a group_tna, group_tna_bootstrap, group_tna_permutation, net_permutation_group, or any class routed to a splot.* method that draws multiple panels such as splot.net_mlvar with type = "all"), combined = FALSE skips the internal graphics::par(mfrow = ...) grid so the caller can drive layout explicitly via panel_layout() or graphics::layout(). For single-network inputs (a single tna, netobject, matrix, etc.) combined has no effect — there is no panel grid to gate.
type	Character. "overlay" (default) renders both networks on a single canvas via plot_mixed_network — co-occurrence as straight undirected edges, transitions as curved directed arrows. "group" plots each component as a separate panel.
combined	Logical: when type = "all", 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 panel_layout()). Ignored for single-network types.
use_thresholded	Logical: use $thresholded_pcor? If FALSE, uses $original_pcor. Default TRUE.
show_inclusion	Logical: scale edge alpha by inclusion probability? Default TRUE.
inclusion_threshold	Numeric: minimum inclusion probability to show an edge. Default 1 - x$alpha (i.e. the complement of the alpha level).
edge_positive_color	Color for positive weights.
edge_negative_color	Color for negative weights.
show_nonsig	Logical: show non-significant edges? Default FALSE.

show_effect Logical: show effect size in parentheses? Default FALSE.

edge_nonsig_color
Color for non-significant edges. Default "#888888".

edge_nonsig_style
Line style for non-significant edges. Default 2L.

layout Layout algorithm: "oval" (default), "circle", "spring", "groups", "target" (qgraph-style focal-node BFS levels; node of interest via target), "saqr" (Start/End transition flow; start/ end/jitter), or a matrix of x,y coordinates, or an igraph layout function. Also supports igraph two-letter codes: "kk", "fr", "drl", "mds", "ni", etc.

directed Logical. Force directed interpretation. NULL for auto-detect.

seed Random seed for deterministic layouts. Default 42.

theme Theme name: "classic", "dark", "minimal", "colorblind", etc.

node_size Node size(s). Single value or vector. Default NULL, which resolves to 7 with default scaling.

node_size2 Secondary node size for ellipse/rectangle height.

scale_nodes_by Scale node sizes by a centrality measure. Can be:

- A measure name: "degree", "strength", "betweenness", "closeness", "eigenvector", "pagerank", "authority", "hub", "harmonic", etc.
- A directional shorthand: "indegree", "outdegree", "instrength", "outstrength", "incloseness", "outcloseness", "inharmonic", "outharmonic", "ineccentricity", "outeccentricity".
- A list with measure and parameters: list("pagerank", damping = 0.9)

When used, node_size is ignored. Use node_size_range to control the min/max size. Default NULL (no centrality scaling).

node_size_range
Size range for centrality-based scaling. Numeric vector c(min_size, max_size). Default c(2, 8).

scale_nodes_scale
Dampening exponent for centrality-based sizing. Values < 1 compress differences (e.g., 0.5 applies square root), values > 1 exaggerate differences. Default 1 (linear).

node_shape Node shape(s): "circle", "square", "triangle", "diamond", "pentagon", "hexagon", "star", "heart", "ellipse", "cross", or any custom SVG shape registered with register_svg_shape().

node_svg Custom SVG for nodes: path to SVG file OR inline SVG string.

svg_preserve_aspect
Logical: maintain SVG aspect ratio? Default TRUE.

node_fill Node fill color(s).

node_border_color
Node border color(s).

node_border_width
Node border width(s).

`node_alpha` Node transparency (0-1). Default 1.

`labels` Node labels: TRUE (use node names/indices), FALSE (none), or character vector.

`label_size` Label character expansion factor.

`label_color` Label text color.

`label_position` Label position: "center", "above", "below", "left", "right".

`label_fontface` Font face for labels: "plain", "bold", "italic", "bold.italic". Default "plain".

`label_fontfamily` Font family for labels: "sans", "serif", "mono". Default "sans".

`label_hjust` Horizontal justification (0=left, 0.5=center, 1=right). Default 0.5.

`label_vjust` Vertical justification (0=bottom, 0.5=center, 1=top). Default 0.5.

`label_angle` Text rotation angle in degrees. Default 0.

