Package ‘ggraph’

October 13, 2022

Type Package
Title An Implementation of Grammar of Graphics for Graphs and Networks
Version 2.1.0
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 MIT + file LICENSE
Encoding UTF-8
LazyData TRUE
Imports Rcpp (>= 0.12.2), dplyr, ggforce (>= 0.3.1), grid, igraph (>= 1.0.0), scales, MASS, digest, ggrepel, utils, stats, viridis, rlang, tidygraph, graphlayouts (>= 0.5.0), withr, cli, vctrs, lifecycle
Suggests network, knitr, rmarkdown, purrr, tibble, seriation, deldir, gganimate, covr
LinkingTo Rcpp
RoxygenNote 7.2.1
Depends R (>= 2.10), ggplot2 (>= 3.0.0)
VignetteBuilder knitr
URL https://ggraph.data-imaginist.com,
https://github.com/thomasp85/ggraph
BugReports https://github.com/thomasp85/ggraph/issues
NeedsCompilation yes
Author Thomas Lin Pedersen [cre, aut]
(<https://orcid.org/0000-0002-5147-4711>),
RStudio [cph]
Repository CRAN
Date/Publication 2022-10-09 20:33:19 UTC
R topics documented:

autograph .................................................. 3
facet_edges ............................................... 4
facet_graph ............................................... 6
facet_nodes ............................................... 8
flare ......................................................... 9
geometry .................................................... 10
geom_axis_hive ........................................... 11
geom_conn_bundle ....................................... 13
geom_edge_arc ............................................ 16
geom_edge_bend .......................................... 21
geom_edge_density ...................................... 26
geom_edge_diagonal ..................................... 28
geom_edge_elbow ......................................... 32
geom_edge_fan ............................................ 37
geom_edge_hive .......................................... 42
geom_edge_link .......................................... 47
geom_edge_loop .......................................... 51
geom_edge_parallel ..................................... 55
geom_edge_point ......................................... 60
geom_edge_span .......................................... 62
geom_edge_tile ........................................... 67
geom_node_arc_bar ...................................... 69
geom_node_circle ....................................... 70
geom_node_point ......................................... 72
geom_node_range ......................................... 74
geom_node_text .......................................... 75
geom_node_tile ........................................... 78
geom_node_voronoi ...................................... 80
get_con ..................................................... 82
get_edges .................................................. 83
get_nodes ................................................ 84
ggraph ...................................................... 85
guide_edge_colourbar ................................... 87
guide_edge_coloursteps ................................ 89
guide_edge_direction ................................... 91
highschool ............................................... 92
layout_tbl_graph_auto ................................ 93
layout_tbl_graph_backbone ............................. 94
layout_tbl_graph_centrality ......................... 95
layout_tbl_graph_circlepack ......................... 96
layout_tbl_graph_dendrogram ......................... 97
layout_tbl_graph_eigen ................................ 99
layout_tbl_graph_fabric ................................ 100
layout_tbl_graph_focus ................................. 101
layout_tbl_graph_hive ................................ 102
layout_tbl_graph_igraph ............................... 104
autograph

Description

This function is intended to quickly show an overview of your network data. While it returns a ggraph object that layers etc can be added to it is limited in use and should not be used as a foundation for more complicated plots. It allows colour, labeling and sizing of nodes and edges, and the exact combination of layout and layers will depend on these as well as the features of the network. The output of this function may be fine-tuned at any release and should not be considered stable. If a plot should be reproducible it should be created manually.

Usage

```r
autograph(graph, ...)
```

## Default S3 method:
autograph(
  graph,
  ...,
  node_colour = NULL,
  edge_colour = NULL,
  node_size = NULL,
  edge_width = NULL,
  node_label = NULL,
)
facet_edges

```r
  edge_label = NULL
)
```

Arguments

- `graph` An object coercible to a tbl_graph
- ... arguments passed on to methods
- `node_colour, edge_colour` Colour mapping for nodes and edges
- `node_size, edge_width` Size/width mapping for nodes and edges
- `node_label, edge_label` Label mapping for nodes and edges

Examples

```r
library(tidygraph)
gr <- create_notable('herschel') %>%
  mutate(class = sample(letters[1:3], n(), TRUE)) %>%
  mutate(weight = runif(n()))

# Standard graph
autograph(gr)

# Adding node labels will cap edges
autograph(gr, node_label = class)

# Use tidygraph calls for mapping
autograph(gr, node_size = centrality_pagerank())

# Trees are plotted as dendrograms
iris_tree <- hclust(dist(iris[1:4], method = 'euclidean'), method = 'ward.D2')
autograph(iris_tree)
```

Description

This function is equivalent to `ggplot2::facet_wrap()` but only facets edges. Nodes are repeated in every panel.
Usage

```r
facet_edges(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  switch = deprecated(),
  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, 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 `vars(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()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it 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.
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", "bottom", "left", "right")`

**See Also**

Other ggraph-facets: `facet_graph()`, `facet_nodes()`

**Examples**

```r
gr <- tidygraph::as_tbl_graph(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 `ggplot2::facet_grid()` in that it allows for building a grid of small multiples where rows and columns correspond to a specific data value. While `ggplot2::facet_grid()` could be used it would lead to unexpected results as it is not possible to specify whether you are referring to a node or an edge attribute. Furthermore `ggplot2::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**

```r
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 use rows and cols instead.
 row_type, col_type Either 'node' or 'edge'. Which data type is being faceted 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 vars(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(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it 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(tidygraph)
gr <- as_tbl_graph(highschool) %>%
  mutate(popularity = as.character(cut(centrality_degree(mode = 'in'), broken = 3),
  breaks = 3,
```r
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 `ggplot2::facet_wrap()` but only facets nodes. Edges are drawn if their terminal nodes are both present in a panel.

**Usage**

```r
code = facet_nodes(
  facets,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  shrink = TRUE,
  labeller = "label_value",
  as.table = TRUE,
  switch = deprecated(),
  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, 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 vars(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(). You can use different labeling functions for different kind of labels, for example use label_parsed() for formatting facet labels. label_value() is used by default, check it 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", "bottom", "left", "right")

See Also
Other ggraph-facets: facet_edges(), facet_graph()

Examples

library(tidygraph)
gr <- as_tbl_graph(highschool) %>%
mutate(popularity = as.character(cut(centrality_degree(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 hierarchical 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 col-
umn 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 ggplot2::margin() function
... Passed on to grid::gpar()
x An object to test for geometry inheritance

Details

gometry 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

gometry(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(). 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:

position     Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
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.
...

Aesthetics

geom_axis_hive understand the following aesthetics.

• alpha
• colour
• fill
• size
geom_conn_bundle

- linetype
- label_size
- family
- fontface
- lineheight

Author(s)

Thomas Lin Pedersen

Examples

```r
# Plot the flare import graph as a hive plot
library(tidygraph)
flareGr <- as_tbl_graph(flare$imports) %>%
  mutate(
    type = dplyr::case_when(
      centrality_degree(mode = 'in') == 0 ~ 'Source',
      centrality_degree(mode = 'out') == 0 ~ 'Sink',
      TRUE ~ 'Both'
    )
  ) %>%
  activate(edges) %>%
  mutate(
    type = dplyr::case_when(
      grepl('flare.analytics', paste(.N()$name[from], .N()$name[to])) ~ 'Analytics',
      TRUE ~ 'Other'
    )
  )

  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 hierarchical 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 follow the underlying hierarchical structure. This results in a flow-like structure where lines that partly move in the same direction will be bundled together.
Usage

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

```r
gem_conn_bundle2(
  mapping = NULL,
  data = get_con(),
  position = "identity",
  arrow = NULL,
  lineend = "butt",
  show.legend = NA,
  n = 100,
  tension = 0.8,
  ...
)
```

```r
gem_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 `ggplot2::aes()` or `ggplot2::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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
- `arrow`: Arrow specification, as created by `grid::arrow()`.
- `lineend`: Line end style (round, butt, square).
### geom_conn_bundle

- **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 off 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 `colour = "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

Examples

```r
# Create a graph of the flare class system
library(tidygraph)
flareGraph <- tbl_graph(flare$vertices, flare$edges) %>%
  mutate(
    class = map_bfs_chr(node_is_root(), .f = function(node, dist, path, ...) {
      if (dist <= 1) {
        return(shortName[node])
      }
      path$result[[nrow(path)]]
    })
  )
importFrom <- match(flare$imports$from, flare$vertices$name)
importTo <- match(flare$imports$to, flare$vertices$name)

# Use class inheritance for layout but plot class imports as bundles
ggraph(flareGraph, 'dendrogram', circular = TRUE) +
  geom_conn_bundle(aes(colour = after_stat(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_tbl_graph_linear()`), though it can be used elsewhere. It draws edges as arcs with a height 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**

```r
geom_edge_arc(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  n = 100,
  fold = FALSE,
)```
geom_edge_arc

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,
...,
curvature
)

geom_edge_arc2(
  mapping = NULL,
  data = get_edges("long"),
  position = "identity",
  arrow = NULL,
  strength = 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,
...,
curvature
)

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

Arguments

mapping Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

arrow Arrow specification, as created by `grid::arrow()`.

strength 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 `grDevices::plotmath()`.

check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

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 `grid::unit()` giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'

label_push A `grid::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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

curvature Deprecated. Use strength instead.

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.

- startCap
- endCap
- label
- labelPos
• label_size
• angle
• hjust
• vjust
• family
• fontface
• lineheight

Computed variables

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

Edge variants

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 = after_stat(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 considerably 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 on top 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.

Edge aesthetic name expansion

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_bend(), geom_edge_density(), geom_edge_diagonal(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_link(), geom_edge_loop(), geom_edge_parallel(), geom_edge_point(), geom_edge_span(), geom_edge_tile()

Examples

require(tidygraph)
# Make a graph with different directions of edges
gr <- create_notable('Meredith') %>%
  convert(to_directed) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE)) %>%
  activate(edges) %>%
  mutate(
    class = sample(letters[1:3], n(), replace = TRUE),
    switch = sample(c(TRUE, FALSE), n(), replace = TRUE)
  ) %>%
  reroute(from = to, to = from, subset = switch)

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

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

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

---

**geom_edge_bend**  
**Draw edges as diagonals**

Description

This geom draws edges as cubic bezier curves with the control points positioned along the elbow edge. It has the appearance of a softened elbow edge with the hard angle substituted by a tapered bend.

Usage

geom_edge_bend(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  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_bend2(
  mapping = NULL,
  data = get_edges("long"),
  position = "identity",
  arrow = NULL,
  strength = 1,
  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_bend0(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  flipped = FALSE,
  lineend = "butt",
  show.legend = NA,
  ...
)
Arguments

**mapping**
Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

**arrow**
Arrow specification, as created by `grid::arrow()`.

**strength**
The strength of the curvature of the bend. 0 will result in a straight line while 1 will give a strong arc.

**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 `grDevices::plotmath()`.

**check_overlap**
If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

**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 is on it's head. Default to TRUE.

**label_dodge**
A `grid::unit()` giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'

**label_push**
A `grid::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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
Aesthetics

`geom_edge_bend` and `geom_edge_bend0` 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_bend2` 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_bend` and `geom_edge_bend2` 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)

Edge variants

Many `geom_edge_*` layers come 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 = after_stat(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 considerably 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 on top 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.

Edge aesthetic name expansion

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()`, `geom_edge_loop()`, `geom_edge_parallel()`, `geom_edge_point()`, `geom_edge_span()`, `geom_edge_tile()`
Examples

```r
require(tidygraph)

gr <- create_tree(20, 4) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE)) %>%
  activate(edges) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE))

ggraph(gr, 'tree') +
  geom_edge_bend(aes(alpha = after_stat(index)))

ggraph(gr, 'tree') +
  geom_edge_bend2(aes(colour = node.class))

ggraph(gr, 'tree') +
  geom_edge_bend0(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

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

Arguments

- **mapping**: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
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 colour = “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

Edge aesthetic name expansion

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_bend(), geom_edge_diagonal(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_link(), geom_edge_loop(), geom_edge_parallel(), geom_edge_point(), geom_edge_span(), geom_edge_tile()

Examples

require(tidygraph)
gr <- create_notable(‘bull’) %>%
activate(edges) %>%
mutate(class = sample(letters[1:3], n(), replace = TRUE))

ggraph(gr, ‘stress’) +
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. The name comes from D3.js where this shape was called diagonals until it was renamed to links. 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 hierarchical layouts as it implies an overall directionality in the plot.

