Package 'ggraph'

July 7, 2018

Type Package

Title An Implementation of Grammar of Graphics for Graphs and Networks

Version 1.0.2

Date 2018-07-07

Author Thomas Lin Pedersen

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description The grammar of graphics as implemented in ggplot2 is a poor fit for graph and network visualizations due to its reliance on tabular data input. ggraph is an extension of the ggplot2 API tailored to graph visualizations and provides the same flexible approach to building up plots layer by layer.

License GPL-3

LazyData TRUE

Encoding UTF-8

Imports Rcpp (>= 0.12.2), dplyr, plyr, ggforce, grid, igraph (>= 1.0.0), scales, MASS, digest, gtable, ggrepel, utils, stats, viridis

Suggests network, knitr, rmarkdown

LinkingTo Rcpp

RoxygenNote 6.0.1.9000

Depends R (>= 2.10), ggplot2 (>= 2.0.0)

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-07-07 21:20:03 UTC

R topics documented:

`ggraph-package` *ggraph: Grammar of Graph Graphics*

Description

This package is an extension of the grammar of graphics defined in `ggplot2` for relational data such as networks and trees. `ggraph` adds a long list of new geoms and scales as well as a new plot constructor that helps you build up your network visualization layer by layer in the same manner as you know from `ggplot2`.

Author(s)

Thomas Lin Pedersen <thomasp85@gmail.com>

See Also

Useful links:

Development version https://github.com/thomasp85/ggraph

Feature requests and bug reports https://github.com/thomasp85/ggraph/issues

---

`den_to_igraph` *Convert a dendrogram into an igraph object*

Description

This small helper function converts a dendrogram into an igraph object with the same node indexes as would be had were the dendrogram used directly in a ggraph plot. The nodes would have the same attributes as would have been calculated had the dendrogram been used in layout creation, meaning that e.g. it contains a leaf attribute which is `TRUE` for leaf nodes and `FALSE` for the rest.

Usage

```
den_to_igraph(den, even = FALSE, ...)
```

Arguments

`den`	A dendrogram object
`even`	Logical should the position information be calculated based on an even layout (see `layout_dendrogram_even`).
`...`	Additional parameters to pass off to `layout_dendrogram_dendrogram`

Value

An igraph object.

`facet_edges` *Create small multiples based on edge attributes*

Description

This function is equivalent to `facet_wrap` but only facets edges. Nodes are repeated in every panel.

Usage

```
facet_edges(facets, nrow = NULL, ncol = NULL, scales = "fixed",
  shrink = TRUE, labeller = "label_value", as.table = TRUE,
  switch = NULL, drop = TRUE, dir = "h", strip.position = "top")
```

Arguments

`facets`	A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, '~a • `b, or a character vector,`c("a", "b")'.
`nrow`	Number of rows and columns.
`ncol`	Number of rows and columns.
`scales`	Should scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`)?
`shrink`	If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.
`labeller`	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type `~cyl + am`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. See `label_value()` for more details and pointers to other options.
`as.table`	If `TRUE`, the default, the facets are laid out like a table with highest values at the bottom-right. If `FALSE`, the facets are laid out like a plot with the highest value at the top-right.
`switch`	By default, the labels are displayed on the top and right of the plot. If `"x"`, the top labels will be displayed to the bottom. If `"y"`, the right-hand side labels will be displayed to the left. Can also be set to `"both"`.

drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.
strip.position	By default, the labels are displayed on the top of the plot. Using strip.position it is possible to place the labels on either of the four sides by setting strip.position = c("top", "bott

See Also

Other ggraph-facets: facet_graph, facet_nodes

Examples

```
library(igraph)
gr <- graph_from_data_frame(highschool)

ggraph(gr) +
    geom_edge_link() +
    geom_node_point() +
    facet_edges(~year)
```

---

facet_graph *Create a grid of small multiples by node and/or edge attributes*

Description

This function is equivalent to facet_grid in that it allows for building a grid of small multiples where rows and columns correspond to a specific data value. While facet_grid could be used it would lead to unexpected results as it is not possible to specify whether you are refering to a node or an edge attribute. Furthermore facet_grid will draw edges in panels even though the panel does not contain both terminal nodes. facet_graph takes care of all of these issues, allowing you to define which data type the rows and columns are referencing as well as filtering the edges based on the nodes in each panel (even when nodes are not drawn).

Usage

```
facet_graph(facets, row_type = "edge", col_type = "node", margins = FALSE,
  scales = "fixed", space = "fixed", shrink = TRUE,
  labeller = "label_value", as.table = TRUE, switch = NULL, drop = TRUE)
```

Arguments

facets	This argument is soft-deprecated, please us rows and cols instead.
row_type, col_type	Either 'node' or 'edge'. Which data type is being facetted in the rows and columns. Default is to facet on nodes column wise and on edges row wise.

margins	Either a logical value or a character vector. Margins are additional facets which contain all the data for each of the possible values of the faceting variables. If FALSE, no additional facets are included (the default). If TRUE, margins are included for all faceting variables. If specified as a character vector, it is the names of variables for which margins are to be created.
scales	Are scales shared across all facets (the default, "fixed"), or do they vary across rows ("free_x"), columns ("free_y"), or both rows and columns ("free")?
space	If "fixed", the default, all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
switch	By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.

See Also

Other ggraph-facets: facet_edges, facet_nodes

Examples

```
library(igraph)
gr <- graph_from_data_frame(highschool)
V(gr)$popularity <- as.character(cut(degree(gr, mode = 'in'), breaks = 3,
                                     labels = c('low', 'medium', 'high')))
ggraph(gr) +
    geom_edge_link() +
    geom_node_point() +
    facet_graph(year~popularity)
```

facet_nodes *Create small multiples based on node attributes*

Description

This function is equivalent to facet_wrap but only facets nodes. Edges are drawn if their terminal nodes are both present in a panel.

Usage

```
facet_nodes(facets, nrow = NULL, ncol = NULL, scales = "fixed",
  shrink = TRUE, labeller = "label_value", as.table = TRUE,
  switch = NULL, drop = TRUE, dir = "h", strip.position = "top")
```

Arguments

facets	A set of variables or expressions quoted by vars() and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to labeller). For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, '~a • b, or a character vector,c("a", "b")'.
nrow	Number of rows and columns.
ncol	Number of rows and columns.
scales	Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?
shrink	If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
labeller	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with formulae of the type ~cyl + am. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with labeller(). See label_value() for more details and pointers to other options.
as.table	If TRUE, the default, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
switch	By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".
drop	If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
dir	Direction: either "h" for horizontal, the default, or "v", for vertical.

`strip.position` By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bott`

See Also

Other ggraph-facets: `facet_edges`, `facet_graph`

Examples

```
library(igraph)
gr <- graph_from_data_frame(highschool)
V(gr)$popularity <- as.character(cut(degree(gr, mode = 'in'), breaks = 3,
                                     labels = c('low', 'medium', 'high')))
ggraph(gr) +
    geom_edge_link() +
    geom_node_point() +
    facet_nodes(~popularity)
```

---

flare *The class hierarchy of the flare visualization library*

Description

This dataset contains the graph that describes the class hierarchy for the Flare ActionScript visualization library. It contains both the class hierarchy as well as the import connections between classes. This dataset has been used extensively in the D3.js documentation and examples and are included here to make it easy to redo the examples in ggraph.

Usage

```
flare
```

Format

A list of three data.frames describing the software structure of flare:

edges This data.frame maps the heirarchical structure of the class hierarchy as an edgelist, with the class in `from` being the superclass of the class in `to`.

vertices This data.frame gives additional information on the classes. It contains the full name, size and short name of each class.

imports This data.frame contains the class imports for each class implementation. The `from` column gives the importing class and the `to` column gives the import.

Source

The data have been adapted from the JSON downloaded from `https://gist.github.com/mbostock/1044242#file-readme-flare-imports-json` courtesy of Mike Bostock. The Flare framework is the work of the UC Berkeley Visualization Lab.

`geometry` *Define simple shapes for line capping*

Description

This set of functions makes it easy to define shapes at the terminal points of edges that are used to shorten the edges. The shapes themselves are not drawn, but the edges will end at the boundary of the shape rather than at the node position. This is especially relevant when drawing arrows at the edges as the arrows will be partly obscured by the node unless the edge is shortened. Edge shortening is dynamic and will update as the plot is resized, making sure that the capping remains at an absolute distance to the end point.

Usage

```
geometry(type = "circle", width = 1, height = width, width_unit = "cm",
  height_unit = width_unit)

circle(radius = 1, unit = "cm")

square(length = 1, unit = "cm")

ellipsis(a = 1, b = 1, a_unit = "cm", b_unit = a_unit)

rectangle(width = 1, height = 1, width_unit = "cm",
  height_unit = width_unit)

label_rect(label, padding = margin(1, 1, 1.5, 1, "mm"), ...)

is.geometry(x)
```

Arguments

`type`	The type of geometry to use. Currently `'circle'` and `'rect'` is supported.
`width, height, length, radius, a, b`	The dimensions of the shape.
`unit, width_unit, height_unit, a_unit, b_unit`	The unit for the numbers given.
`label`	The text to be enclosed
`padding`	extra size to be added around the text using the `margin` function
`...`	Passed on to `gpar`
`x`	An object to test for geometry inheritance

Details

geometry is the base constructor, while the rest are helpers to save typing. circle creates circles width a given radius, square creates squares at a given side length, ellipsis creates ellipses with given a and b values (width and height radii), and rectangle makes rectangles of a given width and height. label_rect is a helper that, given a list of strings and potentially formatting options creates a rectangle that encloses the string.

Value

A geometry object encoding the specified shape.

Examples

```
geometry(c('circle', 'rect', 'rect'), 1:3, 3:1)

circle(1:4, 'mm')

label_rect(c('some', 'different', 'words'), fontsize = 18)
```

---

geom_axis_hive *Draw rectangular bars and labels on hive axes*

Description

This function lets you annotate the axes in a hive plot with labels and color coded bars.

Usage

```
geom_axis_hive(mapping = NULL, data = NULL, position = "identity",
  label = TRUE, axis = TRUE, show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.

label	Should the axes be labelled. Defaults to TRUE
axis	Should a rectangle be drawn along the axis. Defaults to TRUE
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

geom_axis_hive understand the following aesthetics.

- alpha
- colour
- fill
- size
- linetype
- label_size
- family
- fontface
- lineheight

Author(s)

Thomas Lin Pedersen

Examples

```
# Plot the flare import graph as a hive plot
library(igraph)
flareGr <- graph_from_data_frame(flare$imports)
# Add some metadata to divide nodes by
V(flareGr)$type <- 'Both'
V(flareGr)$type[degree(flareGr, mode = 'in') == 0] <- 'Source'
V(flareGr)$type[degree(flareGr, mode = 'out') == 0] <- 'Sink'
analyticsNodes <- grep('flare.analytics', V(flareGr)$name)
E(flareGr)$type <- 'Other'
E(flareGr)[inc(analyticsNodes)]$type <- 'Analytics'
ggraph(flareGr, 'hive', axis = 'type') +
    geom_edge_hive(aes(colour = type), edge_alpha = 0.1) +
    geom_axis_hive(aes(colour = type)) +
    coord_fixed()
```

geom_conn_bundle *Create heirarchical edge bundles between node connections*

Description

Hierarchical edge bundling is a technique to introduce some order into the hairball structure that can appear when there's a lot of overplotting and edge crossing in a network plot. The concept requires that the network has an intrinsic hierarchical structure that defines the layout but is not shown. Connections between points (that is, not edges) are then drawn so that they loosely follows the underlying hierarchical structure. This results in a flow-like structure where lines that partly move in the same direction will be bundled togeether.

Usage

