# Package ‘cograph’

March 2, 2026

**Title** Modern Network Visualization

**Version** 1.5.2

**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/`

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

**Depends** R (>= 4.4.0)

**Imports** R6, grid, grDevices, ggplot2, stats, utils

**Suggests** igraph, network, scales, testthat (>= 3.0.0), knitr,
rmarkdown, digest, grImport2, rsvg, qgraph, tna, RColorBrewer,
colorspace

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Mohammed Saqr [aut],
Sonsoles López-Pernas [aut, cre],
Santtu Tikka [aut]

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

**Repository** CRAN

**Date/Publication** 2026-03-02 21:40:08 UTC

# Contents

---

`aes-edges` *Edge Aesthetics*

---

## Description

Functions for setting edge aesthetic properties.

---

`aes-nodes` *Node Aesthetics*

---

## Description

Functions for setting node aesthetic properties.

## 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, ...)
```

### Arguments

| | |
|---|---|
| `x` | Network input. Can be:<br>• A square numeric matrix (adjacency/weight matrix)<br>• A data frame with edge list (from, to, optional weight columns)<br>• An igraph object<br>• A statnet network object<br>• A qgraph object<br>• A tna object<br>• An existing cograph_network object (returned as-is) |
| `directed` | Logical. Force directed interpretation. NULL for auto-detect. |
| `...` | Additional arguments (currently unused). |

### Details

The cograph_network format is designed to be:

- Simple: All data accessible via `net$from`, `net$to`, `net$weight`, etc.
- Modern: Uses named list elements instead of attributes for clean `$` access
- Compatible: Works seamlessly with splot() and other cograph functions

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

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

### Value

A cograph_network object: a named list with components:

`from` Integer vector of source node indices

`to` Integer vector of target node indices

`weight` Numeric vector of edge weights

`nodes` Data frame with id, label, (x, y if layout applied) `directed` Logical indicating if network is directed

`n_nodes` Integer count of nodes

`n_edges` Integer count of edges

`labels` Character vector of node labels

`source` Character indicating input type

`layout` Layout coordinates (NULL until computed)

`layout_info` Layout algorithm info (NULL until computed)

## 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

```
# From adjacency matrix
mat <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
net <- as_cograph(mat)

# Direct $ access to all data
net$from       # edge sources
net$to         # edge targets
net$weight     # edge weights
net$nodes      # nodes data frame
net$directed   # TRUE/FALSE
net$n_nodes    # 3
net$n_edges    # 3

# Getter functions (recommended for programmatic use)
get_nodes(net)   # nodes data frame
get_edges(net)   # edges data frame (from, to, weight)
get_labels(net)  # character vector of labels
n_nodes(net)     # 3
n_edges(net)     # 3
is_directed(net) # FALSE (symmetric matrix)

# Setter functions
net <- set_nodes(net, data.frame(id = 1:3, label = c("A", "B", "C")))
net <- set_edges(net, data.frame(from = c(1,2), to = c(2,3), weight = c(0.5, 0.8)))
net <- set_layout(net, data.frame(x = c(0, 1, 0.5), y = c(0, 0, 1)))

# Plot it
splot(net)

# From igraph (if installed)
if (requireNamespace("igraph", quietly = TRUE)) {
  library(igraph)
  g <- make_ring(10)
  net <- as_cograph(g)
  splot(net)
}
```

## 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 = "spring",
  directed = NULL,
  node_labels = NULL,
  seed = 42,
  ...
)
```

### Arguments

| | |
|---|---|
| `input` | Network input. Can be: <br>• A square numeric matrix (adjacency/weight matrix)<br>• A data frame with edge list (from, to, optional weight columns)<br>• An igraph object<br>• A statnet network object<br>• A qgraph object<br>• A tna object |
| `layout` | Layout algorithm: "circle", "spring", "groups", "grid", "random", "star", "bipartite", or "custom". Default "spring". |
| `directed` | Logical. Force directed interpretation. NULL for auto-detect. |
| `node_labels` | Character vector of node labels. |
| `seed` | Random seed for deterministic layouts. Default 42. Set NULL for random. |
| `...` | 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
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj)