**Usage**
```r
gem_edge_diagonal(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  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,
  ...
)
```
```r
gem_edge_diagonal2(
  mapping = NULL,
  data = get_edges("long"),
  position = "identity",
  arrow = NULL,
  strength = 1,
  flipped = FALSE,
  n = 100,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
```
Arguments

mapping Set of aesthetic mappings created by ggplot2::aes() or ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

arrow Arrow specification, as created by grid::arrow().

strength The strength of the curvature of the diagonal. 0 will result in a straight line while 1 will give the familiar S-shape.

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.
If TRUE, the labels will be parsed into expressions and displayed as described in `grDevices::plotmath()`.

If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

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.

Logical. If angle_calc is either 'along' or 'across' should the label be flipped if it is on it's head. Default to TRUE.

A `grid::unit()` giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'

A `grid::unit()` giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'

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 `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

**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`
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)

**Edge variants**

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 = after_stat(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 on top 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
geom_edge_elbow

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.

Edge aesthetic name expansion

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_bend(), geom_edge_density(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_link(), geom_edge_loop(), geom_edge_parallel(), geom_edge_point(), geom_edge_span(), geom_edge_tile()

Examples

require(tidygraph)
gr <- create_tree(20, 4) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE)) %>%
  activate(edges) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE))

ggraph(gr, 'tree') +
  geom_edge_diagonal(aes(alpha = after_stat(index)))

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

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

g_graph(g, 'tree') +
  geom_edge_diagonal2(aes(colour = node.class))

g_graph(g, 'tree') +
  geom_edge_diagonal0(aes(colour = class))
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_tbl_graph_dendrogram()`, `layout_tbl_graph_partition()` and `layout_tbl_graph_igraph()` with the "tree" algorithm).

Usage

```r
geom_edge_elbow(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  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,
  ...)
```

```r
gem_edge_elbow2(
  mapping = NULL,
  data = get_edges("long"),
  position = "identity",
  arrow = NULL,
  strength = 1,
  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,
  ...
)
```
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 \texttt{ggplot2::aes()} or \texttt{ggplot2::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 \texttt{get_edges()} or a data.frame giving edges in correct format (see details for for guidance on the format). See \texttt{get_edges()} for more details on edge extraction.
position Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use \texttt{position_jitter}), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
arrow Arrow specification, as created by \texttt{grid::arrow()}.
strength How bend the elbow should be. 1 will give a right angle, while 0 will give a straight line. Ignored for circular layouts
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 \texttt{grDevices::plotmath()).
check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling \texttt{geom_text()}. Note that this argument is not supported by \texttt{geom_label()}.
geom_edge_elbow

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 grid::unit() giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'

label_push A grid::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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

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)

**Edge variants**

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 = after_stat(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 considerably 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 on top 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.

Edge aesthetic name expansion

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_bend()`, `geom_edge_density()`, `geom_edge_diagonal()`, `geom_edge_fan()`, `geom_edge_hive()`, `geom_edge_link()`, `geom_edge_loop()`, `geom_edge_parallel()`, `geom_edge_point()`, `geom_edge_span()`, `geom_edge_tile()`

Examples

```r
require(tidygraph)
irisDen <- hclust(dist(iris[1:4], method = "euclidean"), method = "ward.D2") %>%
  as_tbl_graph() %>%
  mutate(class = sample(letters[1:3], n(), TRUE)) %>%
  activate(edges) %>%
  mutate(class = sample(letters[1:3], n(), TRUE))

ggraph(irisDen, "dendrogram", circular = TRUE) +
  geom_edge_elbow(aes(alpha = after_stat(index)))

ggraph(irisDen, "dendrogram") +
  geom_edge_elbow2(aes(colour = node.class))

ggraph(irisDen, "dendrogram", height = height) +
  geom_edge_elbow0(aes(colour = class))
```

---

`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 `igraph::plot.igraph()`. Before calculating the curvature the edges are sorted by direction so that edges going 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,
    strength = 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,
    strength = 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(}
geom_edge_fan

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

Arguments

mapping Set of aesthetic mappings created by ggplot2::aes() or ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
arrow Arrow specification, as created by grid::arrow().
strength Modify the width of the fans strength > 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 grDevices::plotmath().
check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().
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 grid::unit() giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push  A `grid::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 `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

spread  Deprecated. Use `strength` instead.

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` understands 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)

Edge variants

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 = after_stat(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 on top 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.

Edge aesthetic name expansion

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_bend()`, `geom_edge_density()`, `geom_edge_diagonal()`, `geom_edge_elbow()`, `geom_edge_hive()`, `geom_edge_link()`, `geom_edge_loop()`, `geom_edge_parallel()`, `geom_edge_point()`, `geom_edge_span()`, `geom_edge_tile()`

Examples

```r
require(tidygraph)
gr <- create_notable("bull") %>%
  convert(to_directed) %>%
  bind_edges(data.frame(from = c(1, 2, 2, 3), to = c(2, 1, 3, 2))) %>
  mutate(class = sample(letters[1:3], 9, TRUE)) %>
  mutate(class = sample(c("x", "y"), 5, TRUE))
ggraph(gr, "stress") +
geom_edge_fan(aes(alpha = after_stat(index)))
ggraph(gr, "stress") +
geom_edge_fan2(aes(colour = node.class))
ggraph(gr, "stress") +
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 radii as the start or end point, and at a distance defined by the curvature argument.

**Usage**

```r
geom_edge_hive(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  strength = 1,
  n = 100,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
```
geom_edge_hive

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,
..., curvature
)

geom_edge_hive2(
  mapping = NULL,
data = get_edges("long"),
position = "identity",
arrow = NULL,
strength = 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,
..., curvature
)

geom_edge_hive0(
  mapping = NULL,
data = get_edges(),
position = "identity",
arrow = NULL,
strength = 1,
lineend = "butt",
show.legend = NA,
..., curvature
)
Arguments

mapping Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct
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 naming the adjustment (e.g. "jitter" to
use `position_jitter`), or the result of a call to a position adjustment function.
Use the latter if you need to change the settings of the adjustment.

arrow Arrow specification, as created by `grid::arrow()`.

strength 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 halfway between the nodes and the middle of the two axes, 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
`grDevices::plotmath()`.

check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted.
check_overlap happens at draw time and in the order of the data. Therefore
data should be arranged by the label column before calling `geom_text()`. Note
that this argument is not supported by `geom_label()`.

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 `grid::unit()` giving a fixed vertical shift to add to the label in case of
angle_calc is either ‘along’ or ‘across’

label_push A `grid::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.

curvature Deprecated. Use `strength` instead.
Aesthetics

text

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)
Edge variants

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 = after_stat(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 on top 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.

Edge aesthetic name expansion

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_bend()`, `geom_edge_density()`, `geom_edge_diagonal()`, `geom_edge_elbow()`, `geom_edge_fan()`, `geom_edge_link()`, `geom_edge_loop()`, `geom_edge_parallel()`, `geom_edge_point()`, `geom_edge_span()`, `geom_edge_tile()`

Examples

# Plot the flare import graph as a hive plot
library(tidygraph)
flareGr <- as_tbl_graph(flare$imports) %>%
  mutate(
    type = dplyr::case_when(
      ...
```r
centrality_degree(mode = 'in') == 0 ~ 'Source',
centrality_degree(mode = 'out') == 0 ~ 'Sink',
TRUE ~ 'Both'
}
) %>%
activate(edges) %>%
mutate(
  type = dplyr::case_when(
    grepl('flare.analytics', paste(.N()$name[from], .N()$name[to])) ~ 'Analytics',
    TRUE ~ 'Other'
  )
)

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**

```r
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 `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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 grDevices::plotmath().
check_overlap  If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().
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 grid::unit() giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push  A grid::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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

Edge variants

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 = after_stat(index). The version postfixied 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 considerably 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 on top 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.

**Edge aesthetic name expansion**

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.

**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`
.geom_edge_loop

- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

Computed variables

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

Author(s)

Thomas Lin Pedersen

See Also

Other geom_edge_*: geom_edge_arc(), geom_edge_bend(), geom_edge_density(), geom_edge_diagonal(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_loop(), geom_edge_parallel(), geom_edge_point(), geom_edge_span(), geom_edge_tile()

Examples

```r
require(tidygraph)
gr <- create_notable("bull") %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE)) %>%
  activate(edges) %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE))
ggraph(gr, 'stress') +
  geom_edge_link(aes(alpha = after_stat(index)))
ggraph(gr, 'stress') +
  geom_edge_link2(aes(colour = node.class))
ggraph(gr, 'stress') +
  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

```r
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,
  ...
)
```

gem_edge_loop0(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  show.legend = NA,
  ...
)

Arguments

- **mapping**: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct format (see details for for guidance on the format). See `get_edges()` for more ...)
geom_edge_loop

details on edge extraction.

position Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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 grDevices::plotmath().

check_overlap If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().

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 grid::unit() giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'

label_push A grid::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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

gem_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**

gem_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)

**Edge variants**

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 = after_stat(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 on top 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.

**Edge aesthetic name expansion**

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_bend(), geom_edge_density(), geom_edge_diagonal(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_link(), geom_edge_parallel(), geom_edge_point(), geom_edge_span(), geom_edge_tile()

**Examples**

```r
easure(tidygraph)
gr <- as_tbl_graph(
  data.frame(from = c(1, 1, 2, 2, 3, 3, 3), to = c(1, 2, 2, 3, 3, 1, 2))
)

ggplot(gr, 'stress') +
  geom_edge_loop(aes(alpha = after_stat(index))) +
  geom_edge_fan(aes(alpha = after_stat(index)))

ggplot(gr, 'stress') +
  geom_edge_loop0() +
  geom_edge_fan0()
```

---

**geom_edge_parallel**

*Draw multi edges as parallel lines*

**Description**

This geom draws multi edges as parallel lines. The edges are first sorted by direction and then shifted a fixed amount so that all edges are visible.
Usage

geom_edge_parallel(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  arrow = NULL,
  sep = unit(2, "mm"),
  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_parallel2(
  mapping = NULL,
  data = get_edges("long"),
  position = "identity",
  arrow = NULL,
  sep = unit(2, "mm"),
  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_parallel0(
  mapping = NULL,
  data = get_edges(),
  ...
Arguments

- **mapping**: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use `position_jitter()`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

- **arrow**: Arrow specification, as created by `grid::arrow()`.

- **sep**: The separation between parallel edges, given as a `grid::unit()`

- **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 `grDevices::plotmath()`.

- **check_overlap**: If TRUE, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

- **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 `grid::unit()` giving a fixed vertical shift to add to the label in case of `angle_calc` is either 'along' or 'across'

- **label_push**: A `grid::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 `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Aesthetics

`geom_edge_parallel` and `geom_edge_parallel0` 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_parallel2` 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_parallel` and `geom_edge_parallel2` 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)

Edge variants

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 = after_stat(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 considerably 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 on top 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.