```
geom_conn_bundle(mapping = NULL, data = get_con(), position = "identity",
  arrow = NULL, lineend = "butt", show.legend = NA, n = 100,
  tension = 0.8, ...)

geom_conn_bundle2(mapping = NULL, data = get_con(), position = "identity",
  arrow = NULL, lineend = "butt", show.legend = NA, n = 100,
  tension = 0.8, ...)

geom_conn_bundle0(mapping = NULL, data = get_con(), position = "identity",
  arrow = NULL, lineend = "butt", show.legend = NA, tension = 0.8, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The result of a call to get_con
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
lineend	Line end style (round, butt, square).
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
n	The number of points to create along the path.
tension	How "loose" should the bundles be. 1 will give very tight bundles, while 0 will turn of bundling completely and give straight lines. Defaults to 0.8
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

geom_conn_bundle* understands the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- **circular**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

References

Holten, D. (2006). *Hierarchical edge bundles: visualization of adjacency relations in hierarchical data.* IEEE Transactions on Visualization and Computer Graphics, **12**(5), 741-748. http://doi.org/10.1109/TVCG.2006.147

Examples

```
# Create a graph of the flare class system
library(igraph)
flareGraph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)
importFrom <- match(flare$imports$from, flare$vertices$name)
importTo <- match(flare$imports$to, flare$vertices$name)
flareGraph <- tree_apply(flareGraph, function(node, parent, depth, tree) {
    tree <- set_vertex_attr(tree, 'depth', node, depth)
    if (depth == 1) {
        tree <- set_vertex_attr(tree, 'class', node, V(tree)$shortName[node])
    } else if (depth > 1) {
        tree <- set_vertex_attr(tree, 'class', node, V(tree)$class[parent])
    }
    tree
```

```
})
V(flareGraph)$leaf <- degree(flareGraph, mode = 'out') == 0

# Use class inheritance for layout but plot class imports as bundles
ggraph(flareGraph, 'dendrogram', circular = TRUE) +
    geom_conn_bundle(aes(colour = ..index..), data = get_con(importFrom, importTo),
                     edge_alpha = 0.25) +
    geom_node_point(aes(filter = leaf, colour = class)) +
    scale_edge_colour_distiller('', direction = 1, guide = 'edge_direction') +
    coord_fixed() +
    ggforce::theme_no_axes()
```

---

geom_edge_arc *Draw edges as Arcs*

Description

This geom is mainly intended for arc linear and circular diagrams (i.e. used together with layout_igraph_linear), though it can be used elsewere. It draws edges as arcs with a hight proportional to the distance between the nodes. Arcs are calculated as beziers. For linear layout the placement of control points are related to the curvature argument and the distance between the two nodes. For circular layout the control points are placed on the same angle as the start and end node at a distance related to the distance between the nodes.

Usage

```
geom_edge_arc(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, curvature = 1, n = 100, fold = FALSE,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_arc2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, curvature = 1, n = 100,
  fold = FALSE, lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_arc0(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, curvature = 1, lineend = "butt", show.legend = NA,
  fold = fold, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
curvature	The bend of the curve. 1 approximates a halfcircle while 0 will give a straight line. Negative number will change the direction of the curve. Only used if circular = FALSE.
n	The number of points to create along the path.
fold	Logical. Should arcs appear on the same side of the nodes despite different directions. Default to FALSE.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (`n`) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. `colour = ..index..` The version postfixed with a "2" uses the "long" edge format (see `get_edges`) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a `geometry` specification and dynamically caps the termini of the edges based on the given specifications. This means that if `end_cap = circle(1, 'cm')` the edges will end at a distance of 1cm even during resizing of the plot window.

All `geom_edge_*` and `geom_edge_*2` have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_arc and geom_edge_arc0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- **circular**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_arc2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**

- **y**
- **group**
- **circular**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_arc and geom_edge_arc2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_link`, `geom_edge_loop`

Examples

```
require(igraph)
# Make a graph with different directions of edges
gr <- graph_from_edgelist(
  t(apply(as_edgelist(make_graph('Meredith')), 1, sample))
)
E(gr)$class <- sample(letters[1:3], gsize(gr), replace = TRUE)
V(gr)$class <- sample(letters[1:3], gorder(gr), replace = TRUE)

ggraph(gr, 'linear') +
  geom_edge_arc(aes(alpha = ..index..))

ggraph(gr, 'linear') +
  geom_edge_arc2(aes(colour = node.class), curvature = 0.6)

ggraph(gr, 'linear', circular = TRUE) +
  geom_edge_arc0(aes(colour = class))
```

---

geom_edge_density *Show edges as a density map*

Description

This geom makes it possible to add a layer showing edge presence as a density map. Each edge is converted to n points along the line and a jitter is applied. Based on this dataset a two-dimensional kernel density estimation is applied and plotted as a raster image. The density is mapped to the alpha level, making it possible to map a variable to the fill.

Usage

```
geom_edge_density(mapping = NULL, data = get_edges("short"),
  position = "identity", show.legend = NA, n = 100, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

n	The number of points to estimate in the x and y direction, i.e. the resolution of the raster.
...	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Aesthetics

geom_edge_density understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- edge_fill
- filter

Computed variables

x, y The coordinates for each pixel in the raster

density The density associated with the pixel

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_link`, `geom_edge_loop`

Examples

```
require(igraph)
gr <- make_graph('bull')
E(gr)$class <- sample(letters[1:3], gsize(gr), replace = TRUE)

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_density(aes(fill = class)) +
  geom_edge_link() + geom_node_point()
```

`geom_edge_diagonal` *Draw edges as diagonals*

Description

This geom draws edges as diagonal bezier curves as known from d3.svg.diagonal. A diagonal in this context is a quadratic bezier with the control points positioned halfway between the start and end points but on the same axis. This produces a pleasing fan-in, fan-out line that is mostly relevant for heirarchical layouts as it implies an overall directionality in the plot.

Usage

```
geom_edge_diagonal(mapping = NULL, data = get_edges(),
  position = "identity", arrow = NULL, flipped = FALSE, n = 100,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_diagonal2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, flipped = FALSE, n = 100,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_diagonal0(mapping = NULL, data = get_edges(),
  position = "identity", arrow = NULL, flipped = FALSE,
  lineend = "butt", show.legend = NA, ...)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes` or `aes_`. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
`data`	The return of a call to `get_edges()` or a data.frame giving edges in corrent format (see details for for guidance on the format). See `get_edges` for more details on edge extraction.
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`arrow`	Arrow specification, as created by `grid::arrow()`.
`flipped`	Logical, Has the layout been flipped by reassigning the mapping of x, y etc?
`n`	The number of points to create along the path.
`lineend`	Line end style (round, butt, square).
`linejoin`	Line join style (round, mitre, bevel).

linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (n) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. colour = ..index.. The version postfixed with a "2" uses the "long" edge format (see get_edges) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a geometry specification and dynamically caps the termini of the edges based on the given specifications. This means that if end_cap = circle(1, 'cm') the edges will end at a distance of 1cm even during resizing of the plot window.

All geom_edge_* and geom_edge_*2 have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot

dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_diagonal and geom_edge_diagonal0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- **circular**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_diagonal2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- **circular**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_diagonal and geom_edge_diagonal2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_link`, `geom_edge_loop`

Examples

```
require(igraph)
gr <- make_tree(20, 4, 'out')
E(gr)$class <- sample(letters[1:3], gsize(gr), replace = TRUE)
V(gr)$class <- sample(letters[1:3], gorder(gr), replace = TRUE)

ggraph(gr, 'igraph', algorithm = 'tree') +
  geom_edge_diagonal(aes(alpha = ..index..))

ggraph(gr, 'igraph', algorithm = 'tree') +
  geom_edge_diagonal2(aes(colour = node.class))

ggraph(gr, 'igraph', algorithm = 'tree') +
  geom_edge_diagonal0(aes(colour = class))
```

---

geom_edge_elbow *Draw edges as elbows*

Description

This geom draws edges as an angle in the same manner as known from classic dendrogram plots of hierarchical clustering results. In case a circular transformation has been applied the first line segment will be drawn as an arc as expected. This geom is only applicable to layouts that return a direction for the edges (currently `layout_dendrogram_dendrogram` and `layout_dendrogram_dendrogram`).

Usage

```
geom_edge_elbow(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, flipped = FALSE, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, show.legend = NA, ...)

geom_edge_elbow2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, flipped = FALSE, n = 100,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_elbow0(mapping = NULL, data = get_edges(),
  position = "identity", arrow = NULL, flipped = FALSE,
  lineend = "butt", show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
flipped	Logical, Has the layout been flipped by reassigning the mapping of x, y etc?
n	The number of points to create along the path.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.

force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (n) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. colour = ..index.. The version postfixed with a "2" uses the "long" edge format (see get_edges) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a geometry specification and dynamically caps the termini of the edges based on the given specifications. This means that if end_cap = circle(1, 'cm') the edges will end at a distance of 1cm even during resizing of the plot window.

All geom_edge_* and geom_edge_*2 have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_elbow and geom_edge_elbow0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**

- **xend**
- **yend**
- **circular**
- **direction**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_elbow2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- **circular**
- **direction**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_elbow and geom_edge_elbow2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_link`, `geom_edge_loop`

Examples

```
# Can take some seconds to compute
## Not run:
irisDen <- as.dendrogram(
  hclust(dist(iris[1:4],
              method='euclidean', ),
         method='ward.D2')
)
irisDen <- dendrapply(irisDen, function(x) {
  attr(x, 'nodePar') <- list(class = sample(letters[1:3], 1))
  attr(x, 'edgePar') <- list(class = sample(letters[1:3], 1))
  x
})

ggraph(irisDen, 'even', circular = TRUE) +
  geom_edge_elbow(aes(alpha = ..index..))

ggraph(irisDen, 'even') +
  geom_edge_elbow2(aes(colour = node.class))

ggraph(irisDen, 'dendrogram') +
  geom_edge_elbow0(aes(colour = class))

## End(Not run)
```

`geom_edge_fan` *Draw edges as curves of different curvature*

Description

This geom draws edges as cubic beziers with the control point positioned half-way between the nodes and at an angle dependent on the presence of parallel edges. This results in parallel edges being drawn in a non-overlapping fashion resembling the standard approach used in `plot.igraph`. Before calculating the curvature the edges are sorted by direction so that edges goind the same way will be adjacent. This geom is currently the only choice for non-simple graphs if edges should not be overplotted.

Usage

```
geom_edge_fan(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, spread = 1, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, show.legend = NA, ...)

geom_edge_fan2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, spread = 1, n = 100,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_fan0(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, spread = 1, lineend = "butt", show.legend = NA, ...)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes` or `aes_`. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
`data`	The return of a call to `get_edges()` or a data.frame giving edges in corrent format (see details for for guidance on the format). See `get_edges` for more details on edge extraction.
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`arrow`	Arrow specification, as created by `grid::arrow()`.
`spread`	Modify the width of the fans `spread > 1` will create wider fans while the reverse will make them more narrow.
`n`	The number of points to create along the path.
`lineend`	Line end style (round, butt, square).
`linejoin`	Line join style (round, mitre, bevel).
`linemitre`	Line mitre limit (number greater than 1).
`label_colour`	The colour of the edge label. If `NA` it will use the colour of the edge.

label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (n) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. colour = ..index.. The version postfixed with a "2" uses the "long" edge format (see get_edges) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a geometry specification and dynamically caps the termini of the edges based on the given specifications. This means that if end_cap = circle(1, 'cm') the edges will end at a distance of 1cm even during resizing of the plot window.

All geom_edge_* and geom_edge_*2 have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_fan and geom_edge_fan0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- **from**
- **to**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_fan2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- **from**
- **to**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_fan and geom_edge_fan2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_hive`, `geom_edge_link`, `geom_edge_loop`

Examples

```
require(igraph)
gr <- graph_from_data_frame(data.frame(
  from = c(1, 1, 1, 1, 1, 2, 2, 2),
  to = c(2, 2, 2, 2, 2, 1, 1, 1),
  class = sample(letters[1:3], 8, TRUE)
))
V(gr)$class <- c('a', 'b')

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_fan(aes(alpha = ..index..))