# From edge list
edges <- data.frame(from = c(1, 1, 2), to = c(2, 3, 3))
cograph(edges)

# With customization (pipe-friendly workflow)
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
cograph(adj, layout = "circle") |>
  sn_nodes(fill = "steelblue") |>
  sn_edges(color = "gray50") |>
  splot()

# Weighted network with automatic styling
w_adj <- matrix(c(0, 0.5, -0.3, 0.5, 0, 0.4, -0.3, 0.4, 0), nrow = 3)
cograph(w_adj) |>
  sn_edges(color = "weight", width = "weight") |>
  splot()

# With igraph (if installed)
if (requireNamespace("igraph", quietly = TRUE)) {
  library(igraph)
  g <- make_ring(10)
  cograph(g) |> splot()
}
```

---

`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,
  ...
)
```

## 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. |
| `...` | 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)
- `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
if (requireNamespace("qgraph", quietly = TRUE)) {
  library(qgraph)
  adj <- matrix(c(0, .5, .3, .5, 0, .4, .3, .4, 0), 3, 3)
  q <- 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(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 = 2,
  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.

TNA networks are always treated as directed because transition matrices represent directional state changes.

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
- `edge_color` = `"#003355"`: Dark blue edges
- `edge_labels` = `TRUE`: Show transition weights on edges
- `edge_label_size` = `0.6`: Readable edge labels
- `edge_label_position` = `0.7`: Labels positioned toward target
- `edge_start_style` = `"dotted"`: Dotted line at edge source
- `edge_start_length` = `0.2`: 20% of edge is 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
if (requireNamespace("tna", quietly = TRUE)) {
  library(tna)
  trans <- tna(group_regulation)
  from_tna(trans)  # Plots with donut rings showing initial probabilities

  # Use soplot engine instead
  from_tna(trans, engine = "soplot")

  # Customize the visualization
  from_tna(trans, layout = "circle", donut_color = c("steelblue", "gray90"))

  # Extract parameters without plotting
  params <- from_tna(trans, plot = FALSE)
  # Modify and plot manually
  params$node_fill <- "coral"
  do.call(splot, params)
}
```

---

## `get_edges` *Get Edges from Cograph Network*

---

## Description

Extracts the edges data frame from a cograph_network object. For the new format, builds a data frame from the from/to/weight vectors.

### 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_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_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 data frame with columns: id, label, name, x, y (and possibly others).

### 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_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")
```

---

`layout-groups` *Group-based Layout*

---

### Description

Arrange nodes in groups, with each group in a circular arrangement.

---

`layout-oval` *Oval/Ellipse Layout*

---

### Description

Arrange nodes in an oval (ellipse) shape.

---

`layout-spring` *Fruchterman-Reingold Spring Layout*

---

### Description

Force-directed layout using the Fruchterman-Reingold algorithm.

---

`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 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. |

## 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 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 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. |

### 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_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 = 500,
  cooling = 0.95,
  repulsion = 1,
  attraction = 1,
  seed = NULL,
  initial = NULL
)
```

### Arguments

| | |
|---|---|
| `network` | A CographNetwork object. |
| `iterations` | Number of iterations (default: 500). |
| `cooling` | Rate of temperature decrease (default: 0.95). |
| `repulsion` | Repulsion constant (default: 1). |
| `attraction` | Attraction constant (default: 1). |
| `seed` | Random seed for reproducibility. |
| `initial` | Optional initial coordinates (matrix or data frame). |

### 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)
```

## `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()
```

## `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
```

`output-save` *Output and Saving*

## Description

Functions for saving network visualizations to files.

`palettes` *Color Palettes*

## Description

Built-in color palettes for network visualization.

