Package: autograph (via r-universe)

July 21, 2026

Title Automatic Plotting and Theming of Many Graphs

Version 1.1.1

Description Visual exploration and presentation of networks should not be difficult. This package includes functions for plotting networks and network-related metrics with sensible and pretty defaults. It includes 'ggplot2'-based plot methods for many popular network package classes. It also includes some novel layout algorithms, and options for straightforward, consistent themes.

URL https://stocnet.github.io/autograph/

BugReports https://github.com/stocnet/autograph/issues

License MIT + file LICENSE

Language en-GB

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0), manynet

Imports dplyr (>= 1.1.0), ggdendro, ggraph (>= 2.2.0), ggplot2 (>= 4.0.0), igraph, patchwork, tidygraph

Suggests gganimate, ggforce (>= 0.5.0), gifski, graphlayouts, knitr, methods, migraph, netrics, testthat (>= 3.0.0)

Enhances ergm, RSiena

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown

Config/testthat/parallel true

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID: <https://orcid.org/0000-0002-8361-9647>), Henrique Sposito [ctb] (ORCID: <https://orcid.org/0000-0003-3420-6085>)

Maintainer James Hollway <james.hollway@graduateinstitute.ch>

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

Date/Publication 2026-07-21 14:30:11 UTC

RemoteUrl https://github.com/cran/autograph

RemoteRef HEAD

RemoteSha 846fe7398e558e75fdfde32a25900abafdd028db

Contents

ag_call 2
layout_configuration 4
layout_layered 5
layout_matching 5
layout_partition 6
layout_valence 8
made_earlier 9
map_measure 10
map_member 11
map_motifs 12
model_mrqap 13
plot.diffusion 13
plot.network_test 14
plot_adequacy 15
plot_convergence 16
plot_gof 17
plot_graphr 19
plot_graphs 22
plot_grapht 23
plot_interp 26
theme_match 28
theme_set 29

Index **31**

`ag_call` *Consistent palette calls*

Description

These functions assist in calling particular parts of a theme's palette. For example, `ag_base()` will return the current theme's base or background color, and `ag_highlight()` will return the color used in that theme to highlight one or more nodes, lines, or such.

Using palettes that are high contrast, aesthetically pleasing, and institutionally or thematically consistent is not without its challenges.

Usage

```
ag_base()

ag_highlight()

ag_positive()

ag_negative()

ag_qualitative(number)

ag_sequential(number)

ag_divergent(number)

ag_font()
```

Arguments

`number`	Integer of how many category colours to return.

Value

One or more hexcodes as strings.

Colour blindness

The default palettes are designed to be colour-blind friendly. There are different types of colour-blindness. The most common type, red-green colour-blindness, finds it difficult to distinguish between the red and green hues used in the rainbow palette, for instance. Fortunately there are a range of palettes that function fairly well for those who are color-blind. These include the viridis palette, and the ColorBrewer palettes (included in the RColorBrewer package). The default palettes in `{autograph}` are designed to be colour-blind friendly, but users should always check that their visualisations serve their intended audience.

`layout_configuration` *Layout algorithms based on configurational positions*

Description

Configurational layouts locate nodes at symmetric coordinates to help illustrate particular configurations. Currently configurational layouts are available for 2-6 nodes. The "configuration" layout will choose the appropriate configurational layout automatically.

Usage

```
layout_configuration(.data, circular = TRUE, times = 1)

layout_tbl_graph_configuration(.data, circular = TRUE, times = 1)

layout_dyad(.data, circular = TRUE, times = 1)

layout_tbl_graph_dyad(.data, circular = TRUE, times = 1)

layout_triad(.data, circular = TRUE, times = 1)

layout_tbl_graph_triad(.data, circular = TRUE, times = 1)

layout_tetrad(.data, circular = TRUE, times = 1)

layout_tbl_graph_tetrad(.data, circular = TRUE, times = 1)

layout_pentad(.data, circular = TRUE, times = 1)

layout_tbl_graph_pentad(.data, circular = TRUE, times = 1)

layout_hexad(.data, circular = TRUE, times = 1)

layout_tbl_graph_hexad(.data, circular = TRUE, times = 1)
```

Arguments

`.data`	Some {manynet} compatible network data.
`circular`	Logical, required for {ggraph} compatibility, default TRUE.
`times`	Integer, how many times to run the algorithm. Required by for {ggraph} compatibility, but not used here, so default = 1.

See Also

Other mapping: `layout_partition`, `plot_graphr`, `plot_graphs`, `plot_grapht`

layout_layered *Layered layout*

Description

Layered layout

Usage

```
layout_tbl_graph_layered(.data, center = NULL, circular = FALSE, times = 4)
```

Arguments

`.data`	Some {manynet} compatible network data.
`center, circular`	Extra parameters required for {tidygraph} compatibility.
`times`	Integer of sweeps that the algorithm will pass through. By default 4.

Value

Returns a table of coordinates.

Examples

```
ties <- data.frame(
  from = c("A", "A", "B", "C", "D", "F", "F", "E"),
  to   = c("B", "C", "D", "E", "E", "E", "G", "G"),
  stringsAsFactors = FALSE)

coords <- layout_tbl_graph_layered(ties, times = 6)
coords
```

layout_matching *Matching layout*

Description

This layout works to position nodes opposite their matching nodes. See `manynet::to_matching()` for more details on the matching procedure.

Usage

```
layout_tbl_graph_matching(.data, center = NULL, circular = FALSE, times = 1)
```

Arguments

`.data`	Some {manynet} compatible network data.
`center, circular, times`	Extra parameters required for {tidygraph} compatibility.

Value

Returns a table of nodes' x and y coordinates.

`layout_partition` *Layout algorithms based on bi- or other partitions*

Description

These algorithms layout networks based on two or more partitions, and are recommended for use with `graphr()` or {ggraph}.

