Package 'GephiForR'

May 7, 2026

Type Package

Title 'Gephi' Network Visualization

Version 0.1.1

Maintainer Julia Manso <gephiforr@gmail.com>

Description Implements key features of 'Gephi' for network visualization, including 'ForceAtlas2' (with LinLog mode), network scaling, and network rotations. It also includes easy network visualization tools such as edge and node color assignment for recreating 'Gephi'-style graphs in R. The package references layout algorithms developed by Jacomy, M., Venturini T., Heymann S., and Bastian M. (2014) <doi:10.1371/journal.pone.0098679> and Noack, A. (2009) <doi:10.48550/arXiv.0807.4052>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports igraph, Rdpack

RdMacros Rdpack

NeedsCompilation no

Author Julia Manso [aut, cre] (ORCID: <https://orcid.org/0009-0004-3227-218X>)

Repository CRAN

Date/Publication 2025-10-14 18:30:02 UTC

Contents

assign_edge_colors *Assign edge colors based on source node colors*

Description

This function assigns colors to edges in an igraph based on the colors of the source nodes, with an optional transparency adjustment.

Usage

```
assign_edge_colors(graph, transparency = 0.4)
```

Arguments

graph	An igraph object. The graph must contain vertex color attributes.
transparency	A numeric value between 0 and 1 indicating the transparency level of the edge colors. Default is 0.4.

Value

The input graph with updated edge color attributes.

Examples

```
library(igraph)

# Creating a sample graph
g <- sample_gnp(10, 0.3)
V(g)$name <- letters[1:10]
V(g)$color <- rainbow(10)

# Assigning edge colors based on source node colors
g <- assign_edge_colors(g, transparency = 0.4)

# Plotting the graph
plot(g, edge.color = E(g)$color, vertex.color = V(g)$color)
```

assign_node_colors *Assign node colors based on a set of attributes*

Description

This function assigns colors to nodes in a graph based on specified attributes.

Usage

```
assign_node_colors(graph, attributes, custom_colors = NULL)
```

Arguments

graph	An igraph object. The graph must contain a vertex attribute name.
attributes	A two-column matrix or data frame. The first column must contain node names, and the second column must contain the attributes that colors will be assigned off of.
custom_colors	A character vector of colors to be used for different attribute values. If not provided, a default palette from rainbow will be used.

Value

The input graph with updated vertex color attributes.

Examples

```
library(igraph)

# Creating a sample graph
g <- sample_gnp(10, 0.3)
V(g)$name <- letters[1:10]

# Creating a sample attributes data frame
attributes <- data.frame(
  Node = letters[1:10],
  Attribute = rep(c("Group1", "Group2", "Group3"), length.out = 10)
)

# Assigning node colors using default colors
g <- assign_node_colors(g, attributes)

# Plotting the graph
plot(g, vertex.color = V(g)$color)

##### Example with custom colors ####

# Defining custom colors
custom_colors <- c("red", "yellow", "pink")

# Assigning node colors
g <- assign_node_colors(g, attributes, custom_colors)

# Plotting the graph
plot(g, vertex.color = V(g)$color)
```

easyplot *Easy way to plot networks similar to 'Gephi'*

Description

This function provides an easy interface for plotting a graph in a similar style to 'Gephi'. It has a customizable layout, label size, edge color, vertex size, edge arrow size, and vertex label color.

Usage

```
easyplot(
  graph,
  layout,
  label_size = 3,
  edge_color = NULL,
  vertex_size = rep(3, vcount(graph)),
  edge_arrow_size = 0.2,
  vertex_label_color = "black"
)
```

Arguments

graph	An igraph object. The graph must contain vertex color attributes.
layout	A matrix representing the layout of the graph. Typically obtained from layout functions like layout_with_fr or layout.forceatlas2.
label_size	A numeric value indicating the size of the vertex labels. Default is 3. Note that when exporting a plot, label size will likely need to be adjusted depending on the plot size.
edge_color	A character vector of colors for the edges. If not provided, the default color is black.
vertex_size	A numeric vector indicating the size of the vertices. Default is 15 for all vertices.
edge_arrow_size	A numeric value indicating the size of the arrows at the end of the edges. Default is 0.2.
vertex_label_color	A character string specifying the color of the vertex labels. Default is "black".