Edge aesthetic name expansion

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.
Draw edges as glyphs

This geom draws edges as glyphs with their x-position defined by the x-position of the start node, and the y-position defined by the y-position of the end node. As such it will result in a matrix layout when used in conjunction with `layout_tbl_graph_matrix()`.

```r
require(tidygraph)
gr <- create_notable('bull') %>%
  convert(to_directed) %>%
  bind_edges(data.frame(from = c(1, 2, 2, 3), to = c(2, 1, 3, 2))) %>%
  mutate(class = sample(letters[1:3], 9, TRUE)) %>%
  mutate(class = sample(c('x', 'y'), 5, TRUE))
ggraph(gr, 'stress') +
  geom_edge_parallel(aes(alpha = after_stat(index)))
ggraph(gr, 'stress') +
  geom_edge_parallel2(aes(colour = node.class))
ggraph(gr, 'stress') +
  geom_edge_parallel0(aes(colour = class))
# Use capping and sep to fine tune the look
  geom_edge_parallel(start_cap = circle(1), end_cap = circle(1),
                     arrow = arrow(length = unit(2, 'mm'), sep = unit(4, 'mm')) +
                     geom_node_point(size = 12))
```

Usage

```r
geom_edge_point(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
)```
geom_edge_point

mirror = FALSE,
show.legend = NA,
...
)

Arguments

mapping
Set of aesthetic mappings created by ggplot2::aes() or ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

mirror
Logical. Should edge points be duplicated on both sides of the diagonal. Intended for undirected graphs. Default to FALSE

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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

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

• x
• y
• edge_shape
• edge_colour
• edge_size
• edge_alpha
• filter

Edge aesthetic name expansion

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
geom_edge_span

**Description**

This edge geom is mainly intended for use with fabric layouts. It draws edges as vertical segments with an optional end shape adornment. Due to the special nature of fabric layouts where nodes are not a single point in space but a line, this geom doesn’t derive the x position from the location of the terminal nodes, but defaults to using the edge_x variable calculated by the fabric layout. If this geom is used with other layouts x and xend must be given explicitly.

**Usage**

```r
geom_edge_span(
  mapping = NULL,
  data = get_edges("short"),
  position = "identity",
  end_shape = NA,
  arrow = NULL,
  n = 100,
  lineend = "butt",
  linejoin = "round",
  linemitre = 1,
  label_colour = "black",
  label_alpha = 1,
  label_parse = FALSE,
```
Arguments

mapping  Set of aesthetic mappings created by ggplot2::aes() or ggplot2::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 correct
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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

**end_shape** The adornment to put at the ends of the span. The naming follows the conventions of the shape aesthetic in `ggplot2::geom_point()`

**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 `grDevices::plotmath()`.

**check_overlap** If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

**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 `grid::unit()` giving a fixed vertical shift to add to the label in case of `angle_calc` is either 'along' or 'across'

**label_push** A `grid::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 `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

### Aesthetics

`geom_edge_span` and `geom_edge_span0` understand the following aesthetics. Bold aesthetics are automatically set, but can be overridden.

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

- xend
- yend
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- filter

geom_edge_span2 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_span and geom_edge_span2 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)
Edge variants

Many `geom_edge_*` layers come 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 = after_stat(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 considerably 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 on top 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.

Edge aesthetic name expansion

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_bend()`, `geom_edge_density()`, `geom_edge_diagonal()`, `geom_edge_elbow()`, `geom_edge_fan()`, `geom_edge_hive()`, `geom_edge_link()`, `geom_edge_loop()`, `geom_edge_parallel()`, `geom_edge_point()`, `geom_edge_tile()`

Examples

```r
require(tidygraph)
gr <- play_smallworld(n_dim = 3, dim_size = 3, order = 1, p_rewire = 0.6)

# Standard use
ggraph(gr, 'fabric', sort.by = node_rank_fabric()) +
```
### geom_edge_tile

**Draw edges as glyphs**

**Description**

This geom draws edges as tiles with their x-position defined by the x-position of the start node, and the y-position defined by the y-position of the end node. As such it will result in a matrix layout when used in conjunction with `layout_tbl_graph_matrix()`.

**Usage**

```r
geom_edge_tile(
  mapping = NULL,
  data = get_edges(),
  position = "identity",
  mirror = FALSE,
  show.legend = NA,
  ...
)
```

**Arguments**

- **mapping**
  Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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 correct 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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

- **mirror**
  Logical. Should edge points be duplicated on both sides of the diagonal. Intended for undirected graphs. Default to FALSE.
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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

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

- x
- y
- edge_fill
- edge_colour
- edge_size
- edge_alpha
- filter

Edge aesthetic name expansion

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_bend(), geom_edge_density(), geom_edge_diagonal(), geom_edge_elbow(), geom_edge_fan(), geom_edge_hive(), geom_edge_link(), geom_edge_loop(), geom_edge_parallel(), geom_edge_point(), geom_edge_span()

Examples

```r
require(tidygraph)
gr <- create_notable('zachary') %>%
    mutate(group = group_infomap()) %>%
    morph(to_split, group) %>%
    activate(edges) %>%
    mutate(edge_group = as.character(.N()$group[1])) %>%
    unmorph()

ggraph(gr, 'matrix', sort.by = node_rank_hclust()) +
    geom_edge_tile(aes(fill = edge_group), mirror = TRUE) +
    scale_y_reverse()
```
coord_fixed() +
labs(edge_colour = 'Infomap Cluster') +
ggtitle("Zachary Karate Club")

**geom_node_arc_bar**  
*Show nodes as thick arcs*

**Description**

This geom is equivalent in functionality to `ggforce::geom_arc_bar()` and allows for plotting of nodes as arcs with an inner and outer radius scaled by the coordinate system. Its main use is currently in sunburst plots as created with circular partition layouts.

**Usage**

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

**Arguments**

- **mapping**: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::aes()`. By default x and y are mapped to x0 and y0 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. A function can be created from a formula (e.g. `~ head(.x, 10)`).
- **position**: Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
- **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 `colour = "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.

- `x0`
- `y0`
- `r0`
- `r`
- `start`
- `end`
- `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_range()`, `geom_node_text()`, `geom_node_tile()`, `geom_node_voronoi()`

Examples

```r
require(tidygraph)
gr <- tbl_graph(flare$vertices, flare$edges)
ggraph(gr, 'partition', circular = TRUE, weight = size) +
  geom_node_arc_bar()
```

---

**geom_node_circle**  Show nodes as circles

Description

This geom is equivalent in functionality to `ggforce::geom_circle()` and allows for plotting of nodes as circles with a radius scaled by the coordinate system. Because of the geoms reliance on the coordinate system it will only produce true circles when combined with `ggplot2::coord_fixed()`
Usage

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

Arguments

- **mapping**: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::aes()`. By default x and y are mapped to x0 and y0 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. A function can be created from a formula (e.g. `~ head(.x, 10)`).
- **position**: Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
- **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 `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Aesthetics

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

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

- stroke
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar(), geom_node_point(), geom_node_range(), geom_node_text(), geom_node_tile(), geom_node_voronoi()

Examples

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

Description

This geom is equivalent in functionality to `ggplot2::geom_point()` and allows for simple plotting of nodes in different shapes, colours and sizes.

Usage

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

Arguments

- mapping: Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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. A function can be created from a formula (e.g. `~ head(.x, 10)`).

**position** Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

**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 `colour = "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_range()
- geom_node_text()
- geom_node_tile()
- geom_node_voronoi()

### Examples

```r
require(tidygraph)
gr <- create_notable('bull') %>%
  mutate(class = sample(letters[1:3], n(), replace = TRUE))

ggraph(gr, 'stress') + geom_node_point()
```
**Description**

This geom is most useful together with the fabric layout for showing the horizontal span of each node.

**Usage**

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

**Arguments**

- **mapping**
  Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::aes()`. By default x is mapped to xmin, xend is mapped to xmax and y and yend are mapped to 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. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **position**
  Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

- **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 `colour = "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
- xend
- y
- yend
- alpha
- colour
- linetype
- size
- filter

Author(s)

Thomas Lin Pedersen

See Also

Other geom_node_*: geom_node_arc_bar(), geom_node_circle(), geom_node_point(), geom_node_text(), geom_node_tile(), geom_node_voronoi()

Examples

```r
require(tidygraph)
gr <- as_tbl_graph(highschool)
ggraph(gr, layout = 'fabric') +
  geom_node_range()
```

<table>
<thead>
<tr>
<th>geom_node_text</th>
<th>Annotate nodes with text</th>
</tr>
</thead>
</table>

Description

These geoms are equivalent in functionality to ggplot2::geom_text() and ggplot2::geom_label() and allows for simple annotation of nodes.
Usage

```r
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,
  ...
)```

```r
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 `ggplot2::aes()` or `ggplot2::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 variables 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. A function can be created from a `formula` (e.g. `~ head(.x, 10)`).

- **position** | Position adjustment, either as a string, or the result of a call to a position adjustment function. Cannot be jointly specified with nudge_x or nudge_y.

- **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. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().

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 colour = "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_text understands 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_range(), geom_node_tile(), geom_node_voronoi()
Examples

```r
require(tidygraph)
gr <- create_notable('bull') %>%
    mutate(class = sample(letters[1:3], n(), replace = TRUE))

ggraph(gr, 'stress') +
    geom_node_point aes(label = class))

ggraph(gr, 'stress') +
    geom_node_label(aes(label = class), repel = TRUE)
```

---

**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**

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

**Arguments**

- **mapping**
  Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::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. A function can be created from a formula (e.g. `~ head(.x, 10)`).
position

Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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 colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

gem_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_range(), geom_node_text(), geom_node_voronoi()
geom_node_voronoi

Show nodes as voronoi tiles

Description

This geom is equivalent in functionality to ggforce::geom_voronoi_tile() and allows for plotting of nodes as tiles from a voronoi tesselation. As with ggforce::geom_voronoi_tile() it is possible to restrict the size of the tile to a fixed radius, as well as round corners and expand/contract the tile.

Usage

geom_node_voronoi(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  bound = NULL,
  eps = 1e-09,
  max.radius = NULL,
  normalize = FALSE,
  asp.ratio = 1,
  expand = 0,
  radius = 0,
  ...
)

Arguments

mapping

Set of aesthetic mappings created by ggplot2::aes() or ggplot2::aes(). By default x and y are mapped to x and y in the node data and group set to -1.

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. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom_node_voronoi

position
Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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.

bound
The bounding rectangle for the tesselation or a custom polygon to clip the tessellation to. Defaults to NULL which creates a rectangle expanded 10\ vector giving the bounds in the following order: xmin, xmax, ymin, ymax. If supplied as a polygon it should either be a 2-column matrix or a data.frame containing an x and y column.

eps
A value of epsilon used in testing whether a quantity is zero, mainly in the context of whether points are collinear. If anomalous errors arise, it is possible that these may averted by adjusting the value of eps upward or downward.

max.radius
The maximum distance a tile can extend from the point of origin. Will in effect clip each tile to a circle centered at the point with the given radius. If normalize = TRUE the radius will be given relative to the normalized values

normalize
Should coordinates be normalized prior to calculations. If x and y are in wildly different ranges it can lead to tesselation and triangulation that seems off when plotted without \texttt{ggplot2::coord_fixed()}. Normalization of coordinates solves this. The coordinates are transformed back after calculations.

asp.ratio
If normalize = TRUE the x values will be multiplied by this amount after normalization.

expand
A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.

radius
As expand but specifying the corner radius.

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

Aesthetics

\texttt{geom_node_voronoi} 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_point(), geom_node_range(), geom_node_text(), geom_node_tile()

Examples
```r
require(tidygraph)
gr <- create_notable('meredith') %>%
  mutate(group = sample(letters[1:4], n(), TRUE))

ggraph(gr) +
  geom_node_voronoi(aes(fill = group, colour = group), alpha = 0.3) +
  geom_edge_link(alpha = 0.3) +
  geom_node_point()