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_fan2(aes(colour = node.class))

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_fan0(aes(colour = class))
```

---

geom_edge_hive *Draw edges in hive plots*

Description

This geom is only intended for use together with the hive layout. It draws edges between nodes as bezier curves, with the control points positioned at the same radi as the start or end point, and at a distance defined by the curvature argument.

Usage

```
geom_edge_hive(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, curvature = 0.5, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, show.legend = NA, ...)

geom_edge_hive2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, curvature = 0.5, n = 100,
  lineend = "butt", linejoin = "round", linemitre = 1,
  label_colour = "black", label_alpha = 1, label_parse = FALSE,
  check_overlap = FALSE, angle_calc = "rot", force_flip = TRUE,
  label_dodge = NULL, label_push = NULL, show.legend = NA, ...)

geom_edge_hive0(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, curvature = 0.5, lineend = "butt", show.legend = NA,
  ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
curvature	The curvature of the bezier. Defines the distance from the control points to the midpoint between the start and end node. 1 means the control points are positioned midway between the nodes, while 0 means it coincide with the nodes (resulting in straight lines)
n	The number of points to create along the path.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.

angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (n) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. colour = ..index.. The version postfixed with a "2" uses the "long" edge format (see get_edges) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a geometry specification and dynamically caps the termini of the edges based on the given specifications. This means that if end_cap = circle(1, 'cm') the edges will end at a distance of 1cm even during resizing of the plot window.

All geom_edge_* and geom_edge_*2 have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_hive and geom_edge_hive0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_hive2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_hive and geom_edge_hive2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_link`, `geom_edge_loop`

Examples

```
# Plot the flare import graph as a hive plot
library(igraph)
flareGr <- graph_from_data_frame(flare$imports)

# Add some metadata to divide nodes by
V(flareGr)$type <- 'Both'
V(flareGr)$type[degree(flareGr, mode = 'in') == 0] <- 'Source'
V(flareGr)$type[degree(flareGr, mode = 'out') == 0] <- 'Sink'

analyticsNodes <- grep('flare.analytics', V(flareGr)$name)
E(flareGr)$type <- 'Other'
E(flareGr)[inc(analyticsNodes)]$type <- 'Analytics'

ggraph(flareGr, 'hive', axis = 'type') +
  geom_edge_hive(aes(colour = type), edge_alpha = 0.1) +
  coord_fixed()
```

---

geom_edge_link *Draw edges as straight lines between nodes*

Description

This geom draws edges in the simplest way - as straight lines between the start and end nodes. Not much more to say about that...

Usage

```
geom_edge_link(mapping = NULL, data = get_edges("short"),
  position = "identity", arrow = NULL, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
```

```
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, show.legend = NA, ...)

geom_edge_link2(mapping = NULL, data = get_edges("long"),
  position = "identity", arrow = NULL, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, show.legend = NA, ...)

geom_edge_link0(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, lineend = "butt", show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
n	The number of points to create along the path.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

... Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (`n`) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. `colour = ..index..` The version postfixed with a "2" uses the "long" edge format (see `get_edges`) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a `geometry` specification and dynamically caps the termini of the edges based on the given specifications. This means that if `end_cap = circle(1, 'cm')` the edges will end at a distance of 1cm even during resizing of the plot window.

All `geom_edge_*` and `geom_edge_*2` have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_link and geom_edge_link0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **xend**
- **yend**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_link2 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **group**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_link and geom_edge_link2 furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_loop`

Examples

```
require(igraph)
gr <- make_graph('bull')
E(gr)$class <- sample(letters[1:3], gsize(gr), replace = TRUE)
V(gr)$class <- sample(letters[1:3], gorder(gr), replace = TRUE)

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_link(aes(alpha = ..index..))

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_link2(aes(colour = node.class))

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_link0(aes(colour = class))
```

---

geom_edge_loop *Draw edges as diagonals*

Description

This geom draws edge loops (edges starting and ending at the same node). Loops are drawn as bezier curves starting and ending at the position of the node and with control points protruding at an angle and in a direction specified in the call. As the start and end node is always the same no *2 method is provided. Loops can severely clutter up your visualization which is why they are decoupled from the other edge drawings. Only plot them if they are of importance. If the graph doesn't contain any loops the geom adds nothing silently.

Usage

```
geom_edge_loop(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, n = 100, lineend = "butt", linejoin = "round",
  linemitre = 1, label_colour = "black", label_alpha = 1,
  label_parse = FALSE, check_overlap = FALSE, angle_calc = "rot",
  force_flip = TRUE, label_dodge = NULL, label_push = NULL,
  show.legend = NA, ...)

geom_edge_loop0(mapping = NULL, data = get_edges(), position = "identity",
  arrow = NULL, lineend = "butt", show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, xend, yend, group and circular are mapped to x, y, xend, yend, edge.id and circular in the edge data.
data	The return of a call to get_edges() or a data.frame giving edges in corrent format (see details for for guidance on the format). See get_edges for more details on edge extraction.

position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
arrow	Arrow specification, as created by grid::arrow().
n	The number of points to create along the path.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
label_colour	The colour of the edge label. If NA it will use the colour of the edge.
label_alpha	The opacity of the edge label. If NA it will use the opacity of the edge.
label_parse	If TRUE, the labels will be parsed into expressions and displayed as described in plotmath.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
angle_calc	Either 'none', 'along', or 'across'. If 'none' the label will use the angle aesthetic of the geom. If 'along' The label will be written along the edge direction. If 'across' the label will be written across the edge direction.
force_flip	Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.
label_dodge	A unit giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A unit giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Details

Many geom_edge_* layers comes in 3 flavors depending on the level of control needed over the drawing. The default (no numeric postfix) generate a number of points (n) along the edge and draws it as a path. Each point along the line has a numeric value associated with it giving the position along the path, and it is therefore possible to show the direction of the edge by mapping to this e.g. colour = ..index.. The version postfixed with a "2" uses the "long" edge format (see get_edges) and makes it possible to interpolate node parameter between the start and end node along the edge. It is considerable less performant so should only be used if this is needed. The version postfixed with a "0" draws the edge in the most performant way, often directly using an appropriate grob from the grid package, but does not allow for gradients along the edge.

Often it is beneficial to stop the drawing of the edge before it reaches the node, for instance in cases where an arrow should be drawn and the arrowhead shouldn't lay ontop or below the node point. geom_edge_* and geom_edge_*2 supports this through the start_cap and end_cap aesthetics that takes a geometry specification and dynamically caps the termini of the edges based on the given

specifications. This means that if `end_cap = circle(1, 'cm')` the edges will end at a distance of 1cm even during resizing of the plot window.

All `geom_edge_*` and `geom_edge_*2` have the ability to draw a label along the edge. The reason this is not a separate geom is that in order for the label to know the location of the edge it needs to know the edge type etc. Labels are drawn by providing a label aesthetic. The label_pos can be used to specify where along the edge it should be drawn by supplying a number between 0 and 1. The label_size aesthetic can be used to control the size of the label. Often it is needed to have the label written along the direction of the edge, but since the actual angle is dependent on the plot dimensions this cannot be calculated beforehand. Using the angle_calc argument allows you to specify whether to use the supplied angle aesthetic or whether to draw the label along or across the edge.

Aesthetics

geom_edge_loop and geom_edge_loop0 understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **from**
- **to**
- **span** *90*
- **direction** *45*
- **strength** *1*
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_loop furthermore takes the following aesthetics.

- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

index The position along the path (not computed for the *0 version)

Note

In order to avoid excessive typing edge aesthetic names are automatically expanded. Because of this it is not necessary to write `edge_colour` within the `aes()` call as `colour` will automatically be renamed appropriately.

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: `geom_edge_arc`, `geom_edge_density`, `geom_edge_diagonal`, `geom_edge_elbow`, `geom_edge_fan`, `geom_edge_hive`, `geom_edge_link`

Examples

```
require(igraph)
gr <- graph_from_data_frame(
  data.frame(from=c(1,1,2,2,3,3,3), to=c(1,2,2,3,3,1,2))
)

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_loop(aes(alpha = ..index..)) +
  geom_edge_fan(aes(alpha = ..index..))

ggraph(gr, 'igraph', algorithm = 'nicely') +
  geom_edge_loop0() +
  geom_edge_fan0()
```

---

geom_node_arc_bar *Show nodes as circles*

Description

This geom is equivalent in functionality to `geom_circle` and allows for plotting of nodes as circles with a radius scaled by the coordinate system. Beceause of the geoms reliance on the coordinate system it will only produce true circles when combined with `coord_fixed`

Usage

```
geom_node_arc_bar(mapping = NULL, data = NULL, position = "identity",
  show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x and y are mapped to x0 and y0 in the node data.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
...	other arguments passed on to layer. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like color = "red" or size = 3. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.

Aesthetics

geom_node_point understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x0**
- **y0**
- **r**
- alpha
- colour
- fill
- shape
- size
- stroke
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_circle, geom_node_point, geom_node_text, geom_node_tile

Examples

```
require(igraph)
gr <- graph_from_data_frame(flare$edges, vertices = flare$vertices)
ggraph(gr, 'circlepack', weight = 'size') + geom_node_circle() + coord_fixed()
```

`geom_node_circle` *Show nodes as circles*

Description

This geom is equivalent in functionality to `geom_circle` and allows for plotting of nodes as circles with a radius scaled by the coordinate system. Beceause of the geoms reliance on the coordinate system it will only produce true circles when combined with `coord_fixed`

Usage

```
geom_node_circle(mapping = NULL, data = NULL, position = "identity",
  show.legend = NA, ...)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes` or `aes_`. By default x and y are mapped to x0 and y0 in the node data.
`data`	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes.
`...`	other arguments passed on to `layer`. There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. • Other arguments to the layer, for example you override the default `stat` associated with the layer. • Other arguments passed on to the stat.

Aesthetics

geom_node_point understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x0**
- **y0**
- **r**
- alpha
- colour
- fill
- shape

- size
- stroke
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar, geom_node_point, geom_node_text, geom_node_tile

Examples

```
require(igraph)
gr <- graph_from_data_frame(flare$edges, vertices = flare$vertices)
ggraph(gr, 'circlepack', weight = 'size') + geom_node_circle() + coord_fixed()
```

---

geom_node_point *Show nodes as points*

Description

This geom is equivalent in functionality to geom_point and allows for simple plotting of nodes in different shapes, colours and sizes.

Usage

```
geom_node_point(mapping = NULL, data = NULL, position = "identity",
  show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x and y are mapped to x and y in the node data.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

... — Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

geom_node_point understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- alpha
- colour
- fill
- shape
- size
- stroke
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar, geom_node_circle, geom_node_text, geom_node_tile

Examples

```
require(igraph)
gr <- make_graph('bull')
V(gr)$class <- sample(letters[1:3], gorder(gr), replace = TRUE)

ggraph(gr, 'igraph', algorithm = 'nicely') + geom_node_point()
```

geom_node_text *Annotate nodes with text*

Description

These geoms are equivalent in functionality to geom_text and geom_label and allows for simple annotation of nodes.

Usage

```
geom_node_text(mapping = NULL, data = NULL, position = "identity",
  parse = FALSE, nudge_x = 0, nudge_y = 0, check_overlap = FALSE,
  show.legend = NA, repel = FALSE, ...)

geom_node_label(mapping = NULL, data = NULL, position = "identity",
  parse = FALSE, nudge_x = 0, nudge_y = 0, label.padding = unit(0.25,
  "lines"), label.r = unit(0.15, "lines"), label.size = 0.25,
  show.legend = NA, repel = FALSE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x and y are mapped to x and y in the node data.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath
nudge_x, nudge_y	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
repel	If TRUE, text labels will be repelled from each other to avoid overlapping, using the GeomTextRepel geom from the ggrepel package.