The "hierarchy" layout layers the first node set along the bottom, and the second node set along the top, sequenced and spaced as necessary to minimise edge overlap. The "alluvial" layout is similar to "hierarchy", but places successive layers horizontally rather than vertically. The "railway" layout is similar to "hierarchy", but nodes are aligned across the layers. The "ladder" layout is similar to "railway", but places successive layers horizontally rather than vertically. The "concentric" layout places a "hierarchy" layout around a circle, with successive layers appearing as concentric circles. The "multilevel" layout places successive layers as multiple levels. The "lineage" layout ranks nodes in Y axis according to values.

Usage

```
layout_concentric(
  .data,
  membership,
  radius = NULL,
  order.by = NULL,
  circular = FALSE,
  times = 1000
)

layout_tbl_graph_concentric(
  .data,
  membership,
  radius = NULL,
  order.by = NULL,
  circular = FALSE,
  times = 1000
)

layout_multilevel(.data, level, circular = FALSE)
```

```
layout_tbl_graph_multilevel(.data, level, circular = FALSE)

layout_lineage(.data, rank, circular = FALSE)

layout_tbl_graph_lineage(.data, rank, circular = FALSE)

layout_hierarchy(.data, center = NULL, circular = FALSE, times = 1000)

layout_tbl_graph_hierarchy(
  .data,
  center = NULL,
  circular = FALSE,
  times = 1000
)

layout_alluvial(.data, circular = FALSE, times = 1000)

layout_tbl_graph_alluvial(.data, circular = FALSE, times = 1000)

layout_railway(.data, circular = FALSE, times = 1000)

layout_tbl_graph_railway(.data, circular = FALSE, times = 1000)

layout_ladder(.data, circular = FALSE, times = 1000)

layout_tbl_graph_ladder(.data, circular = FALSE, times = 1000)
```

Arguments

.data	Some {manynet} compatible network data.
membership	A node attribute or a vector to draw concentric circles for "concentric" layout.
radius	A vector of radii at which the concentric circles should be located for "concentric" layout. By default this is equal placement around an empty centre, unless one (the core) is a single node, in which case this node occupies the centre of the graph.
order.by	An attribute label indicating the (decreasing) order for the nodes around the circles for "concentric" layout. By default ordering is given by a bipartite placement that reduces the number of edge crossings.
circular	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
times	Maximum number of iterations, where appropriate
level	A node attribute or a vector to hierarchically order levels for "multilevel" layout.
rank	A numerical node attribute to place nodes in Y axis according to values for "lineage" layout.

center — Further split "hierarchical" layouts by declaring the "center" argument as the "events", "actors", or by declaring a node name in hierarchy layout. Defaults to NULL.

Source

Diego Diez, Andrew P. Hutchins and Diego Miranda-Saavedra. 2014. "Systematic identification of transcriptional regulatory modules from protein-protein interaction networks". *Nucleic Acids Research*, 42 (1) e6.

See Also

Other mapping: layout_configuration(), plot_graphr, plot_graphs, plot_grapht

Examples

```
#graphr(ison_southern_women, layout = "concentric", membership = "type",
#           node_color = "type", node_size = 3)
#graphr(ison_lotr, layout = "multilevel",
#           node_color = "Race", level = "Race", node_size = 3)
# ison_adolescents %>%
#   mutate(year = rep(c(1985, 1990, 1995, 2000), times = 2),
#          cut = node_is_cutpoint(ison_adolescents)) %>%
#   graphr(layout = "lineage", rank = "year", node_color = "cut",
#              node_size = migraph::node_degree(ison_adolescents)*10)
#graphr(ison_southern_women, layout = "hierarchy", center = "events",
#           node_color = "type", node_size = 3)
#graphr(ison_southern_women, layout = "alluvial")
```

---

layout_valence *Valence-based layout*

Description

Valence-based layout

Usage

```
layout_valence(
  .data,
  times = 500,
  center = NULL,
  circular = FALSE,
  repulsion_coef = 1,
  attraction_coef = 0.05
)

layout_tbl_graph_valence(
  .data,
```

```
  times = 500,
  center = NULL,
  circular = FALSE,
  repulsion_coef = 1,
  attraction_coef = 0.05
)
```

Arguments

.data	Some {manynet} compatible network data.
times	Integer of sweeps that the algorithm will pass through. By default 4.
center, circular	Extra parameters required for {tidygraph} compatibility.
repulsion_coef	Coefficient for global repulsion force. Default is 1.
attraction_coef	Coefficient for edge-based attraction/repulsion force. Default is 0.05.

Examples

```
edges <- data.frame(
  from = c("A", "B", "C", "D"),
  to   = c("B", "C", "D", "A"),
  weight = c(2, 3, 1, 4),
  sign = c(1, -1, 1, -1)  # 1 = positive, -1 = negative
  )
graphr(as_igraph(edges), layout="valence")
```

---

made_earlier *Precooked results for demonstrating plotting*

Description

These are all pre-cooked results objects, saved here to save time in testing and demonstrating how autograph plots look.

Usage

```
data(res_migraph_reg)

data(res_migraph_test)

data(res_migraph_diff)

data(res_manynet_diff)

data(siena_gof)
```

```
data(siena_influence)

data(siena_selection)

data(monan_conv)

data(monan_gof)

data(ergm_gof)

data(goldfish_outliers)

data(goldfish_changepoints)
```

Format

An object of class `netlm` of length 15.

An object of class `network_test` of length 9.

An object of class `diffs_model` (inherits from `data.frame`) with 20 rows and 11 columns.

An object of class `diff_model` (inherits from `tbl_df`, `tbl`, `data.frame`) with 4 rows and 10 columns.

An object of class `sienaGOF` of length 1.

An object of class `influenceTable` (inherits from `data.frame`) with 25 rows and 4 columns.

An object of class `selectionTable` (inherits from `data.frame`) with 25 rows and 4 columns.

An object of class `traces.monan` of length 3.

An object of class `gof.stats.monan` of length 2.

An object of class `gof.ergm` (inherits from `gof`) of length 30.

An object of class `outliers.goldfish` (inherits from `dependent.goldfish`, `data.frame`) with 12 rows and 7 columns.

An object of class `changepoints.goldfish` (inherits from `list`) of length 2.