# Use max.radius to make the tessellation more "node"-like

ggraph(gr) +
  geom_node_voronoi(aes(fill = group, colour = group), alpha = 0.3, max.radius = 1) +
  geom_edge_link(alpha = 0.3) +
  geom_node_point()
```

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 do 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 nodes and automatically finds 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 indices that define a connection.

Usage
```r
get_con(
  from = integer(),
  to = integer(),
  paths = NULL,
  ...,
  weight = NULL,
  mode = "all"
)
```
get_edges

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
weight  An expression to be evaluated on the edge data to provide weights for the shortest path calculations
mode  Character constant, gives whether the shortest paths to or from the given vertices should be calculated for directed graphs. If out then the shortest paths from the vertex, if in then to it will be considered. If all, the default, then the corresponding undirected graph will be used, i.e. not directed paths are searched. This argument is ignored for undirected graphs.

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.
get_nodes

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 `ggplot2::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 occurring 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 occurring 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.
ggraph

Usage
get_nodes(...)

Arguments
... Additional data that should be cbind'ed together 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 ggplot2::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 description 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 'layout_ggraph'
create_layout(graph, ...)

## S3 method for class 'tbl_graph'
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. Either a valid string, a function, a matrix, or a data.frame (see Details)
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. Any type of regular graph/network data can be represented as a tbl_graph object. Because of this the different layouts that can be applied to tbl_graph 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 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.

Any object that has an appropriate as_tbl_graph method can be passed into ggraph() and will automatically be converted underneath.

auto The default layout. See layout_tbl_graph_auto() for further details

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_tbl_graph_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_tbl_graph_dendrogram() for further details

manual Lets the user manually specify the location of each node. See layout_tbl_graph_manual() for further details

linear Arranges the nodes linearly or circularly in order to make an arc diagram. See layout_tbl_graph_linear() for further details

matrix Arranges nodes on a diagonal thus preparing it for use with geom_edge_point() to make a matrix plot. See layout_tbl_graph_matrix() for further details
treemap Creates a treemap from the graph, that is, a space-filing subdivision of rectangles showing a weighted hierarchy. See layout_tbl_graph_treemap() for further details
circlepack Creates a layout showing a hierarchy as circles within circles. Conceptually equal to treemaps. See layout_tbl_graph_circlepack() for further details

partition Create icicle or sunburst charts, where each layer subdivides the division given by the preceding layer. See layout_tbl_graph_partition() for further details

tree create Positions nodes on axes spreading out from the center based on node attributes. See layout_tbl_graph_hive() for further details

Alternatively a matrix or a data.frame can be provided to the layout argument. In the former case the first column will be used as x coordinates and the second column will by used as y coordinates, further columns are dropped. In the latter case the data.frame is used as the layout table and must thus contain a numeric x and y column.

Lastly a function can be provided to the layout argument. It will be called with the graph object as its first argument and any additional argument passed into ggraph()/create_layout(). The function must return either a data.frame or an object coercible to one and have an x and y column, or an object coercible to a tbl_graph. In the latter case the node data is extracted and used as layout (and must thus contain an x and y column) and the graph will be added as the graph attribute.
guide_edge_colourbar

Value

For ggraph() an object of class gg onto which layers, scales, etc. can be added. For create_layout() an object inheriting 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

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

---

guide_edge_colourbar  Colourbar legend for edges

Description

This function is equivalent to ggplot2::guide_colourbar() but works for edge aesthetics.

Usage

```r
guide_edge_colourbar(..., available_aes = c("edge_colour", "edge_fill"))
guide_edge_colorbar(..., available_aes = c("edge_colour", "edge_fill"))
```

Arguments

...  Arguments passed on to ggplot2::guide_colourbar
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.
guide_edge_colourbar

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().

label.hjust A numeric specifying horizontal justification of the label text.

The default for standard text is 0 (left-aligned) and 1 (right-aligned) for expressions.

ticks A logical specifying if tick marks on the colourbar should be visible.

ticks.colour A string specifying the colour of the tick marks.

ticks.linewidth A numeric specifying the width of the tick marks in millimetres.

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 colourbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom.

order positive integer less than 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.
guide_edge_coloursteps

available_aes  A vector of character strings listing the aesthetics for which a colourbar can be drawn.

Value

A guide object

guide_edge_coloursteps

Coloursteps legend for edges

Description

This function is equivalent to `ggplot2::guide_coloursteps()` but works for edge aesthetics.

Usage

```r
guide_edge_coloursteps(
  even.steps = TRUE,
  show.limits = NULL,
  ticks = FALSE,
  ...
)
```

Arguments

- `even.steps`  Should the rendered size of the bins be equal, or should they be proportional to their length in the data space? Defaults to TRUE
- `show.limits` Logical. Should the limits of the scale be shown with labels and ticks. Default is NULL meaning it will take the value from the scale. This argument is ignored if `labels` is given as a vector of values. If one or both of the limits is also given in `breaks` it will be shown irrespective of the value of `show.limits`.
- `ticks` A logical specifying if tick marks on the colourbar should be visible.
- `...` Arguments passed on to `ggplot2::guide_colourbar`
- `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()`.

label.hjust A numeric specifying horizontal justification of the label text. The default for standard text is 0 (left-aligned) and 1 (right-aligned) for expressions.

label.vjust A numeric specifying vertical justification of the label text.

barwidth A numeric or a `grid::unit()` object specifying the width of the colourbar. 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 colourbar. Default value is `legend.key.height` or `legend.key.size` in `theme()` or theme.

nbin A numeric specifying the number of bins for drawing the colourbar. A smoother colourbar results from a larger value.

raster A logical. If TRUE then the colourbar is rendered as a raster object. If FALSE then the colourbar is rendered as a set of rectangles. Note that not all graphics devices are capable of rendering raster image.

frame.colour A string specifying the colour of the frame drawn around the bar. If NULL (the default), no frame is drawn.

frame.linewidth A numeric specifying the width of the frame drawn around the bar in millimetres.

frame.linetype A numeric specifying the linetype of the frame drawn around the bar.

ticks.colour A string specifying the colour of the tick marks.

ticks.linewidth A numeric specifying the width of the tick marks in millimetres.

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 colourbar is reversed. By default, the highest value is on the top and the lowest value is on the bottom.

order  positive integer less than 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.

available_aes  A vector of character strings listing the aesthetics for which a colourbar can be drawn.

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,
  override.aes = list(),
  ...
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>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.</td>
</tr>
</tbody>
</table>
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


Description

This function infers the layout from the graph structure and is the default when calling `ggraph()`. If an `x` and `y` argument is passed along, the manual layout is chosen. Otherwise if the graph is either a rooted tree or a rooted forest the layout will be `dendrogram` if the nodes contains a height variable or `tree` if not. If the tree is unrooted the unrooted layout will be used. If the tree is a DAG the `sygiyama` layout will be used. Otherwise the `stress` layout will be used (or `sparse_tree` if the graph contains more than 2000 nodes).

Usage

`layout_tbl_graph_auto(graph, circular, ...)`

Arguments

graph A tbl_graph object
circular Logical. Should the layout be transformed to a circular representation. Defaults to FALSE. Only applicable if the graph is a tree structure

... Arguments passed on to the chosen layout

Value

A data.frame with the columns `x, y, circular` as well as any information stored as node variables in the tbl_graph object.
See Also

Other layout_tbl_graph_*: layout_tbl_graph_backbone(), layout_tbl_graph_centrality(),
layout_tbl_graph_circlepack(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(),
layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_igraph(),
layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(),
layout_tbl_graph_pmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted.

layout_tbl_graph_backbone

*Place node to emphasize group structure*

Description

This layout is optimised for drawing small-world types of graphs often found in social networks,
where distinct groups are still highly connected to the remaining graph. Typical layouts struggle
with this as they attempt to minimise the edge length of all edges equally. The backbone layout is
based on weighing edges based on how well they hold together communities. The end result is that
communities tend to stick together despite high interconnectivity.

Usage

layout_tbl_graph_backbone(graph, keep = 0.2, circular = FALSE)

Arguments

graph A tbl_graph object
keep The fraction of edges to use for creating the backbone
circular ignored

Value

A data.frame with the columns x, y, circular as well as any information stored as node variables
in the tbl_graph object. Further an edge attribute called backbone is added giving whether the edge
was selected as backbone.

Author(s)

The underlying algorithm is implemented in the graphlayouts package by David Schoch

References

Nocaj, A., Ortmann, M., & Brandes, U. (2015). Untangling the hairballs of multi-centered, small-
world online social media networks. Journal of Graph Algorithms and Applications: JGAA, 19(2),
595-618.
**layout_tbl_graph_centrality**

Place nodes in circles according to centrality measure

**Description**

This layout places nodes in circles with the radii relative to a given centrality measure. Under the hood it uses stress majorisation to place nodes optimally given the radius constraint.

**Usage**

```r
layout_tbl_graph_centrality(
  graph,
  centrality,
  scale = TRUE,
  niter = 500,
  tolerance = 1e-04,
  tseq = seq(0, 1, 0.2),
  circular = FALSE
)
```

**Arguments**

- `graph`: A tbl_graph object
- `centrality`: An expression evaluating to a centrality measure for the nodes. See the different `centrality_*()` algorithms in tidygraph for a selection.
- `scale`: Should the centrality measure be scaled between 0 and 100
- `niter`: number of iterations during stress optimization
- `tolerance`: stopping criterion for stress optimization
- `tseq`: Transitioning steps
- `circular`: ignored

**Value**

A data.frame with the columns `x`, `y`, `circular`, `centrality` as well as any information stored as node variables in the tbl_graph object.
**Author(s)**

The underlying algorithm is implemented in the graphlayouts package by David Schoch

**References**


**See Also**

Other layout_tbl_graph_*: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`

---

`layout_tbl_graph_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 efficiently 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**

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>graph</code></td>
<td>An tbl_graph object</td>
</tr>
<tr>
<td><code>weight</code></td>
<td>An optional node variable to use as weight. Will only affect the weight of leaf nodes as the weight of non-leaf nodes are derived from their children.</td>
</tr>
<tr>
<td><code>circular</code></td>
<td>Logical. Should the layout be transformed to a circular representation. Ignored.</td>
</tr>
<tr>
<td><code>sort.by</code></td>
<td>The name of a node variable to sort the nodes by.</td>
</tr>
<tr>
<td><code>direction</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
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 the 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 Bostock’s JavaScript implementation.

Value

A data.frame with the columns x, y, r, leaf, depth, circular as well as any information stored as node variables in the tbl_graph 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


See Also

Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(), layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_igraph(), layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(), layout_tbl_graph_pmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted

Description

This layout mimics the igraph::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, or read from a node variable.
Usage

```r
layout_tbl_graph_dendrogram(
  graph,
  circular = FALSE,
  offset = pi/2,
  height = NULL,
  length = NULL,
  repel = FALSE,
  ratio = 1,
  direction = "out"
)
```

Arguments

- `graph`: A `tbl_graph` 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.
- `height`: The node variable holding the height of each node in the dendrogram. If `NULL` it will be calculated as the maximal distance to a leaf.
- `length`: An edge parameter giving the length of each edge. The node height will be calculated from the maximal length to the root node (ignored if `height` does not evaluate to `NULL`)
- `repel`: Should leafs repel each other relative to the height of their common ancestor. Will emphasize clusters
- `ratio`: The strength of repulsion if `repel = TRUE`. Higher values will give more defined clusters
- `direction`: The direction to the leaves. Defaults to 'out'

Value

A data.frame with the columns `x`, `y`, `circular`, `depth` and `leaf` as well as any information stored as node variables on the `tbl_graph`

Note

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

See Also

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`
layout_tbl_graph_eigen

Place nodes according to their eigenvalues

Description
This layout is based on the idea of spectral layouts where node coordinates are calculated directly by decomposing a matrix representation of the graph and extracting the eigenvectors.

Usage
layout_tbl_graph_eigen(
  graph,
  type = "laplacian",
  eigenvector = "smallest",
  circular = FALSE
)

Arguments
- graph: A tbl_graph object
- type: The type of matrix to extract the eigenvectors from. Either 'laplacian' or 'adjacency'
- eigenvector: The eigenvector to use for coordinates. Either 'smallest' or 'largest'
- circular: ignored

Value
A data.frame with the columns x, y, circular as well as any information stored as node variables in the tbl_graph object.

Author(s)
The underlying algorithm is implemented in the graphlayouts package by David Schoch

See Also
Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(), layout_tbl_graph_circletree(), layout_tbl_graph_dendrogram(), layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_igraph(), layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(), layout_tbl_graph_pmmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted()
Description

This layout is a bit unusual in that it shows nodes as horizontal line ranges end edges as evenly spaced vertical spans connecting the nodes. As with the matrix layout the strength comes from better scalability but its use require some experience recognising the patterns that different connectivity features gives rise to. As with matrix layouts the ordering of nodes have huge power over the look of the plot. The node_rank_fabric() mimics the default ordering from the original BioFabric implementation, but other ranking algorithms from tidygraph can be used with the sort.by argument as well. Fabric layouts tend to become quite wide as the graph grows which is something that should be handled with care - e.g. by only zooming in on a specific region.

Usage