## `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)
```

## `plot_htna` *Plot Heterogeneous TNA Network (Multi-Group Layout)*

### Description

Plots a TNA model with nodes arranged in multiple groups using geometric layouts:

- 2 groups: Bipartite (two vertical columns or horizontal rows)
- 3+ groups: Polygon (nodes along edges of a regular polygon)

Supports triangle (3), rectangle (4), pentagon (5), hexagon (6), and beyond.

## Usage

```
plot_htna(
  x,
  node_list,
  layout = "auto",
  use_list_order = TRUE,
  jitter = TRUE,
  jitter_amount = 0.8,
  jitter_side = "first",
  orientation = "vertical",
  group1_pos = -1.2,
  group2_pos = 1.2,
  curvature = 0.4,
  group1_color = "#ffd89d",
  group2_color = "#a68ba5",
  group1_shape = "circle",
  group2_shape = "square",
  group_colors = NULL,
  group_shapes = NULL,
  angle_spacing = 0.15,
  edge_colors = NULL,
  legend = TRUE,
  legend_position = "topright",
  extend_lines = FALSE,
  scale = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A tna object or weight matrix. |
| `node_list` | List of 2+ character vectors defining node groups. |
| `layout` | Layout type: "auto" (default), "bipartite", "polygon", or "circular". When "auto", uses bipartite for 2 groups and polygon for 3+ groups. "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:<br>• TRUE (default): Auto-compute jitter based on edge connectivity<br>• FALSE or 0: No jitter (nodes aligned in columns)<br>• Numeric (0-1): Amount of jitter (0.3 = spread nodes 30\\<br>• Named list: Manual per-node offsets by label (e.g., list(Wrong = -0.2))<br>• Numeric vector of length n: Direct x-offsets for each node<br><br>Only applies to bipartite layout. |

| | |
|---|---|
| `jitter_amount` | Base jitter amount when jitter=TRUE. Default 0.5. 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) or "horizontal" (two rows). Ignored for triangle/rectangle layouts. |
| `group1_pos` | Position for first group in bipartite layout. Default -1.2. |
| `group2_pos` | Position for second group in bipartite layout. Default 1.2. |
| `curvature` | Edge curvature amount. Default 0.4 for visible curves. |
| `group1_color` | Color for first group nodes. Default "#ffd89d". |
| `group2_color` | Color for second group nodes. Default "#a68ba5". |
| `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. Required for 3+ groups if not using defaults. |
| `group_shapes` | Vector of shapes for each group. Overrides group1_shape/group2_shape. Required for 3+ groups if not using defaults. |
| `angle_spacing` | Controls empty space at corners (0-1). Default 0.15. Higher values create larger empty angles at vertices. Only applies to triangle/rectangle layouts. |
| `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. |
| `legend` | Logical. Whether to show a legend. Default TRUE for polygon layouts. |
| `legend_position` | Position for legend: "topright", "topleft", "bottomright", "bottomleft", "right", "left", "top", "bottom". Default "topright". |
| `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 high resolution plotting. |
| `...` | Additional parameters passed to tplot(). |

## Value

Invisibly returns the result from tplot().