`pie_values` List of numeric vectors for pie chart nodes. Each element corresponds to a node and contains values for pie segments. If a simple numeric vector with values between 0 and 1 is provided (e.g., centrality scores), it is automatically converted to donut_fill for convenience.

`pie_colors` List of color vectors for pie segments.

`pie_border_width` Border width for pie slice dividers. NULL uses node_border_width.

`donut_fill` Numeric value (0-1) for donut fill proportion. This is the qgraph-style API: 0.1 = 10% filled, 0.5 = 50% filled, 1.0 = fully filled. Can be a single value (all nodes) or vector (per-node values).

`donut_values` Deprecated. Use donut_fill for simple fill proportion.

`donut_color` Fill color(s) for the donut ring. Single color sets fill for all nodes. Two colors set fill and background for all nodes. More than 2 colors set per-node fill colors (recycled to n_nodes). Default: "maroon" fill, "gray90" background when node_shape="donut".

`donut_colors` Deprecated. Use donut_color instead.

`donut_border_color` Border color for donut rings. NULL uses node_border_color.

`donut_border_width` Border width for donut rings. NULL uses node_border_width.

`donut_inner_border_color` Color for the inner boundary (where the donut meets its hole). NULL (default) uses `donut_border_color`. Can be scalar or per-node vector.

`donut_inner_border_width` Width for the inner boundary border. NULL (default) uses `donut_border_width`. Can be scalar or per-node vector.

`donut_outer_border_color` Color for outer boundary border (enables double border). NULL (default) shows single border. Set to a color for double border effect. Can be scalar or per-node vector.

`donut_line_type`
: Line type for donut borders: "solid", "dashed", "dotted", or numeric (1=solid, 2=dashed, 3=dotted). Can be scalar or per-node vector.

`donut_border_lty`
: Deprecated. Use `donut_line_type` instead.

`donut_inner_ratio`
: Inner radius ratio for donut (0-1). Default 0.8.

`donut_bg_color`
: Background color for unfilled donut portion.

`donut_shape`
: Base shape for donut: "circle", "square", "hexagon", "triangle", "diamond", "pentagon". Can be a single value or per-node vector. Default inherits from node_shape (e.g., hexagon nodes get hexagon donuts). Set explicitly to override (e.g., donut_shape = "hexagon" for hexagon donuts on all nodes regardless of node_shape).

`donut_show_value`
: Logical: show value in donut center? Default FALSE.

`donut_value_size`
: Font size for donut center value.

`donut_value_color`
: Color for donut center value.

`donut_value_fontface`
: Font face for donut center value: "plain", "bold", "italic", "bold.italic". Default "bold".

`donut_value_fontfamily`
: Font family for donut center value: "sans", "serif", "mono". Default "sans".

`donut_value_digits`
: Decimal places for donut center value. Default 2.

`donut_value_prefix`
: Text before donut center value (e.g., "$"). Default "".

`donut_value_suffix`
: Text after donut center value (e.g., "%"). Default "".

`donut_empty`
: Logical: render empty donut rings for NA values? Default TRUE.

`donut2_values`
: List of values for inner donut ring (for double donut).

`donut2_colors`
: List of color vectors for inner donut ring segments.

`donut2_inner_ratio`
: Inner radius ratio for inner donut ring. Default 0.4.

`edge_color`
: Edge color(s). If NULL, uses edge_positive_color/edge_negative_color based on weight.

`edge_width`
: Edge width(s). If NULL, scales by weight using edge_size and edge_width_range.

`edge_size`
: Maximum edge size for weight scaling. NULL (default) uses the upper bound of `edge_width_range`. Larger values = thicker edges overall.

`esize`
: Deprecated. Use `edge_size` instead.