...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.
label.padding	Amount of padding around label. Defaults to 0.25 lines.
label.r	Radius of rounded corners. Defaults to 0.15 lines.
label.size	Size of label border, in mm.

Aesthetics

geom_node_point understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden. Italic aesthetics are required but not set by default

- **x**
- **y**
- *label*
- alpha
- angle
- colour
- family
- fontface
- hjust
- lineheight
- size
- vjust

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar, geom_node_circle, geom_node_point, geom_node_tile

Examples

```
require(igraph)
gr <- make_graph('bull')
V(gr)$class <- sample(letters[1:3], gorder(gr), replace = TRUE)

ggraph(gr, 'igraph', algorithm = 'nicely') + geom_node_point(aes(label = class))

ggraph(gr, 'igraph', algorithm = 'nicely') + geom_node_label(aes(label = class))
```

geom_node_tile *Draw the rectangles in a treemap*

Description

A treemap is a space filling layout that recursively divides a rectangle to the children of the node. Often only the leaf nodes are drawn as nodes higher up in the hierarchy would obscure what is below. geom_treemap is a shorthand for geom_node_treemap as node is implicit in the case of treemap drawing

Usage

```
geom_node_tile(mapping = NULL, data = NULL, position = "identity",
  show.legend = NA, ...)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. By default x, y, width and height are mapped to x, y, width and height in the node data.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

geom_treemap understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

- **x**
- **y**
- **width**
- **height**

- alpha
- colour
- fill
- size
- stroke
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar, geom_node_circle, geom_node_point, geom_node_text

Examples

```
require(igraph)
gr <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

ggraph(gr, 'treemap', weight = 'size') + geom_node_tile()

# We can color by modifying the graph
gr <- tree_apply(gr, function(node, parent, depth, tree) {
  if (depth == 1) {
    tree <- set_vertex_attr(tree, 'Class', node, V(tree)$shortName[node])
  } else if (depth > 1) {
    tree <- set_vertex_attr(tree, 'Class', node, V(tree)$Class[parent])
  }
  tree
})

ggraph(gr, 'treemap', weight = 'size') +
  geom_node_tile(aes(fill = Class, filter = leaf, alpha = depth), colour = NA) +
  geom_node_tile(aes(size = depth), colour = 'white') +
  scale_alpha(range = c(1, 0.5), guide = 'none') +
  scale_size(range = c(4, 0.2), guide = 'none')
```

---

get_con *Create a connection extractor function*

Description

Connections within the ggraph terminology are links between nodes that are not part of the network structure itself. In that sense connections does not affect the layout calculation in any way and will not be drawn by the standard `geom_edge_*` functions. A connection does not need to only be defined by a start and end node, but can include intermediary nodes. `get_con` helps in creating connection data by letting you specify start and end node and automatically find the shortest path within the graph structure that connects the given points. If this is not what is needed it is also possible to supply a list of vectors giving node indexes that defines a connection.

Usage

```
get_con(from = integer(), to = integer(), paths = NULL, ...)
```

Arguments

`from, to`	The index of the start and end nodes for the connections
`paths`	A list of integer vectors giving the index of nodes defining connections
`...`	Additional information to be added to the final data output

Value

A function that takes a layout_ggraph object and returns the given connections

See Also

Other extractors: `get_edges`, `get_nodes`

`get_edges` *Create edge extractor function*

Description

This function returns another function that can extract edges from a ggraph_layout object. The functionality of the returned function is decided by the arguments to `get_edges`. The need for `get_edges` is mainly to pass to the `data` argument of the different `geom_edge_*` functions in order to present them with the right kind of data. In general each `geom_edge_*` has the default set correctly so there is only need to modify the data argument if parallel edges should be collapsed.

Usage

```
get_edges(format = "short", collapse = "none", ...)
```

Arguments

format	Either 'short' (the default) or 'long'. See details for a descriptions of the differences
collapse	Either 'none' (the default), 'all' or 'direction'. Specifies whether parallel edges should be merged. See details for more information
...	Additional data that will be cbind'ed together with the returned edge data.

Details

There are two types of return formats possible for the result of the returned function:

short In this format each edge is described in one line in the format expected for geom_segment, that is, the start node position is encoded in the x and y column and the end node position is encoded in the xend and yend column. If node parameters are added to the edge the name of the parameters will be prefixed with node1. for the start node and node2. for the end node.

long In this format each edge consists of two rows with matching edge.id value. The start and end position are both encoded in the x and y column. The relative position of the rows determines which is the start and end node, the first occuring being the start node. If node parameters are added to the edge data the name of the parameters will be prefixed with node..

Node parameters are automatically added so it is possible to format edge aesthetics according to start or end node parameters, or interpolate edge aesthetics between start and end node parameters. Node parameters will be prefixed to avoid name clash with edge parameters. The prefix depends on the format (see above).

If the graph is not simple (it contains at most one edge between each node pair) it can be collapsed so either all edges between two nodes or all edges of the same direction between two nodes are merged. The edge parameters are taken from the first occuring edge, so if some more sophisticated summary is needed it is suggested that the graph be tidied up before plotting with ggraph.

Value

A data.frame with columns dependent on format as well as the graph type. In addition to the columns discussed in the details section, the data.frame will always contain the columns from, to and circular, the two former giving the indexes of the start and end node and the latter if the layout is circular (needed for correct formatting of some geom_edge_*). The graph dependent information is:

dendrogram A label column will hold the value of the edgetext attribute. In addition any value stored in the edgePar attribute will be added. Lastly a direction column will hold the relative position between the start and end nodes (needed for correct formatting of geom_edge_elbow).

igraph All edge attributes of the original graph object is added as columns to the data.frame

See Also

Other extractors: get_con, get_nodes

get_nodes *Create a node extractor function*

Description

This function returns another function that can extract nodes from a ggraph_layout object. As a ggraph_layout object is essentially a data.frame of nodes it might seem useless to provide this function, but since the node data is not necessarily available until after the `ggraph()` call it can be beneficial to be able to add information to the node data on a per-layer basis. Unlike `get_edges` the use of `get_nodes` is not mandatory and is only required if additional data should be added to selected node layers.

Usage

```
get_nodes(...)
```

Arguments

...	Additional data that should be cbind'ed togeether with the node data.

Value

A data.frame with the node data as well of any additional data supplied through ...

See Also

Other extractors: `get_con`, `get_edges`

ggraph *Create a ggraph plot*

Description

This function is the equivalent of `ggplot` in ggplot2. It takes care of setting up the plot object along with creating the layout for the plot based on the graph and the specification passed in. Alternatively a layout can be prepared in advance using `create_layout` and passed as the data argument. See *Details* for a desciption of all available layouts.

Usage

```
ggraph(graph, layout = "auto", ...)

create_layout(graph, layout, circular, ...)

## Default S3 method:
create_layout(graph, layout, ...)

## S3 method for class 'dendrogram'
create_layout(graph, layout, circular = FALSE, ...)

## S3 method for class 'igraph'
create_layout(graph, layout, circular = FALSE, ...)

## S3 method for class 'hclust'
create_layout(graph, layout, circular = FALSE, ...)

## S3 method for class 'network'
create_layout(graph, layout, circular = FALSE, ...)
```

Arguments

graph	The object containing the graph. See *Details* for a list of supported classes. Or a layout_ggraph object as returned from create_layout in which case all subsequent arguments is ignored.
layout	The type of layout to create.
...	Arguments passed on to the layout function.
circular	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE

Details

Following is a short description of the different layout types available in ggraph. Each layout is further described in its own help pages.

Dendrogram objects

The dendrogram class is used to store binary tree from e.g. hierarchical clustering. The layouts provided for this class is constrained to tree-like representations. hclust objects are supported through automatic conversion to dendrogram objects and thus supports the same layouts.

auto The default layout. Eqivalent to layout = 'dendrogram'

dendrogram Creates a tree using the heights already defined in the dendrogram object. See layout_dendrogram_dendrogra for further details

even Ignores the heights given by the dendrogram object and instead spreads the branch points out with an even distance. See layout_dendrogram_even for further details

Further, if the layouts provided for igraph objects are needed for dendrogram objects den_to_igraph is provided to convert dendrograms to igraph.

igraph objects

Any type of regular graph/network data can be represented as an igraph object. Because of this the different layouts that can be applied to igraph objects are quite diverse, but not all layouts makes sense to all types of graphs. It is up to the user to understand their data and choose an appropriate layout. For standard node-edge diagrams igraph itself defines a long range of different layout functions that are all available through the `igraph` layout where the specific layout is specified using the `algorithm` argument. In order to minimize typing all igraph algorithms can also be passed directly into the `layout` argument. `network` objects are supported by automatic conversion to igraph objects using `network_to_igraph` and thus supports the same layouts.

`auto` The default layout. Equivalent to `layout = 'igraph', algorithm = 'nicely'`

`igraph` Use one of the internal igraph layout algorithms. The algorithm is specified using the `algorithm` argument. All strings accepted by the `algorithm` argument can also be supplied directly into `layout`. See `layout_igraph_igraph` for further details

`dendrogram` Lays out the nodes in a tree-like graph as a dendrogram with leaves set at 0 and parents 1 unit above its tallest child. See `layout_igraph_dendrogram` for further details

`manual` Lets the user manually specify the location of each node by supplying a data.frame with an `x` and `y` column. See `layout_igraph_manual` for further details

`linear` Arranges the nodes linearly or circularly in order to make an arc diagram. See `layout_igraph_linear` for further details

`treemap` Creates a treemap from the graph, that is, a space-filing subdivision of rectangles showing a weighted hierarchy. See `layout_igraph_treemap` for further details

`circlepack` Creates a layout showing a hierarchy as circles within circles. Conceptually equal to treemaps. See `layout_igraph_circlepack` for further details

`partition` Create icicle or sunburst charts, where each layer subdivides the division given by the preceeding layer. See `layout_igraph_partition` for further details

`hive` Positions nodes on axes spreading out from the center based on node attributes. See `layout_igraph_hive` for further details

Value

For `ggraph()` an object of class gg onto which layers, scales, etc. can be added. For `create_layout()` an object inherting from `layout_ggraph`. `layout_ggraph` itself inherits from `data.frame` and can be considered as such. The data.frame contains the node positions in the `x` and `y` column along with additional columns generated by the specific layout, as well as node parameters inherited from the graph. Additional information is stored as attributes to the data.frame. The original graph object is stored in the `graph` attribute and the `circular` attribute contains a logical indicating whether the layout has been transformed to a circular representation.

See Also

`get_edges` for extracting edge information from the layout and `get_con` for extracting path information.

Examples

```
require(igraph)
gr <- make_graph('bull')
layout <- create_layout(gr, layout = 'igraph', algorithm = 'kk')
```

---

guide_edge_colourbar *Colourbar legend for edges*

Description

This function is equivalent to guide_colourbar but works for edge aesthetics.

Usage

```
guide_edge_colourbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
  barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
  draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
  default.unit = "line", reverse = FALSE, order = 0, ...)