```r
layout_tbl_graph_fabric(
  graph,
  circular = FALSE,
  sort.by = NULL,
  shadow.edges = FALSE
)
```

Arguments

- **graph**: An tbl_graph object
- **circular**: Ignored
- **sort.by**: An expression providing the sorting of the nodes. If NULL the nodes will be ordered by their index in the graph.
- **shadow.edges**: Should shadow edges be shown.

Value

A data.frame with the columns `x`, `xmin`, `xmax`, `y`, `circular` as well as any information stored as node variables in the tbl_graph object. Further, the edges of the graph will gain a `edge_x` variable giving the horizontal position of the edge as well as a `shadow_edge` variable denoting whether the edge is a shadow edge added by the layout.

References

BioFabric website: http://www.biofabric.org

See Also

Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(), layout_tbl_graph_circlepack(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_igraph(), layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(), layout_tbl_graph_pmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted.

layout_tbl_graph_focus

*Place nodes in circles based on distance to a specific node*

Description

This layout constrains node placement to a radius relative to its distance to a given node. It then uses stress majorisation to find an optimal node distribution according to this constraint.

Usage

```r
layout_tbl_graph_focus(
  graph,
  focus,
  weights = NULL,
  niter = 500,
  tolerance = 1e-04,
  circular = TRUE
)
```

Arguments

- **graph**: a tbl_graph object
- **focus**: An expression evaluating to a selected node. Can either be a single integer or a logical vector with a single TRUE element.
- **weights**: An expression evaluated on the edge data to provide edge weights for the layout. Currently ignored for the sparse version
- **niter**: number of iterations during stress optimization
- **tolerance**: stopping criterion for stress optimization
- **circular**: ignored

Value

A data.frame with the columns `x`, `y`, `circular`, `distance` as well as any information stored as node variables in the tbl_graph object.

Author(s)

The underlying algorithm is implemented in the graphlayouts package by David Schoch.
**References**


**See Also**

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`

---

`layout_tbl_graph_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**

```r
layout_tbl_graph_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 tbl_graph 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, center.size, split, axis, section, angle, circular as well as any information stored as node variables in the tbl_graph object.

References


http://www.hiveplot.net
See Also

Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(), layout_tbl_graph_circlepack(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(), layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph igraph(), layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(), layout_tbl_graph_pmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted.

layout_tbl_graph_igraph

Use igraph layout algorithms for layout_tbl_graph

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

Arguments

graph A tbl_graph object.
algorithm The type of layout algorithm to apply. See Details or igraph::layout_() for links to the layouts supplied by igraph.
circular Logical. Should the layout be transformed to a circular representation. Defaults to FALSE. Only applicable to algorithm = 'tree' and algorithm = 'sugiyama'.
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.
... Arguments passed on to the respective layout functions
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 `igraph::as_tree()`
sugiyama Designed for directed acyclic graphs (that is, hierarchies where multiple parents are allowed) it minimizes the number of crossing edges. See `igraph::with_sugiyama()`

**Standard layouts**
bipartite Minimize edge-crossings in a simple two-row (or column) layout for bipartite graphs. See `igraph::as_bipartite()`
star Place one node in the center and the rest equidistantly around it. See `igraph::as_star()`
circle Place nodes in a circle in the order of their index. Consider using `layout_tbl_graph_linear()` with circular=TRUE for more control. See `igraph::in_circle()`
nicely Tries to pick an appropriate layout. See `igraph::nicely()` for a description of the simple decision tree it uses
dh Uses Davidson and Harels simulated annealing algorithm to place nodes. See `igraph::with_dh()`
gem Place nodes on the plane using the GEM force-directed layout algorithm. See `igraph::with_gem()`
graphopt Uses the Graphopt algorithm based on alternating attraction and repulsion to place nodes. See `igraph::with_graphopt()`
grid Place nodes on a rectangular grid. See `igraph::on_grid()`
mds Perform a multidimensional scaling of nodes using either the shortest path or a user supplied distance. See `igraph::with_mds()`
sphere Place nodes uniformly on a sphere - less relevant for 2D visualizations of networks. See `igraph::on_sphere()`
randomly Places nodes uniformly random. See `igraph::randomly()`
fr Places nodes according to the force-directed algorithm of Fruchterman and Reingold. See `igraph::with_fr()`
kk Uses the spring-based algorithm by Kamada and Kawai to place nodes. See `igraph::with_kk()`
drl Uses the force directed algorithm from the DrL toolbox to place nodes. See `igraph::with_drl()`
lgl Uses the algorithm from Large Graph Layout to place nodes. See `igraph::with_lgl()`

**Value**

A data.frame with the columns `x`, `y`, circular as well as any information stored as node variables in the tbl_graph object.

**Note**

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

Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(),
layout_tbl_graph_circlepack(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(),
layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_linear(),
layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(),
layout_tbl_graph_pmds(), layout_tbl_graph_stress(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted()

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

Arguments

graph An tbl_graph object
circular Logical. Should the layout be transformed to a circular representation. Defaults to FALSE.
sort.by The name of a node variable 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 node variables in the tbl_graph object.
**layout_tbl_graph_manual**

Manually specify a layout for `layout_tbl_graph`

---

### Description

This layout function lets you pass the node positions in manually. The supplied positions must match the order of the nodes in the tbl_graph.

### Usage

```r
layout_tbl_graph_manual(graph, x, y, circular)
```

### Arguments

- **graph**: An tbl_graph object
- **x, y**: Expressions with the x and y positions of the nodes
- **circular**: Ignored

### Value

A data.frame with the columns `x`, `y`, `circular` as well as any information stored as node variables in the tbl_graph.

### See Also

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`
layout_tbl_graph_matrix

*Place nodes on a diagonal*

**Description**

This layout puts all nodes on a diagonal, thus preparing the layout for use with `geom_edge_point()` resulting in a matrix layout. While matrix layouts excel in scalability, the interpretation of the visual is very dependent on the sorting of the nodes. Different sorting algorithms have been implemented in `tidygraph` and these can be used directly. Behrisch et al. (2016) have provided a nice overview of some of the different sorting algorithms and what insight they might bring, along with a rundown of different patterns to look out for.

**Usage**

`layout_tbl_graph_matrix(graph, circular = FALSE, sort.by = NULL)`

**Arguments**

- `graph` : An `tbl_graph` object
- `circular` : Ignored
- `sort.by` : An expression providing the sorting of the nodes. If `NULL` the nodes will be ordered by their index in the graph.

**Value**

A data.frame with the columns `x`, `y`, `circular` as well as any information stored as node variables in the `tbl_graph` object.

**References**


**See Also**

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`
**layout_tbl_graph_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_tbl_graph_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 reminiscent 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**

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

**Arguments**

- **graph**
  - An `tbl_graph` object

- **weight**
  - An optional node variable 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 node variable to use as height. If `NULL` all nodes will be given a height of 1.

- **sort.by**
  - The name of a node variable 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 node variables in the tbl_graph 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 node variables in the tbl_graph 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


See Also

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`

---

`layout_tbl_graph_pmds` *Place nodes based on a multidimensional scaling of a set of pivot nodes*

Description

This layout is similar to the ‘mds’ layout but uses only a subset of pivot nodes for the mds calculation, making it considerably faster and thus suited for large graphs

Usage

`layout_tbl_graph_pmds(graph, pivots, weights = NULL, circular = FALSE)`

Arguments

- `graph` A tbl_graph object
- `pivots` The number of pivot nodes
- `weights` An expression evaluated on the edge data to provide edge weights for the layout. Currently ignored for the sparse version
- `circular` ignored
**layout_tbl_graph_stress**

**Value**

A data.frame with the columns `x`, `y`, `circular` as well as any information stored as node variables in the tbl_graph object.

**Author(s)**

The underlying algorithm is implemented in the graphlayouts package by David Schoch

**References**


**See Also**

Other `layout_tbl_graph_*` functions: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()`, `layout_tbl_graph_unrooted()`

---

**layout_tbl_graph_stress**

*Place nodes using stress majorisation*

**Description**

This layout is related to the stress-minimization algorithm known as Kamada-Kawai (available as the `kk` layout), but uses another optimization strategy. It generally have better runtime, quality, and stability compared to the Kamada-Kawai layout and is thus generally preferred. The sparse version of the layout have better performance (especially on larger networks) at the expense of layout quality, but will generally outperform many other algorithms for large graphs in both runtime and quality (e.g. the `drl` layout from igraph).

**Usage**

```r
layout_tbl_graph_stress(
  graph,
  weights = NULL,
  niter = 500,
  tolerance = 1e-04,
  mds = TRUE,
  bbox = 50,
  circular = FALSE
)
```

```r
layout_tbl_graph_sparse_stress(
```
graph, pivots, weights = NULL, niter = 500, circular = FALSE)

Arguments

graph a tbl_graph object
weights An expression evaluated on the edge data to provide edge weights for the layout. Currently ignored for the sparse version
niter number of iterations during stress optimization
tolerance stopping criterion for stress optimization
mds should an MDS layout be used as initial layout (default: TRUE)
bbox constrain dimension of output. Only relevant to determine the placement of disconnected graphs.
circular ignored
pivots The number of pivot nodes.

Value

A data.frame with the columns x, y, circular as well as any information stored as node variables in the tbl_graph object.

Author(s)

The underlying algorithm is implemented in the graphlayouts package by David Schoch

References


See Also

Other layout_tbl_graph_*: layout_tbl_graph_auto(), layout_tbl_graph_backbone(), layout_tbl_graph_centrality(), layout_tbl_graph_circlepack(), layout_tbl_graph_dendrogram(), layout_tbl_graph_eigen(), layout_tbl_graph_fabric(), layout_tbl_graph_focus(), layout_tbl_graph_hive(), layout_tbl_graph_igraph(), layout_tbl_graph_linear(), layout_tbl_graph_manual(), layout_tbl_graph_matrix(), layout_tbl_graph_partition(), layout_tbl_graph_pmds(), layout_tbl_graph_treemap(), layout_tbl_graph_unrooted()
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

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

Arguments

- **graph**: A tbl_graph object
- **algorithm**: The name of the tiling algorithm to use. Defaults to 'split'
- **weight**: An optional node variable 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 node variables 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 node variables in the `tbl_graph` 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


See Also

Other layout_tbl_graph_*: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centrality()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_unrooted()`
**Description**

When drawing unrooted trees the standard dendrogram layout is a bad fit as it implicitly creates a visual root node. Instead it is possible to spread the leaves out on the plane without putting any special emphasis on a particular node using an unrooted layout. The standard algorithm is the equal angle algorithm, but it can struggle with optimising the leaf distribution for large trees with very uneven branch length. The equal daylight algorithm modifies the output of the equal angle algorithm to better disperse the leaves, at the cost of higher computational cost and the possibility of edge crossings for very large unbalanced trees. For standard sized trees the daylight algorithm is far superior and not too heavy so it is the default.

**Usage**

```r
layout_tbl_graph_unrooted(
  graph,
  daylight = TRUE,
  length = NULL,
  tolerance = 0.05,
  rotation_mod = 1,
  maxiter = 100,
  circular = FALSE
)
```

**Arguments**

- `graph` A tbl_graph object
- `daylight` Should equal-daylight adjustments be made
- `length` An expression evaluating to the branch length of each edge
- `tolerance` The threshold for mean angular adjustment before terminating the daylight adjustment
- `rotation_mod` A modifier for the angular adjustment of each branch. Set it below 1 to let the daylight adjustment progress more slowly
- `maxiter` The maximum number of iterations in the daylight adjustment
- `circular` ignored

**Value**

A data.frame with the columns `x`, `y`, `circular`, `leaf` as well as any information stored as node variables in the tbl_graph object.

**Note**

Unrooted is a layout intended for undirected trees, that is, graphs with no cycles. If the provided graph does not fit this format an attempt to convert it to such a format will be made.