`edge_width_range`
: Output width range as c(min, max) for weight-based scaling. Default c(0.1, 4). Edges are scaled to fit within this range unless `edge_size` supplies the maximum.

edge_scale_mode
: Scaling mode for edge weights: "linear" (default, qgraph-style), "log" (logarithmic for wide weight ranges), "sqrt" (moderate compression), or "rank" (equal visual spacing regardless of weight distribution).

edge_cutoff
: Optional cutoff for edge emphasis. NULL (default) or 0 disables cutoff fading. Positive values fade edges whose absolute weights are below the cutoff; width scaling remains continuous.

cut
: Deprecated. Use edge_cutoff instead.

edge_alpha
: Edge transparency (0-1). Default 0.8.

edge_labels
: Edge labels: TRUE (show weights), FALSE (none), or character vector.

edge_label_size
: Edge label size.

edge_label_color
: Edge label text color.

edge_label_bg
: Edge label background color.

edge_label_position
: Position along edge (0-1).

edge_label_offset
: Perpendicular offset for edge labels (0 = on line, positive = above).

edge_label_fontface
: Font face: "plain", "bold", "italic", "bold.italic".

edge_label_shadow
: Logical: enable drop shadow for edge labels? Default FALSE.

edge_label_shadow_color
: Color for edge label shadow. Default "gray40".

edge_label_shadow_offset
: Offset distance for shadow in points. Default 0.5.

edge_label_shadow_alpha
: Transparency for shadow (0-1). Default 0.5.

edge_label_halo
: Logical: enable white halo/outline around edge labels for readability over dark edges? Default TRUE. When TRUE, overrides shadow settings.

edge_style
: Line type(s): 1=solid, 2=dashed, 3=dotted, etc.

curvature
: Edge curvature. 0 for straight, positive/negative for curves.

curve_scale
: 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., `splot.tna_disparity`). Not intended for direct use.

`edge_start_style`
Style for the start segment of edges: "solid" (default), "dashed", or "dotted". Use dashed/dotted to indicate edge direction (source node).

`edge_start_length`
Fraction of edge length for the styled start segment (0-0.5). Default 0.15 (15% of edge). Only applies when edge_start_style is not "solid".

`edge_start_dot_density`
Pattern for dotted start segments. A two-character string where the first digit is dot length and second is gap length (in line width units). Default "12" (1 unit dot, 2 units gap). Use "11" for tighter dots, "13" for more spacing. Only applies when edge_start_style = "dotted".

`edge_ci` Numeric vector of CI widths (0-1 scale). Larger values = more uncertainty.

`edge_ci_scale` Width multiplier for underlay thickness. Default 2.

`edge_ci_alpha` Transparency for underlay (0-1). Default 0.15.

`edge_ci_color` Underlay color. NA (default) uses main edge color.

`edge_ci_style` Line type for underlay: 1=solid, 2=dashed, 3=dotted. Default 2.

`edge_ci_arrows` Logical: show arrows on underlay? Default FALSE.

`edge_priority` Numeric vector of edge priorities. Higher values render on top. Useful for ensuring significant edges appear above non-significant ones.

`edge_label_style`
Preset style: "none", "estimate", "full", "range", "stars".

`edge_label_template`
Template with placeholders: {est}, {range}, {low}, {up}, {p}, {p_diff}, {stars}. Overrides edge_label_style if provided.

`edge_label_digits`
Decimal places for estimates. Default 2.

`edge_label_oneline`
Logical: single line format? Default TRUE.

`edge_label_ci_format`
CI format: "bracket" for `[low, up]` or "dash" for `low-up`.

`edge_label_leading_zero`
Logical: show leading zero for values < 1? Default TRUE. Set to FALSE to display ".5" instead of "0.5".

`edge_ci_lower` Numeric vector of lower CI bounds for labels.

`edge_ci_upper` Numeric vector of upper CI bounds for labels.

`edge_label_p` Numeric vector of p-values for edges.

`edge_label_p_diff`
Probability-of-difference values for the `{p_diff}` template placeholder: a per-edge numeric vector, or a full node-by-node matrix (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=".

`edge_label_stars`
: Stars for labels: character vector, TRUE (compute from p), or numeric (treated as p-values).

`weight_digits` Number of decimal places to round edge weights to before plotting. Edges that round to zero are automatically removed. Default 2. Set NULL to disable rounding.

`threshold` Minimum absolute weight to display.

`minimum` Alias for threshold (qgraph compatibility). Uses max of threshold and minimum.

`maximum` Maximum weight for scaling. NULL for auto.

`positive_color` Deprecated. Use `edge_positive_color` instead.