`map_measure` *Plotting logical marks Plotting numeric measures*

Description

These functions plot distributions for node, tie, and network measures, as defined in the `{manynet}` package.

Usage

```
## S3 method for class 'node_measure'
plot(x, type = c("h", "d"), ...)

## S3 method for class 'tie_measure'
plot(x, type = c("h", "d"), ...)

## S3 method for class 'network_measures'
plot(x, ...)
```

Arguments

x	An object of "node_measure", "tie_measure", or "network_measures" class.
type	For node and tie measures, whether the plot should be "h" a histogram or "d" a density plot. By default "h".
...	Other arguments to be passed on.

Value

`plot.node_measure()` and `plot.tie_measure()` returns a histogram and/or density plot of the distribution of the measure.

`plot.network_measures()` returns a plot of the measure traced over time.

Examples

```
plot(netrics::node_by_deg(ison_karateka))
plot(netrics::tie_by_betweenness(ison_karateka))
```

---

map_member *Plotting categorical memberships*

Description

This plotting method operates on "node_member" class objects from the {manynet} package, plotting the dendrogram of their membership.

Usage

```
## S3 method for class 'node_member'
plot(x, ...)

## S3 method for class 'matrix'
plot(x, ..., membership = NULL)
```

Arguments

x	An object of "node_member" class, for example as a result of running manynet::node_in_community().
...	Other arguments to be passed on.
membership	A "node_member" membership vector.

Value

plot.node_member() returns a dendrogram, with labels colored to indicate the different clusters, and with the optimal cutpoint shown by a dashed highlight line.

plot.matrix() returns a plot of an adjacency or incidency matrix, potentially with the rows and columns reordered to illustrate an additional membership vector.

Examples

```
plot(netrics::node_in_walktrap(ison_southern_women, "e"))
plot(as_matrix(ison_adolescents),
  membership = netrics::node_in_walktrap(ison_adolescents, "e"))
plot(as_matrix(ison_southern_women),
  membership = netrics::node_in_walktrap(ison_southern_women, "e"))
```

---

map_motifs *Plotting tabular motifs*

Description

These functions will plot graphs of the motifs used in a vector of results of e.g. a triad census.

Usage

```
## S3 method for class 'node_motif'
plot(x, ...)

## S3 method for class 'network_motif'
plot(x, ...)
```

Arguments

x	An object of "node_motif" class, e.g. resulting from a call to manynet::node_by_triad().
...	Other arguments to be passed on.

Value

plot.node_motif() returns a set of graphs that illustrate the motifs mentioned in the results from a node_motif function in {manynet}.

plot.network_motif() returns a set of graphs that illustrate the motifs mentioned in the results from a net_motif function in {manynet}.

model_mrqap *Plotting methods for MRQAP models*

Description

These plotting methods are for results obtained by fitting an MRQAP model. The S3 classes are "netlm" or "netlogit", and so are compatible with the results from either the {sna} or {migraph} packages.

Usage

```
## S3 method for class 'netlm'
plot(x, ...)

## S3 method for class 'netlogit'
plot(x, ...)
```

Arguments

x	An object obtained by fitting an MRQAP model to some data. For example, migraph::net_regression().
...	Further arguments to be passed on to plot.

Value

A plot showing the location of observed statistics compared to the distribution of statistics from permuted networks.

Examples

```
# Here's something I cooked up with migraph earlier:
plot(res_migraph_reg)
```

plot.diffusion *Plotting diffusion models*

Description

Plotting diffusion models

Usage

```
## S3 method for class 'diff_model'
plot(x, ..., all_steps = TRUE)

## S3 method for class 'diffs_model'
plot(x, ...)

## S3 method for class 'learn_model'
plot(x, ...)
```

Arguments

x	A "diff_model" of "diffs_model" class of object. E.g. as a result from `manynet::play_diffusion()`.
...	Other arguments to be passed.
all_steps	Whether all steps should be plotted or just those where there is change in the distributions.

Value

`plot.diff_model()` returns a bar chart of the number of new infected nodes at each time point, as well as an overlay line plot of the total of infected

Examples

```
plot(res_manynet_diff)
plot(res_migraph_diff)
plot(play_learning(ison_networkers, beliefs = runif(net_nodes(ison_networkers))))
```

---

plot.network_test *Plotting methods for CUG and QAP tests*

Description

These plotting methods are for results obtained by testing some statistic against those produced in a reference distribution of conditional uniform graphs or as a quadratic assignment procedure. The S3 class is "network_test".

Usage

```
## S3 method for class 'network_test'
plot(x, ..., threshold = 0.95, tails = c("two", "one"))
```

Arguments

x	An object obtained from a conditional uniform graph or quadratic assignment procedure test. For example, `migraph::test_permutation()`.
...	Other arguments to be passed on.
threshold	The empirical threshold to shade in the plot.
tails	By default "two" indicating a two-tailed test, but "one" for a one-tailed test is also available.

Value

A distribution of the simulated or permuted statistics, with 2.5% shaded at each end, and a line highlighting where the observed statistic lies on this distribution.

Examples

```
# Here's something I cooked up with migraph earlier:
plot(res_migraph_test)
```

---

plot_adequacy *Plotting adequacy diagnostics*

Description

These plotting methods are for diagnosing the adequacy of model specification, such as those used in goldfish. These plots are useful for identifying whether there might be significant outliers affecting the results or significant time heterogeneity.

Usage

```
## S3 method for class 'outliers.goldfish'
plot(x, ...)

## S3 method for class 'changepoints.goldfish'
plot(x, ...)
```

Arguments

x	An object of class "outliers.goldfish" or "changepoints.goldfish".
...	Additional plotting parameters, currently unused.

Value

The function shows a line plot tracing the statistics obtained at each simulation step, as well as a density plot showing the distribution of the statistics over the entire simulation.

Examples

```
plot(goldfish_outliers)
plot(goldfish_changepoints)
```

`plot_convergence` *Plotting convergence diagnostics*

Description

These plotting methods are for diagnosing the convergence of simulation-based estimation procedures, such as those used in MoNAn and ergm. These plots are useful for identifying whether the estimation procedure has adequately explored the state space and converged to a stable distribution.