guide_edge_colorbar(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  label = TRUE, label.position = NULL, label.theme = NULL,
  label.hjust = NULL, label.vjust = NULL, barwidth = NULL,
  barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE,
  draw.ulim = TRUE, draw.llim = TRUE, direction = NULL,
  default.unit = "line", reverse = FALSE, order = 0, ...)
```

Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
title.position	A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
title.theme	A theme object for rendering the title text. Usually the object of element_text() is expected. By default, the theme is specified by legend.title in theme() or theme.
title.hjust	A number specifying horizontal justification of the title text.
title.vjust	A number specifying vertical justification of the title text.
label	logical. If TRUE then the labels are drawn. If FALSE then the labels are invisible.

label.position	A character string indicating the position of a label. One of "top", "bottom" (default for horizontal guide), "left", or "right" (default for vertical guide).
label.theme	A theme object for rendering the label text. Usually the object of element_text() is expected. By default, the theme is specified by legend.text in theme() or theme.
label.hjust	A numeric specifying horizontal justification of the label text.
label.vjust	A numeric specifying vertical justification of the label text.
barwidth	A numeric or a grid::unit() object specifying the width of the colorbar. Default value is legend.key.width or legend.key.size in theme() or theme.
barheight	A numeric or a grid::unit() object specifying the height of the colorbar. Default value is legend.key.height or legend.key.size in theme() or theme.
nbin	A numeric specifying the number of bins for drawing colorbar. A smoother colorbar for a larger value.
raster	A logical. If TRUE then the colorbar is rendered as a raster object. If FALSE then the colorbar is rendered as a set of rectangles. Note that not all graphics devices are capable of rendering raster image.
ticks	A logical specifying if tick marks on colorbar should be visible.
draw.ulim	A logical specifying if the upper limit tick marks should be visible.
draw.llim	A logical specifying if the lower limit tick marks should be visible.
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
default.unit	A character string indicating grid::unit() for barwidth and barheight.
reverse	logical. If TRUE the colorbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom
order	positive integer less that 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
...	ignored.

Value

A guide object

---

guide_edge_direction *Edge direction guide*

Description

This guide is intended to show the direction of edges based on the aesthetics mapped to its progression, such as changing width, colour and opacity.

Usage

```
guide_edge_direction(title = waiver(), title.position = NULL,
  title.theme = NULL, title.hjust = NULL, title.vjust = NULL,
  arrow = TRUE, arrow.position = NULL, barwidth = NULL,
  barheight = NULL, nbin = 500, direction = NULL, default.unit = "line",
  reverse = FALSE, order = 0, ...)
```

Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default (waiver()), the name of the scale object or the name specified in labs() is used for the title.
title.position	A character string indicating the position of a title. One of "top" (default for a vertical guide), "bottom", "left" (default for a horizontal guide), or "right."
title.theme	A theme object for rendering the title text. Usually the object of element_text() is expected. By default, the theme is specified by legend.title in theme() or theme.
title.hjust	A number specifying horizontal justification of the title text.
title.vjust	A number specifying vertical justification of the title text.
arrow	Logical. Should an arrow be drawn to illustrate the direction. Defaults to TRUE
arrow.position	The position of the arrow relative to the example edge.
barwidth	A numeric or a grid::unit() object specifying the width of the colorbar. Default value is legend.key.width or legend.key.size in theme() or theme.
barheight	A numeric or a grid::unit() object specifying the height of the colorbar. Default value is legend.key.height or legend.key.size in theme() or theme.
nbin	A numeric specifying the number of bins for drawing colorbar. A smoother colorbar for a larger value.
direction	A character string indicating the direction of the guide. One of "horizontal" or "vertical."
default.unit	A character string indicating grid::unit() for barwidth and barheight.
reverse	logical. If TRUE the colorbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom
order	positive integer less that 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.
...	ignored.

Examples

```
gr <- igraph::graph_from_data_frame(highschool)
ggraph(gr, layout = 'kk') +
  geom_edge_fan(aes(alpha = ..index..)) +
  guides(edge_alpha = guide_edge_direction())
```

highschool *Friendship among high school boys*

Description

This dataset shows the friendship among high school boys as assessed by the question: "What fellows here in school do you go around with most often?". The question was posed twice, with one year in between (1957 and 1958) and shows the evolution in friendship between the two timepoints.

Usage

```
highschool
```

Format

The graph is stored as an unnamed edgelist with a year attribute.

from The boy answering the question

to The boy being the answer to the question

year The year the friendship was reported

Source

Coleman, J. S. *Introduction to Mathermatical Sociology*. New York: Free Press, pp.450-451.

layout_dendrogram_auto *Dendrogram layout for layout_dendrogram*

Description

This layout positions the branches and leafs according to the values given in the `height` attribute of the dendrogram object. If `repel = FALSE` the layout is equivalent to the one shown with the plot function on dendrogram objects.

Usage

```
layout_dendrogram_auto(graph, circular, ...)

layout_dendrogram_dendrogram(graph, circular = FALSE, offset = pi/2,
  repel = FALSE, ratio = 1)
```

Arguments

graph	A dendrogram object.
circular	Logical. Should the layout be transformed to a circular representation. Defaults to FALSE.
...	Parameters passed on to layout_dendrogram_dendrogram
offset	If circular = TRUE, where should it begin. Defaults to pi/2 which is equivalent to 12 o'clock.
repel	Logical. Should leafs be distanced from their neighbors as a function of their distance in the tree. Defaults to FALSE.
ratio	If repel = TRUE, the strength of the repelation. Defaults to 1.

Value

A data.frame with the columns x, y, circular, label, members, leaf as well as any value stored in the nodePar attribute of the dendrogram.

Note

This function is not intended to be used directly but by setting layout = 'dendrogram' in create_layout

---

layout_dendrogram_even

Even layout for layout_dendrogram

Description

This layout sets the heights of the branch points to be the maximum distance to a leaf. In this way the branch points are spread out evenly in the y-direction. After modifying the height it calls layout_dendrogram_dendrogram.

Usage

```
layout_dendrogram_even(graph, ...)
```

Arguments

graph	A dendrogram object
...	parameters passed on to layout_dendrogram_dendrogram

Value

A data.frame as layout_dendrogram_dendrogram

Note

This function is not intended to be used directly but by setting layout = 'even' in create_layout

`layout_igraph_auto` *Use igraph layout algorithms for layout_igraph*

Description

This layout function makes it easy to apply one of the layout algorithms supplied in igraph when plotting with ggraph. Layout names are auto completed so there is no need to write `layout_with_graphopt` or `layout_as_tree`, just `graphopt` and `tree` (though the former will also work if you want to be super explicit). Circular layout is only supported for tree-like layout (`tree` and `sugiyama`) and will throw an error when applied to other layouts.

Usage

```
layout_igraph_auto(graph, circular, ...)

layout_igraph_igraph(graph, algorithm, circular, offset = pi/2,
  use.dummy = FALSE, ...)
```

Arguments

`graph`	An igraph object.
`circular`	Logical. Should the layout be transformed to a circular representation. Defaults to `FALSE`. Only applicable to `algorithm = 'tree'` and `algorithm = 'sugiyama'`.
`...`	Arguments passed on to the respective layout functions
`algorithm`	The type of layout algorithm to apply. See `layout_` for links to the layouts supplied by igraph.
`offset`	If `circular = TRUE`, where should it begin. Defaults to `pi/2` which is equivalent to 12 o'clock.
`use.dummy`	Logical. In the case of `algorithm = 'sugiyama'` should the dummy-infused graph be used rather than the original. Defaults to `FALSE`.

Details

igraph provides a huge amount of possible layouts. They are all briefly described below:

Hierarchical layouts

`tree` Uses the *Reingold-Tilford* algorithm to place the nodes below their parent with the parent centered above its children. See `as_tree`

`sugiyama` Designed for directed acyclic graphs (that is, hierarchies where multiple parents are allowed) it minimizes the number of crossing edges. See `with_sugiyama`

Standard layouts

`bipartite` Minimize edge-crossings in a simple two-row (or column) layout for bipartite graphs. See `as_bipartite`

`star` Place one node in the center and the rest equidistantly around it. See `as_star`

`circle` Place nodes in a circle in the order of their index. Consider using `layout_igraph_linear` with `circular=TRUE` for more control. See `in_circle`

`nicely` Tries to pick an appropriate layout. See `nicely` for a description of the simpe decision tree it uses

`dh` Uses *Davidson and Harels* simulated annealing algorithm to place nodes. See `with_dh`

`gem` Place nodes on the plane using the GEM force-directed layout algorithm. See `with_gem`

`graphopt` Uses the Graphopt algorithm based on alternating attraction and repulsion to place nodes. See `with_graphopt`

`grid` Place nodes on a rectangular grid. See `on_grid`

`mds` Perform a multidimensional scaling of nodes using either the shortest path or a user supplied distance. See `with_mds`

`sphere` Place nodes uniformly on a sphere - less relevant for 2D visualizations of networks. See `on_sphere`

`randomly` Places nodes uniformly random. See `randomly`

`fr` Places nodes according to the force-directed algorithm of Fruchterman and Reingold. See `with_fr`

`kk` Uses the spring-based algorithm by Kamada and Kawai to place nodes. See `with_kk`

`drl` Uses the force directed algorithm from the DrL toolbox to place nodes. See `with_drl`

`lgl` Uses the algorithm from Large Graph Layout to place nodes. See `with_lgl`

Value

A data.frame with the columns `x`, `y`, `circular` as well as any information stored as vertex attributes on the igraph object.

Note

This function is not intended to be used directly but by setting `layout = 'igraph'` in `create_layout`

See Also

Other layout_igraph_*: `layout_igraph_circlepack`, `layout_igraph_dendrogram`, `layout_igraph_hive`, `layout_igraph_linear`, `layout_igraph_manual`, `layout_igraph_partition`, `layout_igraph_treemap`

`layout_igraph_circlepack` *Calculate nodes as circles packed within their parent circle*

Description

The circle packing algorithm is basically a treemap using circles instead of rectangles. Due to the nature of circles they cannot be packed as efficeintly leading to increased amount of "empty space" as compared to a treemap. This can be beneficial though, as the added empty space can aid in visually showing the hierarchy.

Usage

```
layout_igraph_circlepack(graph, weight = NULL, circular = FALSE,
  sort.by = NULL, direction = "out")
```

Arguments

`graph`	An igraph object
`weight`	An optional vertex attribute to use as weight. Will only affect the weight of leaf nodes as the weight of non-leaf nodes are derived from their children.
`circular`	Logical. Should the layout be transformed to a circular representation. Ignored.
`sort.by`	The name of a vertex attribute to sort the nodes by.
`direction`	The direction of the tree in the graph. `'out'` (default) means that parents point towards their children, while `'in'` means that children point towards their parent.

Details

The circle packing is based on the algorithm developed by Weixin Wang and collaborators which tries to find the most dense packing of circles as they are added, one by one. This makes the algorithm very dependent on the order in which circles are added and it is possible that layouts could sometimes be optimized by choosing a different ordering. The algorithm for finding the enclosing circle is that randomized incremental algorithm proposed by Emo Welzl. Both of the above algorithms are the same as used in the D3.js implementation of circle packing and their C++ implementation in ggraph is inspired by Mike Bostocks JavaScript implementation.

Value

A data.frame with the columns `x`, `y`, `r`, `leaf`, `depth`, `circular` as well as any information stored as vertex attributes on the igraph object.

Note

Circle packing is a layout intended for trees, that is, graphs where nodes only have one parent and zero or more children. If the provided graph does not fit this format an attempt to convert it to such a format will be made.

References

Wang, W., Wang, H. H., Dai, G., & Wang, H. (2006). *Visualization of large hierarchical data by circle packing*. Chi, 517-520.

Welzl, E. (1991). *Smallest enclosing disks (balls and ellipsoids)*. New Results and New Trends in Computer Science, 359-370.

See Also

Other layout_igraph_*: `layout_igraph_auto`, `layout_igraph_dendrogram`, `layout_igraph_hive`, `layout_igraph_linear`, `layout_igraph_manual`, `layout_igraph_partition`, `layout_igraph_treemap`

`layout_igraph_dendrogram`

Apply a dendrogram layout to layout_igraph

Description

This layout mimicks the `layout_as_tree` algorithm supplied by igraph, but puts all leaves at 0 and builds it up from there, instead of starting from the root and building it from there. The height of branch points are related to the maximum distance to an edge from the branch node.

Usage