`negative_color` Deprecated. Use `edge_negative_color` instead.

`edge_duplicates`
: How to handle duplicate edges in undirected networks. NULL (default) = stop with error listing duplicates. Options: "sum", "mean", "first", "max", "min", or a custom aggregation function.

`title` Plot title.

`title_size` Title font size.

`margins` Margins as c(bottom, left, top, right).

`background` Background color.

`rescale` Logical: rescale layout to -1 to 1 range?

`layout_scale` Scale factor for layout. >1 expands (spreads nodes apart), <1 contracts (brings nodes closer). Use "auto" to automatically scale based on node count (compact for small networks, expanded for large). Default 1.

`layout_margin` Margin around the layout as fraction of range. Default 0.15. Set to 0 for no extra margin (tighter fit). Affects white space around nodes.

`aspect` Logical: maintain aspect ratio?

`use_pch` Logical: use points() for simple circles (faster). Default FALSE.

`usePCH` Deprecated. Use `use_pch` instead.

`scaling` Scaling mode: "default" for qgraph-matched scaling where node_size=6 looks similar to qgraph vsize=6, or "legacy" to preserve pre-v2.0 behavior.

`align_panels` Logical. If `TRUE`, forces a uniform symmetric plot box (`c(-layout_scale, layout_scale)` on each axis) so two networks plotted side-by-side in a `par(mfrow)` grid render at identical absolute scales — 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 behaviour) which renders tighter on the canvas. The per-node loop-reservation pad in compute_plot_limits runs regardless, so networks with different self-loop patterns stay centered consistently in either mode.

legend — Logical: show legend?

legend_position — Position: "topright", "topleft", "bottomright", "bottomleft".

legend_size — Legend text size.

legend_edge_colors — Logical: show positive/negative edge colors in legend?

legend_node_sizes — Logical: show node size scale in legend?

groups — Group assignments for node coloring/legend.

node_names — Alternative names for legend (separate from labels).

tna_styling — Logical or NULL. If TRUE, applies TNA visual defaults (oval layout, TNA color palette, edge labels as estimates, dotted edge starts, etc.) as a base layer. Any explicitly provided argument overrides the TNA default. If FALSE, no TNA styling is applied. If NULL (default), automatically set to TRUE when x is a tna object, FALSE otherwise. Can be used with any input type (matrix, igraph, cograph_network).

psych_styling — Logical or NULL. Undirected counterpart of tna_styling. If TRUE, applies psychometric-network defaults (spring layout, Okabe-Ito palette, no arrows, thin edges) as a base layer. If NULL (default), splot.netobject auto-enables it on correlation-family input (glasso, cor, pcor, ising) and on the undirected constituents of net_mlvar. Explicit user args always win.

predictability — Logical or NULL. Draws a per-node predictability ring (a donut fill) from a predictability column on the network's node table, the way qgraph/bootnet show node predictability. If TRUE, draws it when the column is present; if FALSE, never; if NULL (default), draws it when the object marks it as its default (network$meta$predictability_default, set e.g. by a psychnet glasso network). A caller's own pie_values / donut_fill takes precedence.

i — Group index or name when x is a group_tna object. If NULL (default), plots all groups in a grid. If specified (e.g., i = 1 or i = "Treatment"), plots only that group.

filetype — Output format: "default" (screen), "png", "pdf", "svg", "jpeg", "tiff".

filename — Output filename (without extension).

width — Output width in inches.

height — Output height in inches.

res — Resolution in DPI for raster outputs (PNG, JPEG, TIFF). Default 600.

Details

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 modelling 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 "wtna_mixed". Arbitrary function names are never evaluated.

weight Optional character scalar naming the default edge weight to render. If it names an edge column, that column is copied to edges$weight for the plot (the producer's edge set is kept, and the weights matrix is rebuilt to match). If it names a matrix stored on the object, that matrix becomes the rendered network: it is copied to weights and the drawn edge set is rebuilt from its nonzero cells (aligned to the object's node order via dimnames when present). This is useful when the analytical object stores several edge quantities (for example counts, probabilities, residuals, effects) but has one preferred plot view. When the name matches both an edge column and a stored matrix, the matrix form wins.

defaults Named list of splot() or renderer arguments. These are defaults only: any argument explicitly supplied by the user wins. Defaults can include regular splot() arguments such as layout, node_fill, edge_labels, weight_digits, or renderer-specific arguments such as display for bootstrap renderers.