Value

A plot of the graph with the specified parameters.

Examples

```
library(igraph)

# Creating a sample graph
g <- sample_gnp(10, 0.3)
V(g)$name <- letters[1:10]
V(g)$color <- rainbow(10)
layout <- layout_with_fr(g)

# Plot the graph using easyplot
easyplot(g, layout, label_size = 1, vertex_size = rep(10, vcount(g)))

# Assign edge colors based on source node colors
g <- assign_edge_colors(g, transparency = 0.4)

# Plot the graph using easyplot, now with edge color
easyplot(g, layout, label_size = 1, vertex_size = rep(10, vcount(g)))
```

---

layout.forceatlas2 *Apply ForceAtlas2 layout to a graph*

Description

This function applies Jacomy et al. (2014)'s 'ForceAtlas2' layout algorithm to an `igraph` object.

Usage

```
layout.forceatlas2(
  g,
  iterations = 100,
  linlog = FALSE,
  pos = NULL,
  gravity = 1,
  center = NULL,
  plotstep = 10,
  plotlabels = TRUE,
  scalingratio = 10,
  stronggravity = FALSE,
  jittertol = 1
)
```

Arguments

`g`	An `igraph` object representing the graph.
`iterations`	Integer. The number of iterations to run the algorithm. Default is 100.

`linlog`	Logical. If `linlog = TRUE`, uses the Noack LinLog model implemented for 'Gephi' to calculate attractive and repulsive forces (see Noack 2009). Default is `linlog = FALSE`.
`pos`	A 2-column matrix of initial positions, where the columns contain x-coordinates and y-coordinates, respectively. If `pos = NULL`, positions for the first iteration are generated randomly. Default is `pos = NULL`.
`gravity`	Numeric. The strength of the gravity force. Default is 1. Note that this is only included in calculations if `stronggravity = TRUE`. Higher gravity values result in tighter networks.
`center`	A numeric vector of length 2 specifying the center of gravity. If `center = NULL`, the center is calculated automatically. Default is `center = NULL`.
`plotstep`	Integer. The number of iterations between plots. If `plotstep = 0`, no plotting is done. Default is `plotstep = 10`. These plots appear as intermediate output in the console.
`plotlabels`	Logical. If `plotlabels = TRUE`, plot node labels appear during the intermediate plotstep graphs. Default is `plotlabels = TRUE`.
`scalingratio`	Numeric. The scaling ratio of the layout. Default is 10, in line with 'Gephi'.
`stronggravity`	Logical. If `stronggravity = TRUE`, gravity will be an additional force acting on each node, and the gravity parameter kicks in. Default is `stronggravity = FALSE`.
`jittertol`	Numeric. The tolerance for jittering nodes. Default is `jittertol = 1`; Jacomy et al. (2014) do not recommend increasing it above 1.

Details

This function implements Jacomy et al. (2014)'s ForceAtlas2 layout algorithm on an `igraph` object. It can handle large graphs and is particularly suitable for visualizing networks. It also includes LinLog mode and a stronger gravity feature, like 'Gephi'.

Value

A matrix of node positions.

References

Jacomy M, Venturini T, Heymann S, Bastian M (2014). "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software." *PLoS ONE*, **9**(6). doi:10.1371/journal.pone.0098679, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0098679.
Noack A (2009). "Modularity clustering is force-directed layout." *Physical Review*, **79**. https://arxiv.org/pdf/0807.4052.

Examples

```
# Create a random graph
library(igraph)
g <- sample_gnp(100, 0.05)
```

```
# Assign non-numeric row names
V(g)$name <- paste0("node", 1:vcount(g))

# Apply ForceAtlas2 layout
pos <- layout.forceatlas2(g, linlog = TRUE, iterations = 100, scalingratio = 10)

V(g)$x = as.numeric(pos[,1])
V(g)$y = as.numeric(pos[,2])

# Plotting (preserves the exact degree of rotation)
igraph::plot.igraph(g, rescale = FALSE,
                    xlim = range(pos[,1]), ylim = range(pos[,2]))
```

`rotate_layout` *Rotate layout positions by a custom angle*

Description

This function rotates each node's position by a specified angle.