**References**

See Also

Other `layout_tbl_graph_*`: `layout_tbl_graph_auto()`, `layout_tbl_graph_backbone()`, `layout_tbl_graph_centralities()`, `layout_tbl_graph_circlepack()`, `layout_tbl_graph_dendrogram()`, `layout_tbl_graph_eigen()`, `layout_tbl_graph_fabric()`, `layout_tbl_graph_focus()`, `layout_tbl_graph_hive()`, `layout_tbl_graph_igraph()`, `layout_tbl_graph_linear()`, `layout_tbl_graph_manual()`, `layout_tbl_graph_matrix()`, `layout_tbl_graph_partition()`, `layout_tbl_graph_pmds()`, `layout_tbl_graph_stress()`, `layout_tbl_graph_treemap()

---

### 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

```r
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

```r
require(tidygraph)
flareGraph <- tbl_graph(flare$vertices, flare$edges)

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))
```
Description

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

Usage

```r
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


Examples

```r
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')
```
Description

This set of scales defines new alpha scales for edge geoms equivalent to the ones already defined by ggplot2. See \texttt{ggplot2::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 \texttt{edge_alpha} in the call to the geom - just use \texttt{alpha}.

Usage

\begin{verbatim}
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_binned(..., range = c(0.1, 1))
scale_edge_alpha_manual(..., values, breaks = waiver(), na.value = NA)
scale_edge_alpha_identity(..., guide = "none")
\end{verbatim}

Arguments

\begin{verbatim}
... Other arguments passed on to \texttt{continuous_scale()}, \texttt{binned_scale()}, or \texttt{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. The values will be matched in order (usually alphabetical) with the limits of the scale, or with \texttt{breaks} if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don't match will be given \texttt{na.value}.
breaks One of:
  \begin{itemize}
    \item \texttt{NULL} for no breaks
    \item \texttt{waiver()} for the default breaks (the scale limits)
    \item A character vector of breaks
    \item A function that takes the limits as input and returns breaks as output
  \end{itemize}
na.value The aesthetic value to use for missing (NA) values
guide Guide to use for this scale. Defaults to "none".
\end{verbatim}

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",
  aesthetics = "edge_colour"
)

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

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

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

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

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

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

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

scale_edge_colour_manual(
  ..., 
  values, 
  aesthetics = "edge_colour",
  ...
scale_edge_colour

    breaks = waiver(),
    na.value = "grey50"
  )

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

scale_edge_colour_steps(
    ...,  
    low = "#132B43",
    high = "#56B1F7",
    space = "Lab",
    na.value = "grey50",
    guide = "edge_coloursteps",
    aesthetics = "edge_colour"
  )

scale_edge_colour_steps2(
    ...,  
    low = muted("red"),
    mid = "white",
    high = muted("blue"),
    midpoint = 0,
    space = "Lab",
    na.value = "grey50",
    guide = "edge_coloursteps",
    aesthetics = "edge_colour"
  )

scale_edge_colour_stepsn(
    ...,  
    colours,
    values = NULL,
    space = "Lab",
    na.value = "grey50",
    guide = "edge_coloursteps",
    aesthetics = "edge_colour",
    colors
  )
scale_edge_colour_fermenter(
    ..., 
    type = "seq", 
    palette = 1, 
    direction = -1, 
    na.value = "grey50", 
    guide = "edge_coloursteps", 
    aesthetics = "edge_colour"
)

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

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

code_edge_colour_binned(
    ..., 
    low = "#132B43", 
    high = "#56B1F7", 
    space = "Lab", 
    na.value = "grey50", 
    guide = "edge_coloursteps", 
    aesthetics = "edge_colour"
)

code_edge_color_hue(
    ..., 
    h = c(0, 360) + 15, 
    c = 100, 
    l = 65, 
    h.start = 0, 
    direction = 1,
scale_edge_colour

    na.value = "grey50",
    aesthetics = "edge_colour"
)

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

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

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

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

scale_edge_color_gradientn(
    ..., 

scale_edge_colour

colours,
values = NULL,
space = "Lab",
na.value = "grey50",
guide = "edge_colourbar",
aesthetics = "edge_colour",
colors
)
scale_edge_color_grey(
...
start = 0.2,
end = 0.8,
na.value = "red",
aesthetics = "edge_colour"
)
scale_edge_color_identity(..., guide = "none", aesthetics = "edge_colour")
scale_edge_color_manual(
...
values,
aesthetics = "edge_colour",
breaks = waiver(),
na.value = "grey50"
)
scale_edge_color_continuous(
...
low = "#132B43",
high = "#56B1F7",
space = "Lab",
na.value = "grey50",
guide = "edge_colourbar",
aesthetics = "edge_colour"
)
scale_edge_color_discrete(
...
h = c(0, 360) + 15,
c = 100,
l = 65,
h.start = 0,
direction = 1,
na.value = "grey50",
aesthetics = "edge_colour"
scale_edge_color_viridis(
  ..., 
  alpha = 1, 
  begin = 0, 
  end = 1, 
  direction = 1, 
  discrete = FALSE, 
  option = "D", 
  aesthetics = "edge_colour"
)

scale_edge_color_steps(
  ..., 
  low = "#132B43", 
  high = "#56B1F7", 
  space = "Lab", 
  na.value = "grey50", 
  guide = "edge_coloursteps", 
  aesthetics = "edge_colour"
)

scale_edge_color_steps2(
  ..., 
  low = muted("red"), 
  mid = "white", 
  high = muted("blue"), 
  midpoint = 0, 
  space = "Lab", 
  na.value = "grey50", 
  guide = "edge_coloursteps", 
  aesthetics = "edge_colour"
)

scale_edge_color_stepsn(
  ..., 
  colours, 
  values = NULL, 
  space = "Lab", 
  na.value = "grey50", 
  guide = "edge_coloursteps", 
  aesthetics = "edge_colour", 
  colors
)

scale_edge_color_fermenter(
  ..., 
  type = "seq", 
  palette = 1,
direction = -1,
    na.value = "grey50",
    guide = "edge_coloursteps",
    aesthetics = "edge_colour"
)

scale_edge_color_binned(
    ...,                  
    low = "#132B43",
    high = "#56B1F7",
    space = "Lab",
    na.value = "grey50",
    guide = "edge_coloursteps",
    aesthetics = "edge_colour"
)

Arguments

Arguments passed on to discrete_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 (e.g., scales::hue_pal()).

breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks (the scale limits)
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

limits One of:
  • NULL to use the default scale values
  • A character vector that defines possible values of the scale and their order
  • A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

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.

scale_name The name of the scale that should be used for error messages associated with this scale.

name The name of the scale. Used as the 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
scale_edge_colour

- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See `plotmath` for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.

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

expand For position scales, a 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 `expansion()` 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.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

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

aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

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. The list of available palettes can found in the Palettes section.

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 convenience 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
grey value at low end of palette

end
grey value at high end of palette

breaks
One of:
- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

alpha
The alpha transparency, a number in [0,1], see argument alpha in hsv.

begin
The (corrected) hue in [0,1] at which the color map begins.

discrete
Generate a discrete palette? (default: FALSE - generate continuous palette).

option
A character string indicating the color map option to use. Eight options are available:
- "magma" (or "A")
- "inferno" (or "B")
- "plasma" (or "C")
- "viridis" (or "D")
- "cividis" (or "E")
- "rocket" (or "F")
- "mako" (or "G")
- "turbo" (or "H")

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()

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.
scale\_edge\_fill

Usage

scale\_edge\_fill\_hue(  
  ...,  
  h = c(0, 360) + 15,  
  c = 100,  
  l = 65,  
  h.start = 0,  
  direction = 1,  
  na.value = "grey50",  
  aesthetics = "edge\_fill"
)

scale\_edge\_fill\_brewer(  
  ...,  
  type = "seq",  
  palette = 1,  
  direction = 1,  
  aesthetics = "edge\_fill"
)

scale\_edge\_fill\_distiller(  
  ...,  
  type = "seq",  
  palette = 1,  
  direction = -1,  
  values = NULL,  
  space = "Lab",  
  na.value = "grey50",  
  guide = "edge\_colourbar",  
  aesthetics = "edge\_fill"
)

scale\_edge\_fill\_gradient(  
  ...,  
  low = "#132B43",  
  high = "#56B1F7",  
  space = "Lab",  
  na.value = "grey50",  
  guide = "edge\_colourbar",  
  aesthetics = "edge\_fill"
)

scale\_edge\_fill\_gradient2(  
  ...,  
  low = muted("red"),  
  mid = "white",  
  high = muted("blue"),  
  midpoint = 0,
scale_edge_fill

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

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

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

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

scale_edge_fill_manual(
  ..., 
  values,
  aesthetics = "edge_fill",
  breaks = waiver(),
  na.value = "grey50"
)

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

scale_edge_fill_steps(
  ..., 
  ...
)
scale_edge_fill

low = "#132B43",
high = "#56B1F7",
na.value = "grey50",
guide = "edge_coloursteps",
aesthetics = "edge_fill"
)
scale_edge_fill_steps2(
   ...
   low = muted("red"),
   mid = "white",
   high = muted("blue"),
   midpoint = 0,
   na.value = "grey50",
   guide = "edge_coloursteps",
aesthetics = "edge_fill"
)
scale_edge_fill_stepsn(
   ...
   colours = NULL,
   space = "Lab",
   na.value = "grey50",
   guide = "edge_coloursteps",
aesthetics = "edge_fill"
)
scale_edge_fill_fermenter(
   ...
   type = "seq",
   palette = 1,
   direction = -1,
   na.value = "grey50",
   guide = "edge_coloursteps",
aesthetics = "edge_fill"
)
scale_edge_fill_continuous(
   ...
   low = "#132B43",
   high = "#56B1F7",
   na.value = "grey50",
   guide = "edge_colourbar"
aesthetics = "edge_fill"

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

scale_edge_fill_binned(
  ..., 
  low = "#132B43", 
  high = "#56B1F7", 
  space = "Lab", 
  na.value = "grey50", 
  guide = "edge_coloursteps", 
  aesthetics = "edge_fill"
)

**Arguments**

Arguments passed on to `discrete_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 (e.g., `scales::hue_pal()`).

breaks One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

limits One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

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`. 
scale_edge_fill

scale_name The name of the scale that should be used for error messages associated with this scale.

name The name of the scale. Used as the 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)
  • An expression vector (must be the same length as breaks). See ?plotmath for details.
  • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

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

expand For position scales, a 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 expansion() 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.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

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

aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

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. The list of available palettes can be found in the Palettes section.

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 convenience 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  grey value at low end of palette

derg  grey value at high end of palette

breaks  One of:
  • NULL for no breaks
  • waiver() for the default breaks (the scale limits)
  • A character vector of breaks
  • A function that takes the limits as input and returns breaks as output

alpha  The alpha transparency, a number in [0,1], see argument alpha in hsv.

begin  The (corrected) hue in [0,1] at which the color map begins.

discrete  Generate a discrete palette? (default: FALSE - generate continuous palette).

option  A character string indicating the color map option to use. Eight options are available:
  • "magma" (or "A")
  • "inferno" (or "B")
  • "plasma" (or "C")
  • "viridis" (or "D")
  • "cividis" (or "E")
  • "rocket" (or "F")
  • "mako" (or "G")
  • "turbo" (or "H")

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 `ggplot2::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

```r
scale_edge_linetype(..., na.value = "blank")
scale_edge_linetype_continuous(...)
scale_edge_linetype_discrete(..., na.value = "blank")
scale_edge_linetype_binned(..., na.value = "blank")
scale_edge_linetype_manual(..., values, breaks = waiver(), na.value = "blank")
scale_edge_linetype_identity(..., guide = "none")
```

Arguments

...  
Arguments passed on to `discrete_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 (e.g., `scales::hue_pal()`).

`breaks` One of:
- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`limits` One of:
- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`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.
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 that should be used for error messages associated with this scale.

**name** The name of the scale. Used as the 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)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.

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

**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. The values will be matched in order (usually alphabetical) with the limits of the scale, or with `breaks` if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don’t match will be given `na.value`.