```
layout_igraph_dendrogram(graph, circular = FALSE, offset = pi/2,
  direction = "out")
```

Arguments

`graph`	An igraph object
`circular`	Logical. Should the layout be transformed to a circular representation. Defaults to `FALSE`.
`offset`	If `circular = TRUE`, where should it begin. Defaults to `pi/2` which is equivalent to 12 o'clock.
`direction`	The direction to the leaves. Defaults to 'out'

Value

A data.frame with the columns `x`, `y`, `circular` and `leaf` as well as any information stored as vertex attributes on the igraph object.

Note

This function is not intended to be used directly but by setting `layout = 'dendrogram'` in `create_layout`

See Also

Other layout_igraph_*: layout_igraph_auto, layout_igraph_circlepack, layout_igraph_hive, layout_igraph_linear, layout_igraph_manual, layout_igraph_partition, layout_igraph_treemap

---

`layout_igraph_hive` *Place nodes in a Hive Plot layout*

Description

Hive plots were invented by Martin Krzywinski as a perceptually uniform and scalable alternative to standard node-edge layouts. In hive plots nodes are positioned on axes radiating out from a center based on their own information e.g. membership of a class, size of neighborhood, etc. Edges are then drawn between nodes as bezier curves. As the placement of nodes is not governed by convoluted algorithms but directly reflects the qualities of the nodes itself the resulting plot can be easier to interpret as well as compare to other graphs.

Usage

```
layout_igraph_hive(graph, axis, axis.pos = NULL, sort.by = NULL,
  divide.by = NULL, divide.order = NULL, normalize = TRUE,
  center.size = 0.1, divide.size = 0.05, use.numeric = FALSE,
  offset = pi/2, split.axes = "none", split.angle = pi/6,
  circular = FALSE)
```

Arguments

`graph`	An igraph object
`axis`	The node attribute to use for assigning nodes to axes
`axis.pos`	The relative distance to the prior axis. Default (`NULL`) places axes equidistant.
`sort.by`	The node attribute to use for placing nodes along their axis. Defaults (`NULL`) places nodes sequentially.
`divide.by`	An optional node attribute to subdivide each axis by.
`divide.order`	The order the axis subdivisions should appear in
`normalize`	Logical. Should axis lengths be equal or reflect the number of nodes in each axis. Defaults to `TRUE`.
`center.size`	The size of the blank center, that is, the start position of the axes.
`divide.size`	The distance between subdivided axis segments.
`use.numeric`	Logical, If the `sort.by` attribute is numeric, should these values be used directly in positioning the nodes along the axes. Defaults to `FALSE` which sorts the numeric values and positions them equidistant from each other.
`offset`	Change the overall rotation of the hive plot by changing the offset of the first axis.
`split.axes`	Should axes be split to show edges between nodes on the same axis? One of:

	`'none'` Do not split axes and show in-between edges `'loops'` Only split axes that contain in-between edges `'all'` Split all axes
`split.angle`	The angular distance between the two axes resulting from a split.
`circular`	Ignored.

Details

In order to be able to draw all edges without edges crossing axes you should not assign nodes to axes based on a variable with more than three levels.

Value

A data.frame with the columns `x`, `y`, `r`, `centerSize`, `split`, `axis`, `section`, `angle`, `circular` as well as any information stored as vertex attributes on the igraph object.

References

Krzywinski, M., Birol, I., Jones, SJM., and Marra, MA. (2012). *Hive plots-rational approach to visualizing networks*. Brief Bioinform 13 (5): 627-644. http://doi.org/10.1093/bib/bbr069

http://www.hiveplot.net

See Also

Other layout_igraph_*: `layout_igraph_auto`, `layout_igraph_circlepack`, `layout_igraph_dendrogram`, `layout_igraph_linear`, `layout_igraph_manual`, `layout_igraph_partition`, `layout_igraph_treemap`

---

layout_igraph_linear *Place nodes on a line or circle*

Description

This layout puts all nodes on a line, possibly sorted by a node attribute. If `circular = TRUE` the nodes will be laid out on the unit circle instead. In the case where the `sort.by` attribute is numeric, the numeric values will be used as the x-position and it is thus possible to have uneven spacing between the nodes.

Usage

```
layout_igraph_linear(graph, circular, sort.by = NULL, use.numeric = FALSE,
  offset = pi/2)
```

Arguments

graph	An igraph object
circular	Logical. Should the layout be transformed to a circular representation. Defaults to FALSE.
sort.by	The name of a vertex attribute to sort the nodes by.
use.numeric	Logical. Should a numeric sort.by attribute be used as the actual x-coordinates in the layout. May lead to overlapping nodes. Defaults to FALSE
offset	If circular = TRUE, where should it begin. Defaults to pi/2 which is equivalent to 12 o'clock.

Value

A data.frame with the columns x, y, circular as well as any information stored as vertex attributes on the igraph object.

See Also

Other layout_igraph_*: layout_igraph_auto, layout_igraph_circlepack, layout_igraph_dendrogram, layout_igraph_hive, layout_igraph_manual, layout_igraph_partition, layout_igraph_treemap

---

layout_igraph_manual *Manually specify a layout for layout_igraph*

Description

This layout function lets you pass the node positions in manually. Each row in the supplied data frame will correspond to a vertex in the igraph object matched by index.

Usage

```
layout_igraph_manual(graph, node.positions, circular)
```

Arguments

graph	An igraph object
node.positions	A data.frame with the columns x and y (additional columns are ignored).
circular	Ignored

Value

A data.frame with the columns x, y, circular as well as any information stored as vertex attributes on the igraph object.

See Also

Other layout_igraph_*: layout_igraph_auto, layout_igraph_circlepack, layout_igraph_dendrogram, layout_igraph_hive, layout_igraph_linear, layout_igraph_partition, layout_igraph_treemap

layout_igraph_partition *Calculate nodes as areas dividing their parent*

Description

The partition layout is a way to show hierarchical data in the same way as layout_igraph_treemap. Instead of subdividing the parent area the partition layout shows the division of a nodes children next to the area of the node itself. As such the node positions will be very reminicent of a reingold-tilford tree layout but by plotting nodes as areas it better communicate the total weight of a node by summing up all its children. Often partition layouts are called icicle plots or sunburst diagrams (in case a radial transform is applied).

Usage

```
layout_igraph_partition(graph, weight = NULL, circular = FALSE,
  height = NULL, sort.by = NULL, direction = "out", offset = pi/2,
  const.area = TRUE)
```

Arguments

graph	An igraph object
weight	An optional vertex attribute to use as weight. Will only affect the weight of leaf nodes as the weight of non-leaf nodes are derived from their children.
circular	Logical. Should the layout be transformed to a circular representation. If TRUE the resulting layout will be a sunburst diagram.
height	An optional vertex attribute to use as height. If NULL all nodes will be given a height of 1.
sort.by	The name of a vertex attribute to sort the nodes by.
direction	The direction of the tree in the graph. 'out' (default) means that parents point towards their children, while 'in' means that children point towards their parent.
offset	If circular = TRUE, where should it begin. Defaults to pi/2 which is equivalent to 12 o'clock.
const.area	Logical. Should 'height' be scaled for area proportionality when using circular = TRUE. Defaults to TRUE.

Value

If circular = FALSE A data.frame with the columns x, y, width, height, leaf, depth, circular as well as any information stored as vertex attributes on the igraph object. If circular = TRUE A data.frame with the columns x, y, r0, r, start, end, leaf, depth, circular as well as any information stored as vertex attributes on the igraph object.

Note

partition is a layout intended for trees, that is, graphs where nodes only have one parent and zero or more children. If the provided graph does not fit this format an attempt to convert it to such a format will be made.

References

Kruskal, J. B., Landwehr, J. M. (1983). *Icicle Plots: Better Displays for Hierarchical Clustering*. American Statistician Vol 37(2), 162-168. http://doi.org/10.2307/2685881

See Also

Other layout_igraph_*: layout_igraph_auto, layout_igraph_circlepack, layout_igraph_dendrogram, layout_igraph_hive, layout_igraph_linear, layout_igraph_manual, layout_igraph_treemap

---

layout_igraph_treemap *Calculate nodes as rectangels subdividing that of their parent*

Description

A treemap is a space filling hierarchical layout that maps nodes to rectangles. The rectangles of the children of a node is packed into the rectangle of the node so that the size of a rectangle is a function of the size of the children. The size of the leaf nodes can be mapped arbitrarily (defaults to 1). Many different algorithms exists for dividing a rectangle into smaller bits, some optimizing the aspect ratio and some focusing on the ordering of the rectangles. See details for more discussions on this. The treemap layout was first developed by Ben Shneiderman for visualizing disk usage in the early '90 and has seen many improvements since.

Usage

```
layout_igraph_treemap(graph, algorithm = "split", weight = NULL,
  circular = FALSE, sort.by = NULL, direction = "out", height = 1,
  width = 1)
```

Arguments

graph	An igraph object
algorithm	The name of the tiling algorithm to use. Defaults to 'split'
weight	An optional vertex attribute to use as weight. Will only affect the weight of leaf nodes as the weight of non-leaf nodes are derived from their children.
circular	Logical. Should the layout be transformed to a circular representation. Ignored.
sort.by	The name of a vertex attribute to sort the nodes by.
direction	The direction of the tree in the graph. 'out' (default) means that parents point towards their children, while 'in' means that children point towards their parent.
height	The height of the bounding rectangle
width	The width of the bounding rectangle

Details

Different approaches to dividing the rectangles in a treemap exists; all with their strengths and weaknesses. Currently only the split algorithm is implemented which strikes a good balance between aspect ratio and order preservation, but other, more well-known, algorithms such as squarify and slice-and-dice will eventually be implemented.

Algorithms

Split (default)

The Split algorithm was developed by Bjorn Engdahl in order to address the downsides of both the original slice-and-dice algorithm (poor aspect ratio) and the popular squarify algorithm (no ordering of nodes). It works by finding the best cut in the ordered list of children in terms of making sure that the two rectangles associated with the split will have optimal aspect ratio.

Value

A data.frame with the columns `x`, `y`, `width`, `height`, `leaf`, `depth`, `circular` as well as any information stored as vertex attributes on the igraph object.

Note

Treemap is a layout intended for trees, that is, graphs where nodes only have one parent and zero or more children. If the provided graph does not fit this format an attempt to convert it to such a format will be made.

References

Engdahl, B. (2005). *Ordered and unordered treemap algorithms and their applications on handheld devices*. Master's Degree Project.

Johnson, B., & Ben Shneiderman. (1991). *Tree maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures*. IEEE Visualization, 284-291. http://doi.org/10.1109/VISUAL.1991.175815

See Also

Other layout_igraph_*: `layout_igraph_auto`, `layout_igraph_circlepack`, `layout_igraph_dendrogram`, `layout_igraph_hive`, `layout_igraph_linear`, `layout_igraph_manual`, `layout_igraph_partition`

`network_to_igraph` *Coerce network to igraph*

Description

This utility function performs a conversion of network objects from the network package into igraph object compatible with ggraph. Edge and node attributes are preserved which, for the context of ggraph, is the most important.

Usage

```
network_to_igraph(graph)
```

Arguments

graph	A graph as a network object

Value

A representation of the same graph as given in the function call but as an igraph object.

---

node_angle *Get the angle of nodes and edges*

Description

These helper functions makes it easy to calculate the angle associated with nodes and edges. For nodes the angle is defined as the angle of the vector pointing towards the node position, and is thus mainly suited for circular layouts where it can be used to calculate the angle of labels. For edges it is simply the angle of the vector describing the edge.

Usage