Usage

```
rotate_layout(layout, angle)
```

Arguments

`layout`	A 2-column matrix or dataframe of position data, where the columns are the x- and y-coordinates of each node, respectively.
`angle`	The angle by which to rotate the layout, in degrees.

Details

This function rotates each node position in a 2-column matrix/dataframe of position data by a specified angle.

Value

A matrix of node positions.

Examples

```
# Create a random graph
library(igraph)
g <- sample_gnp(6, 0.05)

# Initializing position vector and plotting
position <- as.matrix(data.frame(X = c(1, 2, 3, 4, 5, 3), Y = c(4, 5, 6, 7, 1, 2)))
```

```
plot(g, layout = position)

# Rotating position vector 90 degrees
rotated_df <- rotate_layout(position, 90)

# Updating node position in the graph object
V(g)$x = as.numeric(rotated_df[,1])
V(g)$y = as.numeric(rotated_df[,2])

# Plotting (preserves the exact degree of rotation)
igraph::plot.igraph(g, rescale = FALSE,
                    xlim = range(rotated_df[,1]), ylim = range(rotated_df[,2]))

# Alternatively, GephiForR's easyplot can be used if nodes are assigned a color
V(g)$color <- rainbow(6)
easyplot(g, rotated_df)

# Rotating position vector 283 degrees and plotting
rotated_df <- rotate_layout(position, 283)

# Updating node position in the graph object
V(g)$x = as.numeric(rotated_df[,1])
V(g)$y = as.numeric(rotated_df[,2])

# Plotting (preserves the exact degree of rotation)
igraph::plot.igraph(g, rescale = FALSE,
                    xlim = range(rotated_df[,1]), ylim = range(rotated_df[,2]))
```

`scale_node_positions` *Scale node positions in an* `igraph` *object*

Description

This function scales the positions of nodes in a graph relative to their centroid; compatible with expansions and contractions.

Usage

```
scale_node_positions(layout, scale_factor = 1.2)
```

Arguments

`layout`	A 2-column matrix representing the position data, where the columns are the x- and y-coordinates of each node, respectively.
`scale_factor`	A numeric value indicating the factor by which to scale the node positions. The default is `scale_factor = 1.2`.

Details

This function expands or contracts the graph around a centroid, facilitating visualization. Factors greater than 1 expand the network around the centroid while those less than 1 contract it around the centroid. `scale_factor = 1` is no change. Note that unscaled plot commands like `plot(g, layout = layout_expanded)` make the change difficult to see, so scaled plots are recommended for comparison.

Value

A matrix of updated node positions.

Examples

```
set.seed(10)
library(igraph)

# Generating graph and setting initial layout
g <- sample_gnp(100, 0.05)
layout <- layout_with_fr(g)

# Plotting original graph, maintaining fixed scale so expansion is clear
plot(layout, main="Original Graph", xlab="", ylab="", xlim=c(-20, 15), ylim=c(-20, 15))

# Expanding node positions
layout_expanded <- scale_node_positions(layout, 2)

# Plotting expanded graph, maintaining fixed scale so expansion is clear
plot(layout_expanded, main="Expanded Graph", xlab="", ylab="", xlim=c(-20, 15), ylim=c(-20, 15))

# Contract node positions
layout_cont <- scale_node_positions(layout, 0.8)

# Plotting contracted graph, maintaining fixed scale so transformation is clear
plot(layout_cont, main="Contracted Graph", xlab="", ylab="", xlim=c(-20, 15), ylim=c(-20, 15))


# Note that igraph plots like below make it difficult to see the transformation,
# because they are autoscaled.
# plot(g, layout = layout_expanded)
# The change is easy to see in scaled plots, as shown.
```

Index