

Package: ggdag (via r-universe)

September 19, 2026

Title Analyze and Create Elegant Directed Acyclic Graphs

Version 0.2.13.9000

Description Tidy, analyze, and plot directed acyclic graphs (DAGs).
'ggdag' is built on top of 'dagitty', an R package that uses the 'DAGitty' web tool (<<https://dagitty.net/>>) for creating and analyzing DAGs. 'ggdag' makes it easy to tidy and plot 'dagitty' objects using 'ggplot2' and 'ggraph', as well as common analytic and graphical functions, such as determining adjustment sets and node relationships.

License MIT + file LICENSE

URL <https://github.com/r-causal/ggdag>,
<https://r-causal.github.io/ggdag/>

BugReports <https://github.com/r-causal/ggdag/issues>

Depends R (>= 4.1.0)

Imports cli (>= 3.0.0), dagitty, dplyr (>= 1.1.1), forcats, ggarrow (>= 0.1.1.9000), ggplot2 (>= 3.5.0), ggraph (>= 2.0.0), ggrepel (>= 0.9.7), igraph, lifecycle (>= 0.2.0), magrittr, pillar, purrr, rlang, stringr, tibble, tidygraph

Suggests covr, knitr, rmarkdown, spelling, testthat (>= 3.0.0), vdiffr (>= 1.0.2), withr

VignetteBuilder knitr

Remotes teunbrand/ggarrow

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs

libfontconfig1-dev libfreetype6-dev libglpk-dev libicu-dev libxml2-dev libssl-dev libnode-dev

Repository <https://r-causal.r-universe.dev>

Date/Publication 2026-09-01 13:44:24 UTC

RemoteUrl <https://github.com/r-causal/ggdag>

RemoteRef HEAD

RemoteSha b107665fd9167179654f7772315c2744d2d0445a

Contents

activate_collider_paths	4
Adjust for variables	5
aes_dag	7
as.data.frame.tidy_dagitty	8
as.tbl.tidy_dagitty	9
as_tbl_graph	9
as_tidy_dagitty	10
Assess d-separation between variables	12
Assess familial relationships between variables	17
Canonicalize DAGs	23
Colliders	25
coordinates	27
Covariate Adjustment Sets	28
curve_edge	30
curved	31
dag	32
DAG Edges	33
DAG Labels	37
dag_saturate	38
dagify	40
dplyr	41
draw_key_dag_collider	43
draw_key_dag_combined	44
draw_key_dag_edge	44
draw_key_dag_point	45
edge_backdoor	45
Equivalent DAGs and Classes	47
Exogenous Variables	50
expand_plot	52
fortify	53
geom_dag	53
geom_dag_arrow	56
geom_dag_collider_edges	61
geom_dag_edges	63
geom_dag_label	66
geom_dag_text	69
ggdag	72
ggdag_classic	74

ggdag_defaults	75
ggplot.tidy_dagitty	77
ggrepel functions	77
Instrumental Variables	84
is.tidy_dagitty	86
is_confounder	86
is_dag_properties	87
is_node_properties	88
is_node_relationships	89
Nodes	90
Pathways	93
print.tidy_dagitty	97
pull_dag	97
query_adjustment_sets	99
query_ancestors	100
query_children	101
query_colliders	102
query_conditional_independence	103
query_dconnected	105
query_descendants	106
query_dseparated	107
query_exogenous	108
query_instrumental	108
query_markov_blanket	109
query_parents	110
query_paths	111
query_status	112
Quick Plots for Common DAGs	113
remove_axes	123
scale_adjusted	123
set_curve_edges	124
simulate_data	125
tbl_df.tidy_dagitty	126
Test if Variable Is Collider	126
theme_dag_blank	127
theme_dag_grey	128
tidy_dagitty	128
time_ordered_coords	130
Variable Status	132

`activate_collider_paths`*Activate paths opened by stratifying on a collider*

Description

Stratifying on colliders can open biasing pathways between variables. `activate_collider_paths` activates any such pathways given a variable or set of variables to adjust for and adds them to the `tidy_dagitty`. A pathway is added for a pair of variables upstream of an adjusted collider only when the adjustment opens a path between them that is closed without it, so variables joined only by a path that the adjustment leaves as it found it are not connected. Openness is judged under the whole of `adjust_for`, so adjusting for a collider and for a variable that blocks the path it opens leaves the pair unconnected.