Usage

```
## S3 method for class 'ag_conv'
plot(x, ...)

## S3 method for class 'traces.monan'
plot(x, ...)

## S3 method for class 'ergm'
plot(x, ...)

load_ergm_res()
```

Arguments

`x`	An object of class "traces.monan".
`...`	Additional plotting parameters, currently unused.

Value

The function shows a line plot tracing the statistics obtained at each simulation step, as well as a density plot showing the distribution of the statistics over the entire simulation.

See Also

Other MoNAn: `plot_gof`

Other ergm: `plot_gof`

Examples

```
plot(monan_conv)
ergm_res <- load_ergm_res()
plot(ergm_res)
```

plot_gof *Plotting goodness-of-fit results*

Description

These plot methods plot goodness of fit objects created using `RSiena::sienaGOF()`, `MoNAn::monanGOF()`, or the 'ergm' package's gof() function. Internally, the GOF object is translated into a common class (`ag_gof`), which has its own plot method to ensure a consistent look and feel. It is not expected that users will create `ag_gof` class objects themselves.

The plot shows a violin plot of the distribution of statistics from the simulations, with a boxplot inside the violin to show the interquartile range, and dashed lines connecting the 5th and 95th percentiles. The boxplot also shows outliers as crosses. The observed statistics are shown as points and connected by a line. The observed statistics are also labelled with their value. If a p-value is available (as in the case of `RSiena::sienaGOF()`), it is shown beneath the x-axis.

Usage

```
## S3 method for class 'ag_gof'
plot(x, ...)

## S3 method for class 'gof.stats.monan'
plot(x, cumulative = FALSE, ...)

## S3 method for class 'sienaGOF'
plot(x, cumulative = FALSE, ...)

## S3 method for class 'gof.ergm'
plot(
  x,
  cumulative = FALSE,
 statistic = c("degree", "odegree", "idegree", "b1degree", "b2degree", "espartners",
    "dspartners", "distance"),
  ...
)
```

Arguments

`x`	An object of class "sienaGOF", "gof.stats.monan", or "gof.ergm".
`...`	Other parameters to be passed to the plotting function, for example `main = "Title"` for a different title than the default.
`cumulative`	Logical, indicating whether the statistics should be plotted cumulatively (default FALSE). This is typically treated in `sienaGOF()` for {RSiena}, but treated within the plotting function for {MoNAn} and 'ergm'.
`statistic`	Character, indicating which statistic to plot. Since 'ergm' package GOFs include goodness of fit on multiple statistics, the user must specify which statistic to plot. Options are `"deg"` (degree distribution), `"espart"` (edgewise shared partners), and `"dist"` (geodesic distance). The default is `"deg"`.

Details

Since these plots methods are in {autograph}, the plots are automatically themed according to the current theme set using stocnet_theme(). The function uses the highlight colour defined in the current theme to highlight the observed statistics. The function also uses the base colour defined in the current theme to draw the violin and box plots.

It is however completely customisable. While a title is automatically generated so that the graph is informative, this can be customised by specifying the main argument in the plotting function, or added after the fact using {ggplot2} functions such as ggtitle() or labs().

The user can choose whether to plot the statistics cumulatively or not. This is typically handled within RSiena::sienaGOF(), but for MoNAn::monanGOF() and the 'ergm' package's gof() function the cumulative option is handled here. The default is to plot the non-cumulative statistics. This is because the non-cumulative statistics are often more interpretable, and the cumulative statistics can be obtained by setting cumulative = TRUE.

The function also checks whether any of the statistics have zero variance across the simulations, and if so, these statistics are not plotted, with a message to the user indicating which statistics were omitted.

Note that these methods overwrite any plot methods for these classes that may be provided by the original packages. You may receive such a warning in the console when loading the package. Please load {autograph} after these other packages to ensure the plotting methods included in this package are used, or specify the package when calling the plotting method directly, e.g., autograph:::plot.sienaGOF(res_siena_gof).

Value

A violin plot showing the distribution of statistics from the simulations and a line joining points showing the observed statistics.

References

Hintze, J. L. and Nelson, R. D. 1998. "Violin plots: A box plot-density trace synergism". *The American Statistician*, 52:181–184. doi:10.1080/00031305.1998.10480559

See Also

Other MoNAn: plot_convergence

Other RSiena: plot_interp

Other ergm: plot_convergence

Examples

```
plot(monan_gof)
plot(siena_gof, cumulative = TRUE)
plot(ergm_gof, statistic = "espart")
```

plot_graphr *Easily graph networks with sensible defaults*

Description

This function provides users with an easy way to graph (m)any network data for exploration, investigation, inspiration, and communication.

`graphr()` builds upon {ggplot2} and {ggraph} to offer pretty, easy, and extensible graphing solutions. Just passing the function some network data will often be sufficient to return a reasonable-looking graph.

The function also makes it easy to modify many of the most commonly adapted aspects of a graph, including node and edge size, colour, and shape, as arguments rather than additional functions that you need to remember. These can be defined outright, e.g. `node_size = 8`, or in reference to an attribute of the network, e.g. `node_size = "wealth"`.

Lastly, `graphr()` uses {ggplot2}-related theme information, so it is easy to make colour palette and fonts institution-specific and consistent. See e.g. `theme_iheid()` for more.

To learn more about what can be done visually, try `run_tute("Visualisation")`.

Usage

```
graphr(
  .data,
  layout = NULL,
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  node_group,
  edge_color,
  edge_size,
  isolates = c("legend", "caption", "keep"),
  snap = FALSE,
  label_dist = NULL,
  label_repel = TRUE,
  edge_bundle = FALSE,
  ...,
  node_colour,
  edge_colour
)
```

Arguments

`.data`	A manynet-consistent object.
`layout`	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress"

for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.

`labels` Logical, whether to print node names as labels if present.

`node_color`, `node_colour`
Node variable to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be colored by declaring a color instead.