## Examples

```
# --- 2-group bipartite example ---
nodes_2 <- c("Wrong", "Retry", "Right", "Attempt", "Instruction", "Skip",
              "Order", "Correct", "Hint", "Quit", "Clarify", "Question", "Praise")
set.seed(1)
m2 <- matrix(runif(length(nodes_2)^2, 0, 0.3), length(nodes_2), length(nodes_2))
diag(m2) <- 0
dimnames(m2) <- list(nodes_2, nodes_2)

node_types <- list(
  Student = c("Wrong", "Retry", "Right", "Attempt", "Instruction", "Skip"),
  AI = c("Order", "Correct", "Hint", "Quit", "Clarify", "Question", "Praise")
)
plot_htna(m2, node_types)
plot_htna(m2, node_types, jitter_amount = 0.5)

# --- Triangle layout (3 groups) ---
nodes_3 <- c("Explain", "Question", "Feedback",
              "Answer", "Ask", "Attempt",
              "Hint", "Score", "Progress")
m3 <- matrix(runif(81, 0, 0.3), 9, 9)
diag(m3) <- 0
dimnames(m3) <- list(nodes_3, nodes_3)

node_types_3 <- list(
  Teacher = c("Explain", "Question", "Feedback"),
  Student = c("Answer", "Ask", "Attempt"),
  System  = c("Hint", "Score", "Progress")
)
plot_htna(m3, node_types_3)
plot_htna(m3, node_types_3, layout = "triangle")

# --- Rectangle layout (4 groups) ---
nodes_4 <- c("Click", "Type", "Scroll",
              "Validate", "Transform",
              "Display", "Alert",
              "Save", "Load", "Cache")
m4 <- matrix(runif(100, 0, 0.3), 10, 10)
diag(m4) <- 0
dimnames(m4) <- list(nodes_4, nodes_4)

node_types_4 <- list(
  Input   = c("Click", "Type", "Scroll"),
  Process = c("Validate", "Transform"),
  Output  = c("Display", "Alert"),
  Storage = c("Save", "Load", "Cache")
)
plot_htna(m4, node_types_4)
```

---

## `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

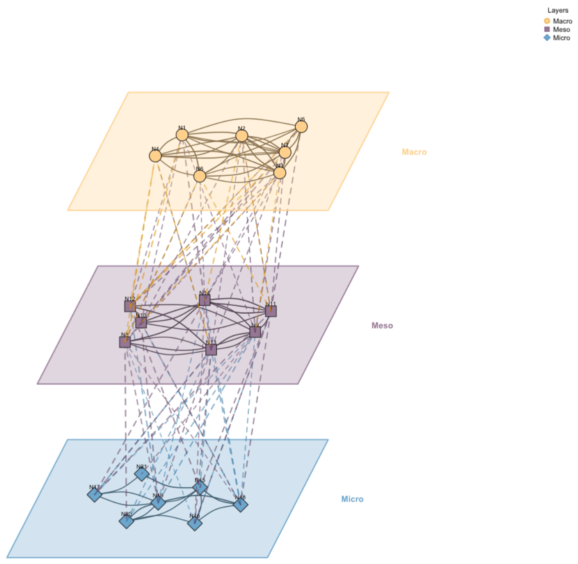
```
plot_mlna(
  model,
  layer_list,
  layout = "horizontal",
  layer_spacing = 2.2,
  layer_width = 4.5,
  layer_depth = 2.2,
  skew_angle = 25,
  node_spacing = 0.7,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  within_edges = TRUE,
  between_edges = TRUE,
  between_style = 2,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.15,
  node_size = 3,
  minimum = 0,
  scale = 1,
  ...
)

mlna(
  model,
  layer_list,
  layout = "horizontal",
  layer_spacing = 2.2,
  layer_width = 4.5,
  layer_depth = 2.2,
  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,
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | A tna object or weight matrix. |
| `layer_list` | List of character vectors defining layers. Each element contains node names belonging to that layer. Layers are displayed from top to bottom in list order. |
| `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 2.2. |
| `layer_width` | Horizontal width of each layer shell. Default 4.5. |
| `layer_depth` | Depth of each layer (for 3D effect). Default 2.2. |
| `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 2.5. |
| `minimum` | Minimum edge weight threshold. Edges below this are hidden. Default 0. |
| `scale` | Scaling factor for high resolution plotting. |
| `...` | Additional parameters (currently unused). |

## Details



## Value

Invisibly returns NULL.

## Examples

```
# Create multilevel network
set.seed(42)
nodes <- paste0("N", 1:15)
m <- matrix(runif(225, 0, 0.3), 15, 15)
diag(m) <- 0
colnames(m) <- rownames(m) <- nodes