**breaks** One of:
- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

**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 `ggplot2::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**

```r
scale_edge_shape(..., solid = TRUE)
scale_edge_shape_discrete(..., solid = TRUE)
scale_edge_shape_continuous(...)
scale_edge_shape_binned(..., solid = TRUE)
scale_edge_shape_manual(..., values, breaks = waiver(), na.value = NA)
scale_edge_shape_identity(..., guide = "none")
```

**Arguments**

- `...` Arguments passed on to `discrete_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 (e.g., `scales::hue_pal()`).
- `breaks` One of:
  - `NULL` for no breaks
  - `waiver()` for the default breaks (the scale limits)
  - A character vector of breaks
  - A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.
- `limits` One of:
  - `NULL` to use the default scale values
  - A character vector that defines possible values of the scale and their order
  - A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.
- `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.
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.translate** If `na.translate = TRUE`, what aesthetic value should the missing values 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 that should be used for error messages associated with this scale.

- **name** The name of the scale. Used as the 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`)
  - An expression vector (must be the same length as `breaks`). See `?plotmath` for details.
  - A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.

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

- **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. The values will be matched in order (usually alphabetical) with the limits of the scale, or with `breaks` if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don’t match will be given `na.value`.

- **breaks** One of:
  - `NULL` for no breaks
  - `waiver()` for the default breaks (the scale limits)
  - A character vector of breaks
  - A function that takes the limits as input and returns breaks as output

- **na.value** The aesthetic value to use for missing (NA) values

- **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 `ggplot2::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**

```r
scale_edge_size_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)

scale_edge_radius(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)

scale_edge_size(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)

scale_edge_size_discrete(...)

scale_edge_size_binned(
  name = waiver(),
  breaks = waiver(),
  ...)
labels = waiver(),
limits = NULL,
range = c(1, 6),
n.breaks = NULL,
nice.breaks = TRUE,
trans = "identity",
guide = "bins"
)

scale_edge_size_area(..., max_size = 6)

scale_edge_size_binned_area(..., max_size = 6)

scale_edge_size_manual(..., values, breaks = waiver(), na.value = NA)

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

Arguments

name
The name of the scale. Used as the 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 (e.g., a function returned by scales::extended_breaks()). Also accepts rlang lambda function notation.

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)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

limits
One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will remove data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).
`scale_edge_size`

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

- **trans**: For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

  A transformation object bundles together a transform, its 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 information.

- **...**: Arguments passed on to `continuous_scale`

- **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. Also accepts rlang `lambda` function notation.

- **oob**: One of:
  - Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang `lambda` function notation.
  - The default (`scales::censor()`) replaces out of bounds values with NA.
  - `scales::squish()` for squishing out of bounds values into range.
  - `scales::squish_infinite()` for squishing infinite values into range.

- **na.value**: Missing values will be replaced with this value.

- **expand**: For position scales, a 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 `expansion()` 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.

- **position**: For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

- **super**: The super class to use for the constructed scale

- **n.breaks**: An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use NULL to use the default number of breaks given by the transformation.

- **nice.breaks**: Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.
max_size  Size of largest points.

values  a set of aesthetic values to map data values to. The values will be matched
        in order (usually alphabetical) with the limits of the scale, or with breaks if
        provided. If this is a named vector, then the values will be matched based on the
        names instead. Data values that don’t match will be given na.value.

na.value  The aesthetic value to use for missing (NA) values

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_continuous

scale_edge_width

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 mimics the use of size in
ggplot2::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

```r
scale_edge_width_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)

scale_edge_width(
```
name = waiver(),
breaks = waiver(),
labels = waiver(),
limits = NULL,
range = c(1, 6),
trans = "identity",
guide = "legend"
)

scale_edge_width_discrete(...)

scale_edge_width_binned(
  name = waiver(),
breaks = waiver(),
labels = waiver(),
limits = NULL,
range = c(1, 6),
n.breaks = NULL,
n.nice.breaks = TRUE,
trans = "identity",
guide = "bins"
)

scale_edge_width_manual(..., values, breaks = waiver(), na.value = NA)

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

**Arguments**

**name**

The name of the scale. Used as the 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 (e.g., a function returned by `scales::extended_breaks()`). Also accepts rlang `lambda` function notation.

**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)
- An expression vector (must be the same length as breaks). See `?plotmath` for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang `lambda` function notation.
limits

One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will remove data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

range

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

trans

For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its 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 information.

... Arguments passed on to `continuous_scale`

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. Also accepts rlang lambda function notation.

oob One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with NA.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

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

expand For position scales, a 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 `expansion()` 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.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.
The super class to use for the constructed scale

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

values a set of aesthetic values to map data values to. The values will be matched in order (usually alphabetical) with the limits of the scale, or with breaks if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don’t match will be given na.value.

na.value The aesthetic value to use for missing (NA) values

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_continuous(name = waiver(), breaks = waiver(), labels = waiver(), limits = NULL, range = c(1, 6), trans = "identity", guide = "legend")

scale_label_size(name = waiver(),

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(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  trans = "identity",
  guide = "legend"
)

scale_label_size(
  name = waiver(),
breaks = waiver(),
labels = waiver(),
limits = NULL,
range = c(1, 6),
trans = "identity",
guide = "legend"
)

scale_label_size_discrete(...)

scale_label_size_binned(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  range = c(1, 6),
  n.breaks = NULL,
  nice.breaks = TRUE,
  trans = "identity",
  guide = "bins"
)

scale_label_size_manual(..., values, breaks = waiver(), na.value = NA)

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

Arguments

name

The name of the scale. Used as the 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 (e.g., a function returned by scales::extended_breaks()). Also accepts rlang lambda function notation.

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)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.
limits  One of:
  • NULL to use the default scale range
  • A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
  • A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will remove data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).

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

trans  For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".
A transformation object bundles together a transform, its 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 information.

...  Arguments passed on to continuous_scale

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. Also accepts rlang lambda function notation.

oob  One of:
  • Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
  • The default (scales::censor()) replaces out of bounds values with NA.
  • scales::squish() for squishing out of bounds values into range.
  • scales::squish_infinite() for squishing infinite values into range.

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

expand  For position scales, a 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 expansion() 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.

position  For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.
super  The super class to use for the constructed scale

n.breaks  An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks  Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

values  a set of aesthetic values to map data values to. The values will be matched in order (usually alphabetical) with the limits of the scale, or with breaks if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don’t match will be given na.value.

na.value  The aesthetic value to use for missing (NA) values

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()

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. ggplot 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). The th_no_axes() helper is provided to modify an existing theme so that grid and axes are removed.
theme_graph

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)

th_no_axes()

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

unset_graph_style()
Arguments

- `base_size`, `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`, `caption_colour`
  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

```r
library(tidygraph)
graph <- as_tbl_graph(highschool)

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

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

Index

* datasets
  flare, 9
  highschool, 92
  whigs, 150
* extractors
  get_con, 82
  get_edges, 83
  get_nodes, 84
* geom_conn *
  geom_conn_bundle, 13
* geom_edge *
  geom_edge_arc, 16
  geom_edge_bend, 21
  geom_edge_density, 26
  geom_edge_diagonal, 28
  geom_edge_elbow, 32
  geom_edgeFan, 37
  geom_edge_hive, 42
  geom_edge_link, 47
  geom_edge_loop, 51
  geom_edge_parallel, 55
  geom_edge_point, 60
  geom_edge_span, 62
  geom_edge_tile, 67
* geom_node *
  geom_node_arc_bar, 69
  geom_node_circle, 70
  geom_node_point, 72
  geom_node_range, 74
  geom_node_text, 75
  geom_node_tile, 78
  geom_node_voronoi, 80
* ggraph-facets
  facet_edges, 4
  facet_graph, 6
  facet_nodes, 8
* graph
  ggraph, 85
  * layout_tbl_graph *
    layout_tbl_graph_auto, 93
    layout_tbl_graph_backbone, 94
    layout_tbl_graph_centrality, 95
    layout_tbl_graph_circlepack, 96
    layout_tbl_graph_dendrogram, 97
    layout_tbl_graph_eigen, 99
    layout_tbl_graph_fabric, 100
    layout_tbl_graph_focus, 101
    layout_tbl_graph_hive, 102
    layout_tbl_graph_igraph, 104
    layout_tbl_graph_linear, 106
    layout_tbl_graph_manual, 107
    layout_tbl_graph_matrix, 108
    layout_tbl_graph_partition, 109
    layout_tbl_graph_pmds, 110
    layout_tbl_graph_stress, 111
    layout_tbl_graph_treemap, 113
    layout_tbl_graph_unrooted, 114
* layout
  ggraph, 85
* network
  ggraph, 85
* scale_edge *
  scale_edge_alpha, 118
  scale_edge_colour, 119
  scale_edge_fill, 128
  scale_edge_linetype, 135
  scale_edge_shape, 137
  scale_edge_size, 139
  scale_edge_width, 142
  scale_label_size, 145
* visualisation
  ggraph, 85
  aes(), 12
  autograph, 3
  binned_scale(), 118
geom_conn_bundle

geom_conn_bundle2

flare

facet_nodes

facet_graph

create_layout(ggraph)

create_layout()

INDEX

circle (geometry), 10
continuous_scale, 141, 144, 147
continuous_scale()

coord_cartesian()

create_layout() 140, 144, 147

create_layout (ggplot2), 85
create_layout(), 98, 105

discrete_scale, 126, 132, 135, 137

edge_angle (node_angle), 116
element_text(), 87, 88, 90, 92

ellipses (geometry), 10
expansion(), 127, 133, 141, 144, 147

fabric, 62, 74

facet_edges, 4, 7, 9

facet_graph, 6, 9

facet_nodes, 6, 7, 9

flare, 9

fortify(), 12, 69, 71, 72, 74, 76, 78, 80

geom_axis_hive, 11

gem_conn_bundle, 13

gem_conn_bundle0 (geom_conn_bundle), 13
gem_conn_bundle2 (geom_conn_bundle), 13

gem_edge_arc, 16, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_arc0 (geom_edge_arc), 16

gem_edge_arc2 (geom_edge_arc), 16

gem_edge_bend, 21, 22, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_bend0 (geom_edge_bend), 21

gem_edge_bend2 (geom_edge_bend), 21

gem_edge_density, 21, 25, 26, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

geom_edge_diagonal, 21, 25, 27, 28, 37, 42, 46, 51, 55, 60, 62, 66, 68

geom_edge_diagonal0

(geom_edge_diagonal), 28

geom_edge_diagonal2

(geom_edge_diagonal), 28

geom_edge_elbow, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

geom_edge_elbow()

geom_edge_elbow0 (geom_edge_elbow), 32

geom_edge_elbow2 (geom_edge_elbow), 32

geom_edgeFan, 21, 25, 27, 32, 37, 46, 51, 55, 60, 62, 66, 68

gem_edge_fan0 (geom_edge_fan), 37

gem_edge_fan2 (geom_edge_fan), 37

gem_edge_hive, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_hive0 (geom_edge_hive), 42

gem_edge_hive2 (geom_edge_hive), 42

geom_edge_link, 21, 25, 27, 32, 37, 42, 46, 47, 51, 55, 60, 62, 66, 68

gem_edge_link0 (geom_edge_link), 47

gem_edge_link2 (geom_edge_link), 47

gem_edge_loop, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_loop0 (geom_edge_loop), 51

gem_edge_parallel, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_parallel0

(geom_edge_parallel), 55

gem_edge_parallel2

(geom_edge_parallel), 55

geom_edge_point, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_edge_point()

, 86, 108

geom_edge_span, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

geom_edge_span0 (geom_edge_span), 62

gem_edge_span2 (geom_edge_span), 62

gem_edge_tile, 21, 25, 27, 32, 37, 42, 46, 51, 55, 60, 62, 66, 68

gem_node_arc_bar, 69, 72, 73, 75, 77, 79, 82

gem_node_circle, 70, 76, 79, 79, 82

gem_node_label (geom_node_text), 75

gem_node_point, 70, 72, 75, 77, 79, 82

gem_node_range, 70, 72, 73, 74, 77, 79, 82

gem_node_text, 70, 72, 75, 77, 79, 82

gem_node_tile, 70, 72, 73, 75, 77, 78, 82

gem_node_voronoi, 70, 72, 73, 75, 77, 79, 80

geometry, 10

geometry()