`node_shape` Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.

`node_size` Node variable to be used for sizing the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all node-related statistics prior to using this function. Nodes can also be sized by declaring a numeric size or vector instead.

`node_group` Node variable to be used for grouping the nodes. It is easiest if this is added as a hull over groups before plotting. Group variables should have a minimum of 3 nodes, if less, number groups will be reduced by merging categories with lower counts into one called "other".

`edge_color`, `edge_colour`
Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.

`edge_size` Tie variable to be used for sizing the edges. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.

`isolates` Character scalar, how to treat isolates. "keep" will keep isolates in the graph as they are. "legend" (default) will remove isolates from the graph but note them in the legend. "caption" will remove isolates from the graph but note them in the caption. If there are no isolates, this argument will be ignored. If the default layout ("stress") is used, we recommend that the "legend" option is used to avoid isolates crowding out the giant component.

`snap` Logical scalar, whether the layout should be snapped to a grid.

label_dist — Numeric scalar, in points (pt), controlling the extra gap left between labels and node borders – similar to igraph's vertex.label.dist. Node size is always accounted for automatically (larger nodes push labels further away without any extra configuration); label_dist adds further spacing on top of that, and defaults to a small gap (5pt). Set to 0 for labels right at the node border, or to a larger value (e.g. 15) for more spacing. Only used when labels = TRUE and label_repel = TRUE (as the padding passed to the repel algorithm) or label_repel = FALSE (as a fixed nudge away from the node, in the layouts where this makes sense, e.g. "circle"/"concentric", "bipartite"/"railway", "alluvial").

label_repel — Logical scalar, whether labels should be repelled away from each other and from nodes using ggrepel (via ggraph's repel argument). Defaults to TRUE. Set to FALSE to place labels at a fixed offset (see label_dist) without the (sometimes slow, and non-deterministic between runs for some layouts) repelling algorithm.

edge_bundle — Edge bundling, off by default (FALSE). When TRUE (or equivalently "force"), edges are bundled together using ggraph's force-directed edge bundling (geom_edge_bundle_force()), which pulls nearby edges into shared paths to reduce visual clutter in dense networks. Alternative non-hierarchical algorithms can be selected by name: "path" (geom_edge_bundle_path()) or "minimal" (geom_edge_bundle_minimal()). Bundling only makes a visible difference when a network has enough edges; for directed networks arrowheads are retained, but the slight reciprocal-tie curvature used for unbundled edges does not apply.

... — Extra arguments to pass on to the layout algorithm, if necessary.

Value

A ggplot2::ggplot() object. The last plot can be saved to the file system using ggplot2::ggsave().

See Also

Other mapping: layout_configuration(), layout_partition, plot_graphs, plot_grapht

Examples

```
graphr(ison_adolescents)
ison_adolescents %>%
  mutate(color = rep(c("introvert","extrovert"), times = 4),
         size = ifelse(netrics::node_is_cutpoint(ison_adolescents), 6, 3)) %>%
  mutate_ties(ecolor = rep(c("friends", "acquaintances"), times = 5)) %>%
  graphr(node_color = "color", node_size = "size",
         edge_size = 1.5, edge_color = "ecolor")
graphr(ison_southern_women, labels = TRUE, label_dist = 10)
graphr(ison_southern_women, labels = TRUE, label_repel = FALSE)
graphr(manynet::generate_random(40, 0.1), edge_bundle = TRUE)
```

plot_graphs *Easily graph a set of networks with sensible defaults*

Description

This function provides users with an easy way to graph lists of network data for comparison.

It builds upon this package's `graphr()` function, and inherits all the same features and arguments. See `graphr()` for more. However, it uses the `{patchwork}` package to plot the graphs side by side and, if necessary, in successive rows. This is useful for lists of networks that represent, for example, ego or component subgraphs of a network, or a list of a network's different types of tie or across time. By default just the first and last network will be plotted, but this can be overridden by the "waves" parameter.

Where the graphs are of the same network (same nodes), the graphs may share a layout to facilitate comparison. By default, successive graphs will use the layout calculated for the "first" network, but other options include the "last" layout, or a mix, "both", of them.

Usage

```
graphs(netlist, waves, based_on = c("first", "last", "both"), ...)
```

Arguments

`netlist`	A list of manynet-compatible networks. This can also be a single manynet network object that encodes time, which will be split automatically (as in `grapht()`): longitudinal or changing networks are split into waves via `manynet::to_waves()`; dynamic (time-stamped, event-based) networks such as `manynet::irps_nuclear` into cumulative time slices via `manynet::to_slices()`; and interval (spell) networks that record tie `begin`/`end` lifespans, such as `manynet::irps_wwi`, into one snapshot per change point. It can also be a diffusion model result from e.g. `manynet::play_diffusion()`.
`waves`	Numeric, the number of plots to be displayed side-by-side. If missing, the number of plots will be reduced to the first and last when there are more than four plots. This argument can also be passed a vector selecting the waves to plot.
`based_on`	Whether the layout of the joint plots should be based on the "first" or the "last" network, or "both".
`...`	Additional arguments passed to `graphr()`.

Value

Multiple `ggplot2::ggplot()` objects displayed side-by-side.

See Also

Other mapping: `layout_configuration()`, `layout_partition`, `plot_graphr`, `plot_grapht`

Examples

```
#graphs(to_egos(ison_adolescents))
#graphs(to_egos(ison_adolescents), waves = 8)
#graphs(to_egos(ison_adolescents), waves = c(2, 4, 6))
#graphs(play_diffusion(ison_adolescents))
```

`plot_grapht` *Easily animate dynamic networks with sensible defaults*

Description

This function provides users with an easy way to graph dynamic network data for exploration and presentation.

It builds upon this package's `graphr()` function, and inherits all the same features and arguments. See `graphr()` for more. However, it uses the {gganimate} package to animate the changes between successive iterations of a network. This is useful for networks in which the ties and/or the node or tie attributes are changing, including networks whose node composition changes over time: every node that ever appears is assigned a stable position, and nodes fade in and out in place as they enter and exit the network.