The precedence rule is always:

```
user arguments > x$meta$splot$defaults > cograph defaults
```

Example producer-side metadata:

```
x$meta$splot <- list(
  renderer = "network",
  weight = "adj_res",
  defaults = list(
    node_fill = "white",
    edge_labels = TRUE,
    weight_digits = 1
  )
)
```

Value

Invisibly returns the plot.

Invisibly, the splot result.

Invisibly, the splot result.

Invisibly returns x.

Invisibly returns the plot.

Invisibly returns the plot.

Invisibly returns x.

Invisibly returns the plot.

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"))

# 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 *Plot Disparity Results with splot*

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

x	A tna_permutation object (from tna::permutation_test).
...	Additional arguments passed to splot().
show_nonsig	Logical: show non-significant edges? Default FALSE (only significant shown).
edge_positive_color	Color for positive differences (x > y). Default "#009900" (green).
edge_negative_color	Color for negative differences (x < y). Default "#C62828" (red).
edge_nonsig_color	Color for non-significant edges. Default "#888888" (grey).
edge_nonsig_style	Line style for non-significant edges (2=dashed). Default 2.
show_stars	Logical: show significance stars (*, **, ***) on edges? Default TRUE.
show_effect	Logical: show effect size in parentheses for significant edges? Default FALSE.
edge_nonsig_alpha	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 plot.

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 `student_interactions[student_interactions$from != student_interactions$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 igraph
el <- student_interactions[student_interactions$from != student_interactions$to, ]
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::graph_from_data_frame(el, directed = FALSE)
  cat("Students:", igraph::vcount(g), "\n")
  cat("Interactions:", igraph::ecount(g), "\n")
}
```

---

subgraphs *Extract Specific Motif Instances (Subgraphs)*

Description

Convenience wrapper for `motifs(x, named_nodes = TRUE, ...)`. Returns one row per concrete node-triple instantiating each MAN pattern, so the same MAN type can appear in many rows with its own `z` / `p` per triple. 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`, `pattern`, `include`, `exclude`, `significance`, `n_perm`, `min_count`, `edge_method`, `edge_threshold`, `min_transitions`, `top`, `seed`).

Value

A `cograph_motif_result` object with `named_nodes = TRUE`. Contains `$results` (data frame with columns `triad`, `type`, `observed`, and optionally `z`, `p`, `sig`), `$type_summary`, `$level`, `$n_units`, and `$params`. In instance mode, `$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` *Build MCML from Raw Transition Data*

Description

Builds a Multi-Cluster Multi-Level (MCML) model from raw transition data (edge lists or sequences) by recoding node labels to cluster labels and counting 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: `list(A = c("N1","N2"), B = c("N3","N4"))`

data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: `data.frame(node = c("N1","N2","N3"), group = c("A","A","B"))`

membership vector Character or numeric vector. Node names are extracted from the data. Example: `c("A","A","B","B")`

column name string For edge list data.frames, the name of a column containing cluster labels. The mapping is built from unique (node, group) pairs in both from and to columns.

NULL Auto-detect from `cograph_network$nodes` or `$node_groups` (same logic as `csum`).

`method` Aggregation method for combining edge weights: "sum", "mean", "median", "max", "min", "density", "geomean". Default "sum".

`type` Post-processing: "tna" (row-normalize), "frequency" or "raw" (no normalization), "cooccurrence" (symmetrize), or "semi_markov". 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 *Summarize Network by Clusters*

Description

Creates a summary network where each cluster becomes a single node. Edge weights are aggregated from the original network using the specified method. 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 cograph_network.
cluster_list	Cluster specification: • Named list of node vectors (e.g., list(A = c("n1", "n2"), B = c("n3", "n4"))) • 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 *Supra-Adjacency Matrix*

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 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 omega[a,b] * I. If no entry matches a pair and no legacy chain layout applies, a warning is emitted and the diagonal default omega[a,b] * I is used (previously this happened silently).

Value

Supra-adjacency matrix of dimension (N*L) x (N*L)

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 *Extract Inter-Layer Block*

Description

Extract Inter-Layer Block

Usage

```
supra_interlayer(x, from, to)

extract_interlayer(x, from, to)
```

Arguments

x	Supra-adjacency matrix
from	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)
```

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 *Dark Theme*

Description

Dark background theme for presentations.