, 20, 25, 31, 36, 41, 46, 49, 54, 59, 66

get_con, 82, 84, 85

get_con()

, 14, 87

get_edges, 83, 83, 85

get_edges()

, 18, 20, 23, 25, 26, 29, 31, 34, 36, 39, 41, 44, 46, 48, 49, 52, 54, 57, 59, 61, 64, 66, 67, 84, 87

get_nodes, 83, 84, 84

get_geom::geom_arc_bar(), 69

ggforce::geom_circle(), 70
ggforce::geom_voronoi_tile(), 80
ggplot(), 12, 69, 71, 72, 74, 76, 78, 80
ggplot2::aes(), 14, 18, 23, 26, 29, 34, 39,
44, 48, 52, 57, 61, 63, 67, 69, 71, 72,
74, 76, 78, 80
ggplot2::aes(), 14, 18, 23, 26, 29, 34, 39,
44, 48, 52, 57, 61, 63, 67, 69, 71, 72,
74, 76, 78, 80
ggplot2::coord_fixed(), 70, 81
ggplot2::facet_grid(), 6
ggplot2::facet_wrap(), 4, 8
ggplot2::geom_label(), 75
ggplot2::geom_line(), 142
ggplot2::geom_point(), 64, 72
ggplot2::geom_segment(), 84
ggplot2::geom_text(), 75
ggplot2::ggplot(), 85
ggplot2::guide_colourbar, 87, 89
ggplot2::guide_colourbar(), 87
ggplot2::guide_coloursteps(), 89
ggplot2::margin(), 11
ggplot2::scale_alpha(), 118
ggplot2::scale_linetype(), 135
ggplot2::scale_shape(), 137
ggplot2::scale_size(), 139
ggraph, 85
ggraph(), 93
grDevices::plotmath(), 18, 23, 30, 34, 39,
44, 49, 53, 57, 64
grid::arrow(), 14, 18, 23, 29, 34, 39, 44, 48,
53, 57, 64
grid::gpar(), 11
grid::unit(), 18, 23, 30, 35, 39, 40, 44, 49,
53, 57, 64, 88, 90, 92
guide_edge_colorbar
  (guide_edge_colourbar), 87
guide_edge_colorsteps
  (guide_edge_coloursteps), 89
guide_edge_colourbar, 87
guide_edge_coloursteps, 89
guide_edge_direction(), 91
guides(), 127, 133, 136, 138, 141, 144, 147
highschool, 92
hsv, 128, 134
igraph::as_bipartite(), 105
igraph::as_star(), 105
igraph::as_tree(), 105
igraph::in_circle(), 105
igraph::layout(), 104
igraph::layout_as_tree(), 97
igraph::nicely(), 105
igraph::on_grid(), 105
igraph::on_sphere(), 105
igraph::plot.igraph(), 37
igraph::randomly(), 105
igraph::with_dh(), 105
igraph::with_drl(), 105
igraph::with_fr(), 105
igraph::with_gem(), 105
igraph::with_graphopt(), 105
igraph::with_kk(), 105
igraph::with_lgl(), 105
igraph::with_mds(), 105
igraph::with_sugiyama(), 105
is.geometry (geometry), 10
label_parsed(), 5, 7, 9
label_rect (geometry), 10
label_value(), 5, 7, 9
labeller(), 5, 7, 9
labs(), 87, 89, 91
lambda, 126, 127, 132, 133, 135–138, 140,
141, 143, 144, 146, 147
layer(), 12, 15, 19, 23, 27, 30, 35, 40, 44, 49,
53, 58, 61, 64, 68, 69, 71, 73, 74, 77,
79, 81
layout_ggraph (ggraph), 85
layout_tbl_graph (ggraph), 85
layout_tbl_graph_auto, 93, 95–99, 101,
102, 104, 106–108, 110–112, 114,
116
layout_tbl_graph_auto(), 86
layout_tbl_graph_backbone, 94, 94, 96–99,
101, 102, 104, 106–108, 110–112,
114, 116
layout_tbl_graph_centrality, 94, 95, 95,
97–99, 101, 102, 104, 106–108,
110–112, 114, 116
layout_tbl_graph_circlepack, 94–96, 96,
98, 99, 101, 102, 104, 106–108,
110–112, 114, 116, 117
layout_tbl_graph_circlepack.six, 86
layout_tbl_graph_dendrogram, 94–97, 97,
99, 101, 102, 104, 106–108,
110–112, 114, 116
layout_tbl_graph_dendrogram(), 33, 86
INDEX

layout_tbl_graph_manual(), 86
layout_tbl_graph_igraph(), 33, 86
layout_tbl_graph_linear(), 16, 86, 105
layout_tbl_graph_manual(), 86
layout_tbl_graph_matrix(), 60, 67, 86
layout_tbl_graph_partition(), 33, 86
layout_tbl_graph_sparse_stress
(layout_tbl_graph_stress), 111
layout_tbl_graph_tree_map(), 86, 109
node_angle, 116
node_rank_fabric
(layout_tbl_graph_fabric), 100
pack_circles, 117
rectangle (geometry), 10
rescale(), 127, 133
scale_edge_alpha, 118, 128, 134, 136, 138, 142, 145, 148
scale_edge_alpha_binned
(scale_edge_alpha), 118
scale_edge_alpha_continuous
(scale_edge_alpha), 118
scale_edge_alpha_discrete
(scale_edge_alpha), 118
scale_edge_alpha_identity
(scale_edge_alpha), 118
scale_edge_alpha_manual
(scale_edge_alpha), 118
scale_edge_color_binned
(scale_edge_colour), 119
scale_edge_color_brewer
(scale_edge_colour), 119
scale_edge_color_continuous
(scale_edge_colour), 119
scale_edge_color_discrete
(scale_edge_colour), 119
scale_edge_color_distiller
(scale_edge_colour), 119
scale_edge_color_fermenter
(scale_edge_colour), 119
scale_edge_color_gradient
(scale_edge_colour), 119
scale_edge_color_gradient2
(scale_edge_colour), 119
scale_edge_color_gradientn
(scale_edge_colour), 119
scale_edge_color_grey
(scale_edge_colour), 119
scale_edge_color_hue
(scale_edge_colour), 119
scale_edge_color_identity
(scale_edge_colour), 119
scale_edge_color_manual
(scale_edge_colour), 119
scale_edge_color_steps
(scale_edge_colour), 119
scale_edge_color_steps2
(scale_edge_colour), 119
scale_edge_color_stepsn
(scale_edge_colour), 119
scale_edge_color_viridis  
  (scale_edge_colour), 119  
scale_edge_colour, 119, 119, 134, 136, 138, 142, 145, 148  
scale_edge_colour_binned  
  (scale_edge_colour), 119  
scale_edge_colour_brewer  
  (scale_edge_colour), 119  
scale_edge_colour_continuous  
  (scale_edge_colour), 119  
scale_edge_colour_discrete  
  (scale_edge_colour), 119  
scale_edge_colour_distiller  
  (scale_edge_colour), 119  
scale_edge_colour_fermenter  
  (scale_edge_colour), 119  
scale_edge_colour_gradient  
  (scale_edge_colour), 119  
scale_edge_colour_gradient2  
  (scale_edge_colour), 119  
scale_edge_colour_gradientn  
  (scale_edge_colour), 119  
scale_edge_colour_grey  
  (scale_edge_colour), 119  
scale_edge_colour_hue  
  (scale_edge_colour), 119  
scale_edge_colour_identity  
  (scale_edge_colour), 119  
scale_edge_colour_manual  
  (scale_edge_colour), 119  
scale_edge_colour_steps  
  (scale_edge_colour), 119  
scale_edge_colour_steps2  
  (scale_edge_colour), 119  
scale_edge_colour_stepsn  
  (scale_edge_colour), 119  
scale_edge_colour_viridis  
  (scale_edge_colour), 119  
scale_edge_fill, 119, 128, 128, 136, 138, 142, 145, 148  
scale_edge_fill_binned  
  (scale_edge_fill), 128  
scale_edge_fill_brewer  
  (scale_edge_fill), 128  
scale_edge_fill_continuous  
  (scale_edge_fill), 128  
scale_edge_fill_discrete  
  (scale_edge_fill), 128  
scale_edge_fill_distiller  
  (scale_edge_fill), 128  
scale_edge_fill_fermenter  
  (scale_edge_fill), 128  
scale_edge_fill_gradient  
  (scale_edge_fill), 128  
scale_edge_fill_gradient2  
  (scale_edge_fill), 128  
scale_edge_fill_gradientn  
  (scale_edge_fill), 128  
scale_edge_fill_grey  
  (scale_edge_fill), 128  
scale_edge_fill_hue  
  (scale_edge_fill), 128  
scale_edge_fill_identity  
  (scale_edge_fill), 128  
scale_edge_fill_manual  
  (scale_edge_fill), 128  
scale_edge_fill_steps  
  (scale_edge_fill), 128  
scale_edge_fill_steps2  
  (scale_edge_fill), 128  
scale_edge_fill_stepsn  
  (scale_edge_fill), 128  
scale_edge_fill_viridis  
  (scale_edge_fill), 128  
scale_edge_linetype, 119, 128, 134, 135, 138, 142, 145, 148  
scale_edge_linetype_binned  
  (scale_edge_linetype), 135  
scale_edge_linetype_continuous  
  (scale_edge_linetype), 135  
scale_edge_linetype_discrete  
  (scale_edge_linetype), 135  
scale_edge_linetype_identity  
  (scale_edge_linetype), 135  
scale_edge_linetype_manual  
  (scale_edge_linetype), 135  
scale_edge_radius  
  (scale_edge_size), 139  
scale_edge_shape, 119, 128, 134, 136, 137, 142, 145, 148  
scale_edge_shape_binned  
  (scale_edge_shape), 137  
scale_edge_shape_continuous  
  (scale_edge_shape), 137  
scale_edge_shape_discrete  
  (scale_edge_shape), 137  
scale_edge_shape_identity
INDEX

(scale_edge_shape, 137)
scale_edge_shape_manual
  (scale_edge_shape, 137)
scale_edge_size, 119, 128, 134, 136, 138, 139, 145, 148
  scale_edge_size_area (scale_edge_size), 139
  scale_edge_size_binned
    (scale_edge_size), 139
  scale_edge_size_binned_area
    (scale_edge_size), 139
  scale_edge_size_continuous
    (scale_edge_size), 139
  scale_edge_size_discrete
    (scale_edge_size), 139
  scale_edge_size_identity
    (scale_edge_size), 139
  scale_edge_size_manual
    (scale_edge_size), 139
  scale_edge_width, 119, 128, 134, 136, 138, 142, 142, 148
  scale_edge_width()
    (scale_edge_width), 142
  scale_edge_width_binned
    (scale_edge_width), 142
  scale_edge_width_continuous
    (scale_edge_width), 142
  scale_edge_width_discrete
    (scale_edge_width), 142
  scale_edge_width_identity
    (scale_edge_width), 142
  scale_edge_width_manual
    (scale_edge_width), 142
  scale_label_size, 119, 128, 134, 136, 138, 142, 145, 145
  scale_label_size_binned
    (scale_label_size), 145
  scale_label_size_continuous
    (scale_label_size), 145
  scale_label_size_discrete
    (scale_label_size), 145
  scale_label_size_identity
    (scale_label_size), 145
  scale_label_size_manual
    (scale_label_size), 145
  scales::boxcox_trans()
    (scale_edge_shape), 137
  scales::censor()
    (scale_edge_shape), 137
  scales::extended_breaks()
    (scale_edge_shape), 137
  scales::hue_pal()
    (scale_edge_shape), 137
  scales::squish()
    (scale_edge_shape), 137
  scales::squish_infinite()
    (scale_edge_shape), 137
  scales::trans_new()
    (scale_edge_shape), 137
  set_graph_style (theme_graph), 148
  square (geometry), 10
  th_foreground (theme_graph), 148
  th_no_axes (theme_graph), 148
  theme(), 87, 88, 90, 92
  theme_graph, 148
  transformation object, 140, 143, 146
  unset_graph_style (theme_graph), 148
  vars(), 5, 8
  waiver(), 87, 89, 91
  whigs, 150