By default node positions transition smoothly between waves using the dynamic stress layout from {graphlayouts} (`graphlayouts::layout_as_dynamic()`), which anchors each wave's layout to a reference layout of the aggregate network. The `alpha` argument controls this trade-off: 0 lets each wave's layout follow that wave's structure freely, while 1 freezes every node at its aggregate position. When another `layout` is requested, a single static layout is computed on the aggregate (union of waves) network instead, so that positions remain constant. Unlike `graphr()`, `grapht()` uses this dynamic stress layout by default even for two-mode networks (rather than a hierarchy layout, which would collapse many nodes onto a line); the two modes remain distinguishable by node shape. For networks with more than 30 nodes, node labels are suppressed by default to keep frames legible; pass `labels = TRUE` to force them.

`grapht()` returns a {ggplot2}-compatible object that can be extended with additional layers such as `ggplot2::labs()`, `ggplot2::theme()`, scale functions, and others, just like plots produced by `graphr()` and `graphs()`. The animation is rendered when the object is printed or displayed. Users who want more control over animation parameters can call `gganimate::animate()` directly on the returned object.

The visual appearance is consistent with `graphr()`: nodes use fillable shapes with the fill aesthetic, the same colour palettes are applied, directed networks receive arrowheads, signed networks distinguish positive from negative ties by linetype, and labels use the current theme font. Legends transition along with the mapped aesthetics.

A progress bar is shown if it takes some time to encode all the .png files into a .gif.

Usage

```
grapht(
  tlist,
  layout = NULL,
```

```
  labels = TRUE,
  node_color,
  node_shape,
  node_size,
  edge_color,
  edge_size,
  isolates = c("keep", "fade"),
  alpha = 0.5,
  label_dist = NULL,
  label_repel = TRUE,
  keep_isolates = NULL,
  ...,
  node_colour,
  edge_colour
)

## S3 method for class 'grapht'
print(x, ...)
```

Arguments

tlist	A manynet-compatible network listed according to a time attribute, waves, or slices. This can also be a single manynet network object that encodes time, which will be split automatically: longitudinal or changing networks are split into waves via `manynet::to_waves()`; dynamic (time-stamped, event-based) networks such as `manynet::irps_nuclear` into cumulative time slices via `manynet::to_slices()`; and interval (spell) networks that record tie begin/end lifespans, such as `manynet::irps_wwi`, into one snapshot per change point showing the ties active in that spell. It can also be a diffusion model result from e.g. `manynet::play_diffusion()`.
layout	An igraph, ggraph, or manynet layout algorithm. If not declared, defaults to "triad" for networks with 3 nodes, "quad" for networks with 4 nodes, "stress" for all other one mode networks, or "hierarchy" for two mode networks. For "hierarchy" layout, one can further split graph by declaring the "center" argument as the "events", "actors", or by declaring a node name. For "concentric" layout algorithm please declare the "membership" as an extra argument. The "membership" argument expects either a quoted node attribute present in data or vector with the same length as nodes to draw concentric circles. For "multilevel" layout algorithm please declare the "level" as extra argument. The "level" argument expects either a quoted node attribute present in data or vector with the same length as nodes to hierarchically order categories. If "level" is missing, function will look for 'lvl' node attribute in data. The "lineage" layout ranks nodes in Y axis according to values. For "lineage" layout algorithm please declare the "rank" as extra argument. The "rank" argument expects either a quoted node attribute present in data or vector with the same length as nodes.
labels	Logical, whether to print node names as labels if present.
node_color, node_colour	Node variable to be used for coloring the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be colored by

declaring a color instead.

node_shape
: Node variable to be used for shaping the nodes. It is easiest if this is added as a node attribute to the graph before plotting. Nodes can also be shaped by declaring a shape instead.

node_size
: Node variable to be used for sizing the nodes. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all node-related statistics prior to using this function. Nodes can also be sized by declaring a numeric size or vector instead.

edge_color, edge_colour
: Tie variable to be used for coloring the nodes. It is easiest if this is added as an edge or tie attribute to the graph before plotting. Edges can also be colored by declaring a color instead.

edge_size
: Tie variable to be used for sizing the edges. This can be any continuous variable on the nodes of the network. Since this function expects this to be an existing variable, it is recommended to calculate all edge-related statistics prior to using this function. Edges can also be sized by declaring a numeric size or vector instead.

isolates
: One of `"keep"` (the default) or `"fade"`. `"keep"` retains isolated nodes at their layout positions in every wave in which they are present. `"fade"` fades nodes out during waves in which they are isolates, and fades them back in when they regain ties. Nodes that are absent from a wave altogether (composition change) always fade out.

alpha
: A number between 0 and 1 controlling the stability of node positions across waves when the default dynamic (stress) layout is used. 0 computes each wave's layout freely, 1 fixes all nodes at their aggregate-network positions. By default 0.5. Passed to `graphlayouts::layout_as_dynamic()`.

label_dist
: Numeric scalar, in points (pt), controlling the extra gap left between labels and node borders – similar to igraph's `vertex.label.dist`. Node size is always accounted for automatically (larger nodes push labels further away without any extra configuration); `label_dist` adds further spacing on top of that, and defaults to a small gap (5pt). Set to `0` for labels right at the node border, or to a larger value (e.g. `15`) for more spacing. Only used when `labels = TRUE` and `label_repel = TRUE` (as the padding passed to the repel algorithm) or `label_repel = FALSE` (as a fixed nudge away from the node, in the layouts where this makes sense, e.g. "circle"/"concentric", "bipartite"/"railway", "alluvial").

label_repel
: Logical scalar, whether labels should be repelled away from each other and from nodes using `ggrepel` (via ggraph's `repel` argument). Defaults to `TRUE`. Set to `FALSE` to place labels at a fixed offset (see `label_dist`) without the (sometimes slow, and non-deterministic between runs for some layouts) repelling algorithm.

keep_isolates
: Deprecated. Use `isolates = "keep"` or `isolates = "fade"` instead.