Usage

```
theme_cograph_dark()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_dark()
```

theme_cograph_gray *Grayscale Theme*

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 *Viridis Theme*

Description

Theme using viridis color palette.

Usage

```
theme_cograph_viridis()
```

Value

A CographTheme object.

Examples

```
theme <- theme_cograph_viridis()
```

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 data frame with columns:

- from: Source node name/label
- to: Target node name/label
- weight: Edge weight

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_igraph *Convert Network to igraph Object*

Description

Converts various network representations to an igraph object. Supports matrices, edge-list data frames, igraph objects, network objects, cograph_network, and tna objects.

Usage

```
to_igraph(x, directed = NULL)
```

Arguments

x	Network input. Can be: • A square numeric matrix (adjacency/weight matrix) • A data frame edge list with source and target columns • An igraph object (returned as-is or converted if directed differs) • A statnet network object • A cograph_network object • A tna object
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

Value

An igraph object.

See Also

to_data_frame, as_cograph

Examples

```
# 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 *Convert Network to Adjacency Matrix*

Description

Converts any supported network format to an adjacency matrix.

Usage

```
to_matrix(x, directed = NULL)
```

Arguments

x	Network input: matrix, cograph_network, igraph, network, tna, etc.
directed	Logical or NULL. If NULL (default), auto-detect from input.

Value

A square numeric adjacency matrix, preserving row/column names when available.

See Also

to_igraph, to_df, as_cograph, to_network

Examples

```
# 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` *Convert Network to statnet network Object*

Description

Converts any supported network format to a statnet network object.

Usage

```
to_network(x, directed = NULL)
```

Arguments

`x`	Network input: matrix, cograph_network, igraph, tna, etc.
`directed`	Logical or NULL. If NULL (default), auto-detect from input.

Value

A network object from the network package.

See Also

`to_igraph`, `to_matrix`, `to_df`, `as_cograph`

Examples

```
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)
}
```

`triad_census` *Triad Census*

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

Named vector of triad counts

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

```
mat <- matrix(sample(0:1, 100, replace = TRUE), 10, 10)
diag(mat) <- 0
triad_census(mat)
```

`trophic_incoherence` *Trophic Incoherence Parameter*

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

x	Adjacency matrix
clusters	Cluster specification
method	Aggregation method
type	Normalization type. Defaults to "raw" for igraph compatibility.

Value

List with comparison results

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

x	Network input: matrix, igraph, network, cograph_network, or tna object.
directed	Logical or NULL. If NULL (default), auto-detect from matrix symmetry. Set TRUE to force directed, FALSE to force undirected.

`normalized`	Logical. If TRUE (default), return the proportional drop. If FALSE, return the raw efficiency difference.
`weighted`	Logical. If TRUE, honor edge weights when computing shortest paths (Dijkstra); distance is `1/weight^alpha` per the usual qgraph/tna convention when `invert_weights = TRUE`. If FALSE (default, matches prior behavior), all edges are treated as unit length.
`invert_weights`	Logical. If TRUE (default) and weights are present, invert weights to distances via `1/weight^alpha` so that higher weight = shorter path (matches `centrality()`'s default). Ignored when `weighted = FALSE`.
`alpha`	Weight-to-distance exponent (default 1).
`digits`	Integer or NULL. Round scores to this many decimal places. Default NULL (no rounding).
`...`	Additional arguments passed to `to_igraph`.

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:

$$E_{global} = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{1}{d(i,j)}$$

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 columns:

node Node labels.

vulnerability Vulnerability scores, sorted descending.

The original input network and normalization mode are stored as attributes.

References

Latora, V. & Marchiori, M. (2007). A measure of centrality based on network efficiency. *New Journal of Physics*, 9(6), 188. doi:10.1088/13672630/9/6/188

See Also

network_global_efficiency, robustness, centrality

Examples

```
# 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