# Define 3 layers
layers <- list(
  Macro = paste0("N", 1:5),
  Meso = paste0("N", 6:10),
  Micro = paste0("N", 11:15)
)

# Basic usage
plot_mlna(m, layers)

# Customized
plot_mlna(m, layers,
     layer_spacing = 2.5,
     layer_width = 5,
     between_style = 2,  # dashed
     minimum = 0.1)

# Circle layout within layers
plot_mlna(m, layers, layout = "circle")
```

---

`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 shape (circle, square, diamond, triangle) containing its nodes.

## Usage

```
plot_mtna(
  x,
  cluster_list,
  layout = "circle",
  spacing = 3,
  shape_size = 1.2,
  node_spacing = 0.5,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  bundle_edges = TRUE,
  bundle_strength = 0.8,
  summary_edges = TRUE,
  within_edges = TRUE,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.3,
  node_size = 2,
  scale = 1,
  ...
)

mtna(
  x,
  cluster_list,
  layout = "circle",
  spacing = 3,
  shape_size = 1.2,
  node_spacing = 0.5,
  colors = NULL,
  shapes = NULL,
  edge_colors = NULL,
  bundle_edges = TRUE,
  bundle_strength = 0.8,
  summary_edges = TRUE,
  within_edges = TRUE,
  show_border = TRUE,
  legend = TRUE,
  legend_position = "topright",
  curvature = 0.3,
  node_size = 2,
  scale = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A tna object or weight matrix. |
| `cluster_list` | List of character vectors defining clusters. Each cluster becomes a separate shape in the layout. |
| `layout` | How to arrange the clusters: "circle" (default), "grid", "horizontal", "vertical". |
| `spacing` | Distance between cluster centers. Default 3. |
| `shape_size` | Size of each cluster shape (shell radius). Default 1.2. |
| `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: "circle", "square", "diamond", "triangle". Default cycles through these. |
| `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. |
| `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 2. |
| `scale` | Scaling factor for high resolution plotting. |
| `...` | Additional parameters passed to plot_tna(). |

## Value

Invisibly returns NULL for summary mode, or the plot_tna result.

## Examples

```
# Create network with 4 clusters
nodes <- paste0("N", 1:20)
m <- matrix(runif(400, 0, 0.3), 20, 20)
diag(m) <- 0
colnames(m) <- rownames(m) <- nodes

clusters <- list(
  North = paste0("N", 1:5),
  East = paste0("N", 6:10),
  South = paste0("N", 11:15),
  West = paste0("N", 16:20)
)

# Summary edges between clusters + individual edges within
plot_mtna(m, clusters, summary_edges = TRUE)

# Control spacing and sizes
plot_mtna(m, clusters, spacing = 4, shape_size = 1.5, node_spacing = 0.6)
```

---

## `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.labels = TRUE,
  edge.label.position = 0.7,
  edge.label.cex = 0.6,
  edge.color = "#003355",
  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.labels = TRUE,
  edge.label.position = 0.7,
  edge.label.cex = 0.6,
  edge.color = "#003355",
  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.labels` | Show edge weight labels |

| | |
|---|---|
| `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().

## 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)
```

---

`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 a custom SVG shape from an inline SVG string
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>')

# Create a small adjacency matrix
adj <- matrix(c(0, 1, 1, 0, 0, 1, 1, 0, 0), nrow = 3,
              dimnames = list(c("A", "B", "C"), c("A", "B", "C")))

# Use in network (requires grImport2 for SVG rendering; falls back to circle)
cograph(adj) |> sn_nodes(shape = "simple_star")
```

## `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"
))
```

---

`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. Updates the from, to, weight vectors and n_edges.

## 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_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. Automatically updates n_nodes and labels.

### 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)
```

---

`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 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` | 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. |
| `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). Default FALSE. |
| `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)