...
: Extra arguments to pass on to the layout algorithm, if necessary.

x
: A grapht object to print.

Details

Unlike `graphr()`, `grapht()` does not use ggrepel-based label repelling (there is no straightforward way to repel labels consistently across animation frames), so `label_repel` here instead toggles a fixed offset nudging labels away from their nodes, and `label_dist` scales the size of that nudge rather than being used as repel padding.

Some further `graphr()` features are not available in animations: `node_group` hulls, edge bundling, curved arcs for reciprocated ties, and self-loops (loops are not drawn; a note is printed if present).

Value

A {ggplot2}-compatible object with {gganimate} animation layers. This object can be extended with additional {ggplot2} layers (e.g. `+ labs(subtitle = "My subtitle")`). When printed or displayed, the animation is rendered as a .gif. For more control over animation parameters, pass the result to `gganimate::animate()` directly.

Source

https://blog.schochastics.net/posts/2021-09-15_animating-network-evolutions-with-gganimate/

See Also

Other mapping: `layout_configuration()`, `layout_partition`, `plot_graphr`, `plot_graphs`

Examples

```
# A dynamic signed network of shifting European alliances 1872-1918,
# split automatically into snapshots of the ties active in each spell.
# Wrapped in \donttest{} because rendering the animation to a .gif is
# slow, not because the code is unsafe to run.

grapht(irps_wwi)
```

---

plot_interp *Plotting effects interpretation*

Description

These functions support the interpretation of network and behavior effects found in stochastic actor-oriented models. They are S3 plotting methods for objects of class "selectionTable" or "influenceTable", created using `RSiena::selectionTable()` or `RSiena::influenceTable()`, respectively. They plot how the evaluation function for selection or influence changes based on ego's value and alter's value of some covariate. This helps to interpret the effect of that covariate on the network dynamics or behavior dynamics, respectively.

Usage

```
## S3 method for class 'selectionTable'
plot(x, quad = TRUE, separation = 0, ...)

## S3 method for class 'influenceTable'
plot(x, separation = 0, ...)
```

Arguments

x	An object of class "selectionTable" or "influenceTable", created using `RSiena::selectionTable()` or `RSiena::influenceTable()`, respectively.
quad	When TRUE (the default), a quadratic function (average and total alter) is plotted. Use `quad = FALSE` for similarity effects.
separation	This can be used to make the curves visually distinguishable if they overlap too much without it. An advisable value then is, e.g., 0.01.
...	Other arguments to be passed.

Details

These functions were originally written by Tom Snijders, and adapted for use in the `{autograph}` package.

Value

A plot showing how the selection/influence evaluation function changes based on ego's value and alter's value of some covariate.

Author(s)

Tom Snijders

Thanks to Steffen Triebel and Rene Veenstra for corrections.

References

For plotting selection tables, please consult the RSiena manual, Sections 13.1 and 13.3.

For plotting selection tables, please consult the RSiena manual, Sections 13.2 and 13.4.

See Also

Other RSiena: plot_gof

Examples

```
plot(siena_selection)
plot(siena_selection) + ggplot2::scale_colour_discrete()
plot(siena_influence)
```

`theme_match` *Matching colors across palettes*

Description

Sometimes a palette or particular colours are chosen to symbolise or represent a particular idea, such as red for "stop" or green for "go", or to convey some other interpretation. Yet institutional palettes do not necessarily include all colours, which can constrain how interpretable visualisations are under institutional branding requirements. `match_color()` helps to find the closest matching colours in a given palette to one or more input colours.

There is also a helper function, `is_dark()`, to determine whether a color is dark or light, which can be useful when deciding whether to use white or black text on top of a colored background.

Usage

```
match_color(colors, pal)

is_dark(colors)
```

Arguments

`colors`	One or more hexcodes to match with colors from the palette.
`pal`	Optionally, a vector of hexcodes representing a palette in which to find matches. By default, the current theme's qualitative palette is used.

Details

This function uses the Euclidean distance of colours in CIELAB space to those of a target palette to find the closes corresponding colours. It also ensures that each input color is matched to a unique color in the palette. If there are more input colors than unique colors in the palette, an error is returned.

By default, the current theme's qualitative palette is used, but any vector of hexcodes can be passed to the `pal` argument.

Value

A vector of hexcodes the length of the first argument.

Examples

```
match_color("#4575b4")
is_dark(c("#000","#FFF"))
```

theme_set *Setting a consistent theme for all plots*

Description

This function enables plots to be quickly, easily and consistently themed. This is achieved by setting a theme option, usually at the start of an R session, that enables the palette to be used for all autograph-consistent plotting methods. This includes thematic colours for backgrounds, highlights, sequential, divergent and categorical colour schemes. The function sets these palettes to options that are then used by the various plotting functions.

If no theme is specified (i.e. the function is called without argument), the current theme is reported. The default theme is "default". This theme uses a white background, blue and red for highlighting, and a blue-white-red divergent palette. The themes can be changed at any time by calling `stocnet_theme()` or its alias `set_stocnet_theme()` with a different theme name.

Other themes include those based on the colour schemes of various universities, including ETH Zurich, UZH, UNIBE, RUG, and Oxford. Other themes include "bw" for black and white, "crisp" for a high-contrast black and white theme, "neon" for a dark theme with neon highlights, and "rainbow" for a colourful theme. Most themes are designed to be colour-blind safe.

Usage

```
stocnet_theme(theme = NULL)

set_stocnet_theme(theme = NULL)
```

Arguments

`theme`	String naming a theme. By default "default". The following themes are currently available: default, bw, crisp, neon, iheid, ethz, uzh, rug, unibe, oxf, unige, cmu, iast, hwu, rainbow. This string can be capitalised or not.

Value

This function sets the theme and palette(s) to be used across all stocnet packages. The palettes are written to options and held there.