```
node_angle(x, y, degrees = TRUE)

edge_angle(x, y, xend, yend, degrees = TRUE)
```

Arguments

x, y	A vector of positions
degrees	Logical. Should the angle be returned in degree (TRUE) or radians (FALSE). Defaults to TRUE.
xend, yend	The end position of the edge

Value

A vector with the angle of each node/edge

Examples

```
require(igraph)
flareGraph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

ggraph(flareGraph, 'dendrogram', circular = TRUE) +
  geom_edge_diagonal0() +
  geom_node_text(aes(filter = leaf, angle = node_angle(x, y), label = shortName),
                 hjust = 'outward', size = 2) +
  expand_limits(x = c(-1.3, 1.3), y = c(-1.3, 1.3))
```

pack_circles *Pack circles together*

Description

This function is a direct interface to the circle packing algorithm used by layout_igraph_circlepack. It takes a vector of sizes and returns the x and y position of each circle as a two-column matrix.

Usage

```
pack_circles(areas)
```

Arguments

areas	A vector of circle areas

Value

A matrix with two columns and the same number of rows as the length of the "areas" vector. The matrix has the following attributes added: "enclosing_radius" giving the radius of the smallest enclosing circle, and "front_chain" giving the terminating members of the front chain (see Wang *et al*. 2006).

References

Wang, W., Wang, H. H., Dai, G., & Wang, H. (2006). *Visualization of large hierarchical data by circle packing*. Chi, 517-520.

Examples

```
library(ggforce)
sizes <- sample(10, 100, TRUE)

position <- pack_circles(sizes)
data <- data.frame(x = position[,1], y = position[,2], r = sqrt(sizes/pi))

ggplot() +
  geom_circle(aes(x0 = x, y0 = y, r = r), data = data, fill = 'steelblue') +
  geom_circle(aes(x0 = 0, y0 = 0, r = attr(position, 'enclosing_radius'))) +
  geom_polygon(aes(x = x, y = y),
               data = data[attr(position, 'front_chain'), ],
               fill = NA,
               colour = 'black')
```

scale_edge_alpha *Edge alpha scales*

Description

This set of scales defines new alpha scales for edge geoms equivalent to the ones already defined by ggplot2. See scale_alpha for more information. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write edge_alpha in the call to the geom - just use alpha.

Usage

```
scale_edge_alpha(..., range = c(0.1, 1))

scale_edge_alpha_continuous(..., range = c(0.1, 1))

scale_edge_alpha_discrete(..., range = c(0.1, 1))

scale_edge_alpha_manual(..., values)

scale_edge_alpha_identity(..., guide = "none")
```

Arguments

...	Other arguments passed on to continuous_scale() or discrete_scale() as appropriate, to control name, limits, breaks, labels and so forth.
range	Output range of alpha values. Must lie between 0 and 1.
values	a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given na.value.
guide	Guide to use for this scale. Defaults to "none".

Value

A ggproto object inheriting from Scale

See Also

Other scale_edge_*: scale_edge_colour, scale_edge_fill, scale_edge_linetype, scale_edge_shape, scale_edge_size, scale_edge_width, scale_label_size

scale_edge_colour *Edge colour scales*

Description

This set of scales defines new colour scales for edge geoms equivalent to the ones already defined by ggplot2. The parameters are equivalent to the ones from ggplot2 so there is nothing new under the sun. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write edge_colour in the call to the geom - just use colour.

Usage

```
scale_edge_colour_hue(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_edge_colour_brewer(..., type = "seq", palette = 1, direction = 1)

scale_edge_colour_distiller(..., type = "seq", palette = 1,
  direction = -1, values = NULL, space = "Lab", na.value = "grey50",
  guide = "edge_colourbar")

scale_edge_colour_gradient(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_colour_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar")

scale_edge_colour_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar", colors)

scale_edge_colour_grey(..., start = 0.2, end = 0.8, na.value = "red")

scale_edge_colour_identity(..., guide = "none")

scale_edge_colour_manual(..., values)

scale_edge_colour_viridis(..., alpha = 1, begin = 0, end = 1,
  discrete = FALSE, option = "D", direction = 1)

scale_edge_colour_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_colour_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_edge_color_hue(..., h = c(0, 360) + 15, c = 100, l = 65,
```

```
  h.start = 0, direction = 1, na.value = "grey50")

scale_edge_color_brewer(..., type = "seq", palette = 1, direction = 1)

scale_edge_color_distiller(..., type = "seq", palette = 1, direction = -1,
  values = NULL, space = "Lab", na.value = "grey50",
  guide = "edge_colourbar")

scale_edge_color_gradient(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_color_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar")

scale_edge_color_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar", colors)

scale_edge_color_grey(..., start = 0.2, end = 0.8, na.value = "red")

scale_edge_color_identity(..., guide = "none")

scale_edge_color_manual(..., values)

scale_edge_color_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_color_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_edge_color_viridis(..., alpha = 1, begin = 0, end = 1,
  discrete = FALSE, option = "D", direction = 1)
```

Arguments

`...` Arguments passed on to `discrete_scale`

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

na.value If `na.translate = TRUE`, what value aesthetic value should missing be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

guide A function used to create a guide or its name. See `guides()` for more info.

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

super The super class to use for the constructed scale

`h`	range of hues to use, in [0, 360]
`c`	chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
`l`	luminance (lightness), in [0, 100]
`h.start`	hue to start at
`direction`	direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise
`na.value`	Colour to use for missing values
`type`	One of seq (sequential), div (diverging) or qual (qualitative)
`palette`	If a string, will use that named palette. If a number, will index into the list of palettes of appropriate `type`
`values`	if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale()` for a convience function to map an arbitrary range to between 0 and 1.
`space`	colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.
`guide`	Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend.

low, high	Colours for low and high ends of the gradient.
mid	colour for mid point
midpoint	The midpoint (in data value) of the diverging scale. Defaults to 0.
colours, colors	Vector of colours to use for n-colour gradient.
start	gray value at low end of palette
end	gray value at high end of palette
alpha	pass through parameter to viridis
begin	The (corrected) hue in [0,1] at which the viridis colormap begins.
discrete	generate a discrete palette? (default: FALSE - generate continuous palette)
option	A character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E").

Value

A ggproto object inheriting from Scale

See Also

Other scale_edge_*: scale_edge_alpha, scale_edge_fill, scale_edge_linetype, scale_edge_shape, scale_edge_size, scale_edge_width, scale_label_size

---

scale_edge_fill *Edge fill scales*

Description

This set of scales defines new fill scales for edge geoms equivalent to the ones already defined by ggplot2. The parameters are equivalent to the ones from ggplot2 so there is nothing new under the sun. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write edge_fill in the call to the geom - just use fill.

Usage

```
scale_edge_fill_hue(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")

scale_edge_fill_brewer(..., type = "seq", palette = 1, direction = 1)

scale_edge_fill_distiller(..., type = "seq", palette = 1, direction = -1,
  values = NULL, space = "Lab", na.value = "grey50",
  guide = "edge_colourbar")

scale_edge_fill_gradient(..., low = "#132B43", high = "#56B1F7",
```

```
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_fill_gradient2(..., low = muted("red"), mid = "white",
  high = muted("blue"), midpoint = 0, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar")

scale_edge_fill_gradientn(..., colours, values = NULL, space = "Lab",
  na.value = "grey50", guide = "edge_colourbar", colors)

scale_edge_fill_grey(..., start = 0.2, end = 0.8, na.value = "red")

scale_edge_fill_identity(..., guide = "none")

scale_edge_fill_manual(..., values)

scale_edge_fill_viridis(..., alpha = 1, begin = 0, end = 1,
  discrete = FALSE, option = "D", direction = 1)

scale_edge_fill_continuous(..., low = "#132B43", high = "#56B1F7",
  space = "Lab", na.value = "grey50", guide = "edge_colourbar")

scale_edge_fill_discrete(..., h = c(0, 360) + 15, c = 100, l = 65,
  h.start = 0, direction = 1, na.value = "grey50")
```

Arguments

`...` Arguments passed on to `discrete_scale`

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

na.value If `na.translate = TRUE`, what value aesthetic value should missing be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

guide A function used to create a guide or its name. See `guides()` for more info.

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

super The super class to use for the constructed scale

`h`	range of hues to use, in [0, 360]
`c`	chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
`l`	luminance (lightness), in [0, 100]
`h.start`	hue to start at
`direction`	direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise
`na.value`	Colour to use for missing values
`type`	One of seq (sequential), div (diverging) or qual (qualitative)
`palette`	If a string, will use that named palette. If a number, will index into the list of palettes of appropriate `type`
`values`	if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale()` for a convience function to map an arbitrary range to between 0 and 1.
`space`	colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.
`guide`	Type of legend. Use `"colourbar"` for continuous colour bar, or `"legend"` for discrete colour legend.
`low, high`	Colours for low and high ends of the gradient.
`mid`	colour for mid point
`midpoint`	The midpoint (in data value) of the diverging scale. Defaults to 0.
`colours, colors`	Vector of colours to use for n-colour gradient.
`start`	gray value at low end of palette
`end`	gray value at high end of palette
`alpha`	pass through parameter to `viridis`

begin	The (corrected) hue in [0,1] at which the viridis colormap begins.
discrete	generate a discrete palette? (default: FALSE - generate continuous palette)
option	A character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E").

Value

A ggproto object inheriting from Scale

See Also

Other scale_edge_*: scale_edge_alpha, scale_edge_colour, scale_edge_linetype, scale_edge_shape, scale_edge_size, scale_edge_width, scale_label_size

---

scale_edge_linetype *Edge linetype scales*

Description

This set of scales defines new linetype scales for edge geoms equivalent to the ones already defined by ggplot2. See scale_linetype for more information. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write edge_linetype in the call to the geom - just use linetype.

Usage

```
scale_edge_linetype(..., na.value = "blank")

scale_edge_linetype_continuous(...)

scale_edge_linetype_discrete(..., na.value = "blank")

scale_edge_linetype_manual(..., values)

scale_edge_linetype_identity(..., guide = "none")
```

Arguments

...	Arguments passed on to discrete_scale

breaks One of:

- NULL for no breaks
- waiver() for the default breaks computed by the transformation object
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

aesthetics The names of the aesthetics that this scale works with

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

guide A function used to create a guide or its name. See `guides()` for more info.

super The super class to use for the constructed scale

`na.value` The linetype to use for `NA` values.

`values` a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given `na.value`.

`guide` Guide to use for this scale. Defaults to `"none"`.

Value

A ggproto object inheriting from `Scale`

See Also

Other scale_edge_*: `scale_edge_alpha`, `scale_edge_colour`, `scale_edge_fill`, `scale_edge_shape`, `scale_edge_size`, `scale_edge_width`, `scale_label_size`

`scale_edge_shape` *Edge shape scales*

Description

This set of scales defines new shape scales for edge geoms equivalent to the ones already defined by ggplot2. See scale_shape for more information. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write edge_shape in the call to the geom - just use shape.

Usage

```
scale_edge_shape(..., solid = TRUE)

scale_edge_shape_discrete(..., solid = TRUE)

scale_edge_shape_continuous(...)