Usage

```
activate_collider_paths(.tidy_dag, adjust_for, ...)
```

Arguments

<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>adjust_for</code>	a character vector, the variable(s) to adjust for.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code> . These are only used when <code>.tidy_dag</code> is not already a <code>tidy_dagitty</code> .

Value

a `tidy_dagitty` with additional rows for collider-activated pathways

See Also

[control_for\(\)](#), [ggdag_adjust\(\)](#), [geom_dag_collider_edges\(\)](#)

Examples

```
dag <- dagify(m ~ x + y, x ~ y)

collided_dag <- activate_collider_paths(dag, adjust_for = "m")
collided_dag
```

Adjust for variables *Adjust for variables and activate any biasing paths that result*

Description

Adjust for variables and activate any biasing paths that result

Usage

```
control_for(.tdy_dag, var, as_factor = TRUE, activate_colliders = TRUE, ...)
```

```
adjust_for(.tdy_dag, var, as_factor = TRUE, activate_colliders = TRUE, ...)
```

```
ggdag_adjust(
  .tdy_dag,
  var = NULL,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = draw_key_dag_point,
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated(),
  collider_lines = TRUE
)
```

Arguments

`.tdy_dag` A tidy_dagitty or dagitty object

<code>var</code>	the variable(s) to adjust for. This can be a character vector of variable names or a list of the form <code>list(c(...))</code> .
<code>as_factor</code>	Logical. Should the column be a factor?
<code>activate_colliders</code>	logical. Include colliders activated by adjustment?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
<code>edge_engine</code>	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using ggarrow geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see ggdag_options_set()).
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .

label	The bare name of a column to use for labels. If use_labels = TRUE, the default is to use label.
node	Deprecated.
stylized	Deprecated.
collider_lines	logical. Should the plot show paths activated by adjusting for a collider? These paths are drawn as dashed ggraph curves whatever edge_engine is in use: they mark an association rather than an edge of the DAG, so they stay visibly apart from the arrows the engine draws.

Value

a tidy_dagitty with a adjusted column for adjusted variables, as well as any biasing paths that arise, or a ggplot

Edge layers of the composite plotters

The plotters that color or fade edges by an analysis column build their edge layers themselves, and which layers they build is settled from the DAG they are called with: a DAG with no bidirected edge is given no bidirected edge layer. Replacing the data of the returned plot afterwards, with ggplot2's %+%, does not bring a layer back, so a plot built for one DAG is not a template for another.

Examples

```
dag <- dagify(m ~ a + b, x ~ a, y ~ b)

control_for(dag, var = "m")
ggdag_adjust(dag, var = "m")
```

aes_dag

Define Aesthetics for Directed Acyclic Graphs (DAGs)

Description

aes_dag() is a wrapper around aes() that specifies x, y, xend, and yend, which are required for most DAG visualizations. It merges any additional aesthetics, e.g. color or shape, with the default aesthetic mappings.

Usage

```
aes_dag(...)
```

Arguments

... Additional aesthetic mappings passed as arguments. These can include any aes-
thetic supported by ggplot2 (e.g., color, size, shape).

Value

A ggplot2 aesthetic mapping object that includes both the default DAG aesthetics and any user-specified aesthetics.

Examples

```
library(ggplot2)
confounder_triangle() |>
  dag_adjustment_sets() |>
  ggplot(aes_dag(color = .data$adjusted)) +
  geom_dag() +
  facet_wrap(~set)
```

```
as.data.frame.tidy_dagitty
```

Convert a tidy_dagitty object to data.frame

Description

Convert a tidy_dagitty object to data.frame

Usage

```
## S3 method for class 'tidy_dagitty'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

<code>x</code>	an object of class tidy_dagitty
<code>row.names</code>	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
<code>optional</code>	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional)
<code>...</code>	optional arguments passed to as.data.frame()

as.tbl.tidy_dagitty *Convert a tidy_dagitty object to tbl*

Description

Convert a tidy_dagitty object to tbl

Usage

```
as.tbl.tidy_dagitty(x, row.names = NULL, optional = FALSE, ...)
```

```
## S3 method for class 'tidy_dagitty'
as_tibble(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	an object of class tidy_dagitty
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional)
...	optional arguments passed to dplyr::as_tibble()

as_tbl_graph *Convert DAGS to tidygraph*

Description

A thin wrapper to convert tidy_dagitty and dagitty objects to tbl_graph, which can then be used to work in tidygraph and ggraph directly. See [tidygraph::as_tbl_graph\(\)](#).