Fonts

Some themes also set a preferred font for use in plots, if available on the system (a check is performed). In some cases, this includes a vector of options to try in sequence. If none of the preferred fonts are available, a sans-serif font is used. If you receive a warning about a missing font when setting a theme, try installing one of the preferred fonts or make sure that the font is available to R using `extrafont::font_import()` and `extrafont::loadfont()`

Custom

If you have specific needs or preferences, you can set your own palettes or overwrite part of an existing one using `options()`. For example, to set a custom base color, you can use: `options(snet_highlight = c("#1b9e77", "#d95f02", "#7570b3"))`. This will set a custom highlight color palette. Similarly, you can set `snet_div` for divergent palettes and `snet_cat` for categorical palettes.

Examples

```
stocnet_theme("default")
plot(netrics::node_by_degree(ison_karateka))
stocnet_theme("uzh")
plot(netrics::node_by_degree(ison_karateka))
```

Index

∗ **MoNAn**
  plot_convergence, 16
  plot_gof, 17
∗ **RSiena**
  plot_gof, 17
  plot_interp, 26
∗ **datasets**
  made_earlier, 9
∗ **ergm**
  plot_convergence, 16
  plot_gof, 17
∗ **mapping**
  layout_configuration, 4
  layout_partition, 6
  plot_graphr, 19
  plot_graphs, 22
  plot_grapht, 23
∗ **themes**
  theme_set, 29

ag_base *(*ag_call*)*, 2
ag_call, 2
ag_divergent *(*ag_call*)*, 2
ag_font *(*ag_call*)*, 2
ag_highlight *(*ag_call*)*, 2
ag_negative *(*ag_call*)*, 2
ag_positive *(*ag_call*)*, 2
ag_qualitative *(*ag_call*)*, 2
ag_sequential *(*ag_call*)*, 2

ergm_gof *(*made_earlier*)*, 9

goldfish_changepoints *(*made_earlier*)*, 9
goldfish_outliers *(*made_earlier*)*, 9
graphr *(*plot_graphr*)*, 19
graphs *(*plot_graphs*)*, 22
grapht *(*plot_grapht*)*, 23

is_dark *(*theme_match*)*, 28

layout_alluvial *(*layout_partition*)*, 6
layout_concentric *(*layout_partition*)*, 6
layout_configuration, 4
layout_configuration(), *8*, *21*, *22*, *26*
layout_dyad *(*layout_configuration*)*, 4
layout_hexad *(*layout_configuration*)*, 4
layout_hierarchy *(*layout_partition*)*, 6
layout_ladder *(*layout_partition*)*, 6
layout_layered, 5
layout_lineage *(*layout_partition*)*, 6
layout_matching, 5
layout_multilevel *(*layout_partition*)*, 6
layout_partition, *4*, 6, *21*, *22*, *26*
layout_pentad *(*layout_configuration*)*, 4
layout_railway *(*layout_partition*)*, 6
layout_tbl_graph_alluvial
        *(*layout_partition*)*, 6
layout_tbl_graph_concentric
        *(*layout_partition*)*, 6
layout_tbl_graph_configuration
        *(*layout_configuration*)*, 4
layout_tbl_graph_dyad
        *(*layout_configuration*)*, 4
layout_tbl_graph_hexad
        *(*layout_configuration*)*, 4
layout_tbl_graph_hierarchy
        *(*layout_partition*)*, 6
layout_tbl_graph_ladder
        *(*layout_partition*)*, 6
layout_tbl_graph_layered
        *(*layout_layered*)*, 5
layout_tbl_graph_lineage
        *(*layout_partition*)*, 6
layout_tbl_graph_matching
        *(*layout_matching*)*, 5
layout_tbl_graph_multilevel
        *(*layout_partition*)*, 6
layout_tbl_graph_pentad
        *(*layout_configuration*)*, 4
layout_tbl_graph_railway

(layout_partition), 6
layout_tbl_graph_tetrad
(layout_configuration), 4
layout_tbl_graph_triad
(layout_configuration), 4
layout_tbl_graph_valence
(layout_valence), 8
layout_tetrad (layout_configuration), 4
layout_triad (layout_configuration), 4
layout_valence, 8
load_ergm_res (plot_convergence), 16

made_earlier, 9
map_measure, 10
map_member, 11
map_motifs, 12
match_color (theme_match), 28
model_mrqap, 13
monan_conv (made_earlier), 9
monan_gof (made_earlier), 9

plot.ag_conv (plot_convergence), 16
plot.ag_gof (plot_gof), 17
plot.changepoints.goldfish
(plot_adequacy), 15
plot.diff_model (plot.diffusion), 13
plot.diffs_model (plot.diffusion), 13
plot.diffusion, 13
plot.ergm (plot_convergence), 16
plot.gof.ergm (plot_gof), 17
plot.gof.stats.monan (plot_gof), 17
plot.influenceTable (plot_interp), 26
plot.learn_model (plot.diffusion), 13
plot.matrix (map_member), 11
plot.netlm (model_mrqap), 13
plot.netlogit (model_mrqap), 13
plot.network_measures (map_measure), 10
plot.network_motif (map_motifs), 12
plot.network_test, 14
plot.node_measure (map_measure), 10
plot.node_member (map_member), 11
plot.node_motif (map_motifs), 12
plot.outliers.goldfish (plot_adequacy),
15
plot.selectionTable (plot_interp), 26
plot.sienaGOF (plot_gof), 17
plot.tie_measure (map_measure), 10
plot.traces.monan (plot_convergence), 16
plot_adequacy, 15
plot_convergence, 16, *18*
plot_gof, *16*, 17, *27*
plot_graphr, *4*, *8*, 19, *22*, *26*
plot_graphs, *4*, *8*, *21*, 22, *26*
plot_grapht, *4*, *8*, *21*, *22*, 23
plot_interp, *18*, 26
print.grapht (plot_grapht), 23

res_manynet_diff (made_earlier), 9
res_migraph_diff (made_earlier), 9
res_migraph_reg (made_earlier), 9
res_migraph_test (made_earlier), 9

set_stocnet_theme (theme_set), 29
siena_gof (made_earlier), 9
siena_influence (made_earlier), 9
siena_selection (made_earlier), 9
stocnet_theme (theme_set), 29

theme_match, 28
theme_set, 29