scale_edge_shape_manual(..., values)

scale_edge_shape_identity(..., guide = "none")
```

Arguments

`...` Arguments passed on to `discrete_scale`

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

limits A character vector that defines possible values of the scale and their order.

drop Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` uses all the levels in the factor.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

na.value If `na.translate = TRUE`, what value aesthetic value should missing be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

aesthetics The names of the aesthetics that this scale works with

scale_name The name of the scale

palette A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object

- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

guide A function used to create a guide or its name. See `guides()` for more info.

super The super class to use for the constructed scale

`solid`	Should the shapes be solid, `TRUE`, or hollow, `FALSE`?
`values`	a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given `na.value`.
`guide`	Guide to use for this scale.

Value

A ggproto object inheriting from `Scale`

See Also

Other scale_edge_*: `scale_edge_alpha`, `scale_edge_colour`, `scale_edge_fill`, `scale_edge_linetype`, `scale_edge_size`, `scale_edge_width`, `scale_label_size`

`scale_edge_size` *Edge size scales*

Description

This set of scales defines new size scales for edge geoms equivalent to the ones already defined by ggplot2. See `scale_size` for more information. The different geoms will know whether to use edge scales or the standard scales so it is not necessary to write `edge_size` in the call to the geom - just use `size`.

Usage

```
scale_edge_size_continuous(..., range = c(1, 6))

scale_edge_radius(..., range = c(1, 6))

scale_edge_size(..., range = c(1, 6))

scale_edge_size_discrete(..., range = c(2, 6))

scale_edge_size_area(..., max_size = 6)

scale_edge_size_manual(..., values)

scale_edge_size_identity(..., guide = "none")
```

Arguments

...	Arguments passed on to `continuous_scale`

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

minor_breaks One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

limits A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum.

oob Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.

na.value Missing values will be replaced with this value.

trans Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".

A transformation object bundles together a transform, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `name_trans`, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

guide A function used to create a guide or its name. See `guides()` for more info.

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

super The super class to use for the constructed scale

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

range	a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.
max_size	Size of largest points.
values	a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given na.value.
guide	A function used to create a guide or its name. See guides() for more info.

Value

A ggproto object inheriting from Scale

Note

In ggplot2 size conflates both line width and point size into one scale. In ggraph there is also a width scale (scale_edge_width) that is used for linewidth. As edges are often represented by lines the width scale is the most common.

See Also

Other scale_edge_*: scale_edge_alpha, scale_edge_colour, scale_edge_fill, scale_edge_linetype, scale_edge_shape, scale_edge_width, scale_label_size

---

scale_edge_width *Edge width scales*

Description

This set of scales defines width scales for edge geoms. Of all the new edge scales defined in ggraph, this is the only one not having an equivalent in ggplot2. In essence it mimicks the use of size in geom_line and related. As almost all edge representations are lines of some sort, edge_width will be used much more often than edge_size. It is not necessary to spell out that it is an edge scale as the geom knows if it is drawing an edge. Just write width and not edge_width in the call to geoms.

Usage

```
scale_edge_width_continuous(..., range = c(1, 6))

scale_edge_width(..., range = c(1, 6))

scale_edge_width_discrete(..., range = c(2, 6))

scale_edge_width_manual(..., values)

scale_edge_width_identity(..., guide = "none")
```

Arguments

...	Arguments passed on to `continuous_scale`

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

minor_breaks One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

limits A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum.

oob Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.

na.value Missing values will be replaced with this value.

trans Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".

A transformation object bundles together a transform, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `name_trans`, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

guide A function used to create a guide or its name. See `guides()` for more info.

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

super The super class to use for the constructed scale

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

range	a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.
values	a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given `na.value`.
guide	A function used to create a guide or its name. See `guides()` for more info.

Value

A ggproto object inheriting from `Scale`

See Also

Other scale_edge_*: `scale_edge_alpha`, `scale_edge_colour`, `scale_edge_fill`, `scale_edge_linetype`, `scale_edge_shape`, `scale_edge_size`, `scale_label_size`

---

scale_label_size *Edge label size scales*

Description

This set of scales defines new size scales for edge labels in order to allow for separate sizing of edges and their labels.

Usage

```
scale_label_size_continuous(..., range = c(1, 6))

scale_label_size(..., range = c(1, 6))

scale_label_size_discrete(..., range = c(2, 6))

scale_label_size_manual(..., values)

scale_label_size_identity(..., guide = "none")
```

Arguments

...	Arguments passed on to `continuous_scale`

name The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object

- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

minor_breaks One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks.

labels One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- A function that takes the breaks as input and returns labels as output

limits A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum.

oob Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.

na.value Missing values will be replaced with this value.

trans Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".
A transformation object bundles together a transform, it's inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `name_trans`, e.g. `scales::boxcox_trans()`. You can create your own transformation with `scales::trans_new()`.

guide A function used to create a guide or its name. See `guides()` for more info.

position The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales

super The super class to use for the constructed scale

expand Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`range` a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.

`values` a set of aesthetic values to map data values to. If this is a named vector, then the values will be matched based on the names. If unnamed, values will be matched in order (usually alphabetical) with the limits of the scale. Any data values that don't match will be given `na.value`.

`guide` A function used to create a guide or its name. See `guides()` for more info.

Value

A ggproto object inheriting from Scale

See Also

Other scale_edge_*: scale_edge_alpha, scale_edge_colour, scale_edge_fill, scale_edge_linetype, scale_edge_shape, scale_edge_size, scale_edge_width

---

theme_graph *A theme tuned for graph visualizations*

Description

When plotting graphs, networks, and trees the coordinate values are often of no importance and axes are thus a distraction. ggraph comes with a build-in theme that removes redundant elements in order to put focus on the data. Furthermore the default behaviour is to use a narrow font so text takes up less space. Theme colour is defined by a background and foreground colour where the background defines the colour of the whole graphics area and the foreground defines the colour of the strip and border. By default strip and border is turned off as it is an unnecessary element unless facetting is used. To add a foreground colour to a plot that is already using theme_graph the th_foreground helper is provided. In order to use this appearance as default use the set_graph_style function. An added benefit of this is that it also changes the default text-related values in the different geoms for a completely coherent look. unset_graph_style can be used to revert the defaults back to their default settings (that is, they are not necessarily reverted back to what they were prior to calling set_graph_style).

Usage

```
theme_graph(base_family = "Arial Narrow", base_size = 11,
  background = "white", foreground = NULL, border = TRUE,
  text_colour = "black", bg_text_colour = text_colour,
  fg_text_colour = text_colour, title_family = base_family,
  title_size = 18, title_face = "bold", title_margin = 10,
  title_colour = bg_text_colour, subtitle_family = base_family,
  subtitle_size = 12, subtitle_face = "plain", subtitle_margin = 15,
  subtitle_colour = bg_text_colour, strip_text_family = base_family,
  strip_text_size = 10, strip_text_face = "bold",
  strip_text_colour = fg_text_colour, caption_family = base_family,
  caption_size = 9, caption_face = "italic", caption_margin = 10,
  caption_colour = bg_text_colour, plot_margin = margin(30, 30, 30, 30))

th_foreground(foreground = "grey80", fg_text_colour = NULL,
  border = FALSE)

set_graph_style(family = "Arial Narrow", face = "plain", size = 11,
  text_size = 11, text_colour = "black", ...)

unset_graph_style()
```

Arguments

`base_size, size, text_size, title_size, subtitle_size, strip_text_size, caption_size`
: The size to use for the various text elements. `text_size` will be used as geom defaults

`background`
: The colour to use for the background. This theme sets all background elements except for plot.background to `element_blank` so this controls the background for all elements of the plot. Set to `NA` to remove the background (thus making the plot transparent)

`foreground`
: The colour of foreground elements, specifically strip and border. Set to `NA` to remove.

`border`
: Logical. Should border be drawn if a foreground colour is provided?

`text_colour, bg_text_colour, fg_text_colour, title_colour, subtitle_colour, strip_text_colour, capt`
: The colour of the text in the various text elements

`title_margin, subtitle_margin, caption_margin`
: The margin to use between the text elements and the plot area

`plot_margin`
: The plot margin

`family, base_family, title_family, subtitle_family, strip_text_family, caption_family`
: The font to use for the different elements

`face, title_face, subtitle_face, strip_text_face, caption_face`
: The fontface to use for the various text elements

`...`
: Parameters passed on the `theme_graph`

Examples

```
library(igraph)
graph <- graph_from_data_frame(highschool)

ggraph(graph) + geom_edge_link() + geom_node_point() + theme_graph()
```

---

tree_apply *Apply a function recursively to a tree*

Description

This function allows for easy recursive function calling of tree-like structures. The recursion can happen either downwards, calling the children after the parent, or upwards, calling the parent after the children. At each recursion there is access to the node(s) that was/were called before and thus have already been through the function.

Usage

```
tree_apply(tree, FUN, ...)

## Default S3 method:
tree_apply(tree, FUN, ...)

## S3 method for class 'igraph'
tree_apply(tree, FUN, direction = "down", mode = "out",
  ...)

## S3 method for class 'dendrogram'
tree_apply(tree, FUN, direction = "down", ...)
```

Arguments

tree	A tree-like object. Currently support for igraph objects
FUN	The function to apply to each node. The function must return a modified version of the graph if `class(tree) == 'igraph'` or a modified version of the node if `class(tree) == 'dendrogram'`
...	Additional parameters to FUN
direction	The direction of the recursion. If `direction = 'down'` The parent will get handled before the children, while the reverse is true if `direction = 'up'`
mode	For `class(tree) == 'igraph'` the directionality of the edges in the graph. If `mode = 'out'` then parents points towards their children, while the reverse is true for `mode = 'in'`

Details

The function is called with a set of predifined parameters along with user defined ones. If `direction = 'down'` The parameters supplied automatically to the function are: node, parent, depth and tree, while if `direction = 'up'` the parameters are: node, children, depth and tree. The nature of node, parent and children depends on the class of tree. If `class(tree) == 'igraph'` they will be indices of the relevant vertices in the graph. If `class(tree) == 'dendrogram'` they will be the actual dendrogram objects.

Value

A modified version of tree (if anything is modified in FUN)

Examples

```
# We'll start with igraph
require(igraph)
gr <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

# Set depth and a class based on the name of the 2nd level node name
gr <- tree_apply(gr, function(node, parent, depth, tree) {
  tree <- set_vertex_attr(tree, 'depth', node, depth)
```

```
  if (depth == 1) {
    tree <- set_vertex_attr(tree, 'Class', node, V(tree)$shortName[node])
  } else if (depth > 1) {
    tree <- set_vertex_attr(tree, 'Class', node, V(tree)$Class[parent])
  }
  tree
})

# For dendrograms it's slightly different
irisDen <- as.dendrogram(hclust(dist(iris[1:4], method='euclidean'),
                         method='ward.D2'))
# Add the species information to the leafs
irisDen <- dendrapply(irisDen, function(d) {
  if(is.leaf(d))
    attr(d, 'nodePar') <- list(species=iris[as.integer(attr(d, 'label')),5])
  d
})

# Set class of node to the class of it's children they are of equal class
irisDen <- tree_apply(irisDen, function(node, children, ...) {
  if (is.leaf(node)) {
    attr(node, 'Class') <- attr(node, 'nodePar')$species
  } else {
    classes <- unique(sapply(children, attr, which = 'Class'))
    if (length(classes) == 1 && !anyNA(classes)) {
      attr(node, 'Class') <- classes
    } else {
      attr(node, 'Class') <- NA
    }
  }
  node
}, direction = 'up')
```

`whigs` *Membership network of American Whigs*

Description

This dataset shows the membership of 136 colonial Americans in 5 whig organization and is a bipartite graph. The data appeared in the appendix to David Hackett Fischer's *Paul Revere's Ride* (Oxford University Press, 1995) and compiled by Kieran Healy for the blog post Using Metadata to Find Paul Revere.

Usage

```
whigs
```

Format

The data is stored as an incidence matrix with persons as rows and organizations as columns. A 0 means no membership while a one means membership.

Source

https://github.com/kjhealy/revere/blob/master/data/PaulRevereAppD.csv adapted from:

Fischer, David H. (1995) *Paul Revere's Ride*. Oxford University Press

Index