# Basic: auto-style by weight
cograph(adj) |>
  sn_edges(width = "weight", color = "weight")

# Direct matrix input (auto-converted)
adj |> sn_edges(width = 2, color = "gray50")

# Custom positive/negative colors
cograph(adj) |>
  sn_edges(
    color = "weight",
    edge_positive_color = "darkblue",
    edge_negative_color = "darkred"
  ) |>
  splot()

# Edge labels showing weights
cograph(adj) |>
  sn_edges(labels = TRUE, label_size = 0.8) |>
  splot()

# Statistical output with CI template
# Suppose we have estimates, lower/upper CI bounds
estimates <- c(0.5, -0.3, 0.8)
ci_lo <- c(0.2, -0.6, 0.5)
ci_hi <- c(0.8, -0.1, 1.1)

cograph(adj) |>
  sn_edges(
    label_template = "{est} [{low}, {up}]",
    ci_lower = ci_lo,
    ci_upper = ci_hi,
    label_digits = 2
  ) |>
  splot()

# Curved edges for reciprocal pairs
cograph(adj) |>
  sn_edges(curves = "mutual", curvature = 0.3) |>
  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.

**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.

**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)

# Built-in layouts
cograph(adj) |> sn_layout("circle") |> splot()
cograph(adj) |> sn_layout("spring") |> splot()

# igraph layouts (if igraph installed)
if (requireNamespace("igraph", quietly = TRUE)) {
  cograph(adj) |> sn_layout("kk") |> splot()
  cograph(adj) |> sn_layout("fr") |> splot()
}

# Custom coordinates
coords <- matrix(c(0, 0, 1, 0, 0.5, 1), ncol = 2, byrow = TRUE)
cograph(adj) |> sn_layout(coords) |> splot()

# Direct matrix input (auto-converts)
adj |> sn_layout("circle")
```

---

## `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)

# Basic usage with cograph()
cograph(adj) |>
  sn_nodes(size = 0.08, fill = "steelblue", shape = "circle")

# Direct matrix input (auto-converted)
adj |> sn_nodes(fill = "coral", size = 0.1)

# Per-node customization with vectors
cograph(adj) |>
  sn_nodes(
    size = c(0.08, 0.06, 0.1),
    fill = c("red", "blue", "green"),
    label_position = c("above", "below", "center")
  ) |>
  splot()

# Donut chart nodes showing proportions
cograph(adj) |>
  sn_nodes(
    donut_fill = c(0.25, 0.75, 0.5),
    donut_color = "steelblue",
    donut_show_value = TRUE,
    donut_value_suffix = "%"
  ) |>
  splot()

# Mixed shapes per node
cograph(adj) |>
  sn_nodes(
    shape = c("circle", "square", "triangle"),
    fill = c("#E41A1C", "#377EB8", "#4DAF4A")
  ) |>
  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.

**bright** Saturated, vivid colors.

**grayscale** Shades of gray.

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)

# Apply palette to nodes
cograph(adj) |> sn_palette("viridis") |> splot()

# Apply to edges
cograph(adj) |> sn_palette("colorblind", target = "edges") |> splot()

# Apply to both
cograph(adj) |> sn_palette("pastel", target = "both") |> splot()

# Custom palette function
my_pal <- function(n) rainbow(n, s = 0.7)
cograph(adj) |> sn_palette(my_pal) |> splot()

# Direct matrix input
adj |> sn_palette("viridis")
```

## 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

Invisible filename.

### Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), nrow = 3)
# With cograph()
net <- cograph(adj)
sn_save(net, file.path(tempdir(), "network.pdf"))

# Direct matrix input
sn_save(adj, file.path(tempdir(), "network.png"), dpi = 300)
```

## 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

Invisible filename.

### 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.

**grayscale** Black and white only.

**vibrant** Bold, saturated 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)

# Apply different themes
cograph(adj) |> sn_theme("dark") |> splot()
cograph(adj) |> sn_theme("minimal") |> splot()

# Override specific theme properties
cograph(adj) |> sn_theme("classic", background = "lightgray") |> splot()

# Direct matrix input
adj |> sn_theme("dark")
```

---

## 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,
  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,
  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. |
| `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

Invisible NULL. Called for 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
)
```

---

`splot` *Base R Graphics Network Plotting*

---

## Description

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(
  x,
  layout = "oval",
  directed = NULL,
  seed = 42,
  theme = NULL,
  node_size = NULL,
  node_size2 = NULL,
  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_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 = "white",
  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_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,
  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_label_style = "none",
  edge_label_template = NULL,
  edge_label_digits = 2,
  edge_label_oneline = TRUE,
  edge_label_ci_format = "bracket",
  edge_ci_lower = NULL,
  edge_ci_upper = NULL,
  edge_label_p = 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",
  legend = FALSE,
  legend_position = "topright",
  legend_size = 0.8,
  legend_edge_colors = TRUE,
  legend_node_sizes = FALSE,
  groups = NULL,
  node_names = NULL,
  filetype = "default",
  filename = file.path(tempdir(), "splot"),
  width = 7,
  height = 7,
  res = 600,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Network input. Can be: <br>• A square numeric matrix (adjacency/weight matrix)<br>• A data frame with edge list (from, to, optional weight columns)<br>• An igraph object<br>• A cograph_network object |
| `layout` | Layout algorithm: "circle", "spring", "groups", or a matrix of x,y coordinates, or an igraph layout function. Also supports igraph two-letter codes: "kk", "fr", "drl", "mds", "ni", etc. Default is "oval" |
| `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 3. |
| `node_size2` | Secondary node size for ellipse/rectangle height. |
| `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_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 0.5. |
| `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` | Base edge size for weight scaling. NULL (default) uses adaptive sizing based on network size: `15 * exp(-n_nodes/90) + 1`. For directed networks, this is halved. 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(0.5, 4). Edges are scaled to fit within this range. |
| `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` | Two-tier cutoff for edge width scaling. NULL (default) = auto-calculate as 75th percentile of weights (qgraph behavior). 0 = disabled (continuous scaling). Positive number = manual threshold. Edges below cutoff get minimal width variation. |
| `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_style` | Line type(s): 1=solid, 2=dashed, 3=dotted, etc. |
| `curvature` | Edge curvature. 0 for straight, positive/negative for curves. |
| `curve_scale` | Logical: auto-curve reciprocal edges? |
| `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. |
| `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_label_style` | Preset style: "none", "estimate", "full", "range", "stars". |
| `edge_label_template` | Template with placeholders: {est}, {range}, {low}, {up}, {p}, {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_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_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. |
| `edge_positive_color` | Color for positive weights. |
| `positive_color` | Deprecated. Use `edge_positive_color` instead. |
| `edge_negative_color` | Color for negative 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. |
| `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. |
| `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). |
| `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. |
| `...` | Additional arguments passed to layout functions. |

## 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, `{stars}` for significance stars.

**edge_label_style** Preset styles: `"estimate"` (weight only), `"full"` (estimate + CI), `"range"` (CI only), `"stars"` (significance).

## Value

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 network from adjacency matrix
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)

# With curved edges
splot(adj, curvature = 0.2)

# Weighted network with colors
w_adj <- matrix(c(0, 0.5, -0.3, 0,
                  0.8, 0, 0.4, -0.2,
                  0, 0, 0, 0.6,
                  0, 0, 0, 0), 4, 4, byrow = TRUE)
splot(w_adj, edge_positive_color = "darkgreen", edge_negative_color = "red")

# Pie chart nodes
splot(adj, pie_values = list(c(1,2,3), c(2,2), c(1,1,1,1), c(3,1)))

# Circle layout with labels
splot(adj, layout = "circle", labels = c("A", "B", "C", "D"))
```

## `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()
```

## `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")
```

# Index