Usage

```
## S3 method for class 'tidy_dagitty'
as_tbl_graph(x, directed = TRUE, ...)
```

```
## S3 method for class 'dagitty'
as_tbl_graph(x, directed = TRUE, ...)
```

Arguments

x	an object of class tidy_dagitty or dagitty
directed	logical. Should the constructed graph be directed? Default is TRUE
...	currently unused

Value

a `tbl_graph`

Examples

```
library(ggraph)
library(tidygraph)
butterfly_bias() |>
  as_tbl_graph() |>
  ggraph() +
  geom_edge_diagonal() +
  geom_node_point()
```

as_tidy_dagitty

Convert objects into tidy_dagitty objects

Description

An alternative API and specification to [tidy_dagitty\(\)](#), `as_tidy_dagitty()` allows you to create `tidy_dagitty` objects from data frames and lists. There is also a method for `dagitty` objects, which is a thin wrapper for [tidy_dagitty\(\)](#). To create a DAG from a list, each element of the list should be a character vector, and the order of the elements should be the time order in which they appear in the DAG, e.g. element 1 occurs at time point 1. To create a DAG from a data frame, it must contain `name` and `to` columns, representing the nodes and any edges leading from the nodes. If there are `x`, `y`, `xend`, and `yend` columns, they will be used as coordinates. Otherwise, `layout` will be used. See [tidy_dagitty](#) for more information about layouts. Additionally, you can specify status (one of `exposure`, `outcome`, or `latent`) by including a `status` column. Any other columns in the data set will also be joined to the `tidy_dagitty` data.

Usage

```
as_tidy_dagitty(x, ...)

## S3 method for class 'dagitty'
as_tidy_dagitty(x, seed = NULL, layout = ggdag_option("layout", "nicely"), ...)

## S3 method for class 'data.frame'
as_tidy_dagitty(
  x,
  exposure = NULL,
  outcome = NULL,
  latent = NULL,
  labels = NULL,
  coords = NULL,
  seed = NULL,
  layout = ggdag_option("layout", "nicely"),
```

```

    saturate = FALSE,
    ...
  )

## S3 method for class 'list'
as_tidy_dagitty(
  x,
  exposure = NULL,
  outcome = NULL,
  latent = NULL,
  labels = NULL,
  coords = NULL,
  seed = NULL,
  layout = "time_ordered",
  ...
)

```

Arguments

x	An object to convert into a tidy_dagitty. Currently supports dagitty and data.frame objects.
...	optional arguments passed to ggraph::create_layout()
seed	a numeric seed for reproducible layout generation
layout	a layout available in ggraph. See ggraph::create_layout() for details. Alternatively, "time_ordered" will use time_ordered_coords() to algorithmically sort the graph by time. You can also pass the result of time_ordered_coords() directly: either the function returned when called with no arguments, or the coordinate tibble returned when called with arguments.
exposure	a character vector for the exposure (must be a variable name in the DAG)
outcome	a character vector for the outcome (must be a variable name in the DAG)
latent	a character vector for any latent variables (must be a variable name in the DAG)
labels	a named character vector, labels for variables in the DAG
coords	coordinates for the DAG nodes. Can be a named list or a data.frame with columns x, y, and name
saturate	Logical. Saturate the DAG such that there is an edge going from every point in the future from a given node? Setting this to TRUE will potentially lead to more edges than present in x.

Value

a tidy_dagitty object

See Also

[tidy_dagitty\(\)](#), [pull_dag\(\)](#)

Examples

```
data.frame(name = c("c", "c", "x"), to = c("x", "y", "y")) |>
  as_tidy_dagitty()

time_points <- list(c("a", "b", "c"), "d", c("e", "f", "g"), "z")

time_points |>
  # create a saturated, time-ordered DAG
  as_tidy_dagitty() |>
  # remove the edge from `c` to `f`
  dag_prune(c("c" = "f"))
```

Assess d-separation between variables

D-relationship between variables

Description

D-separation is a key concept in causal structural models. Variables are d-separated if there are no open paths between them. The `node_d*`() functions label variables as d-connected or d-separated. The `ggdag_d*`() functions plot the results. The `*_dconnected()`, `*_dseparated()`, and `*_drelationship()` functions essentially produce the same output and are just different ways of thinking about the relationship. See `dagitty::dseparated()` for details.

Usage

```
node_dconnected(
  .tdy_dag,
  from = NULL,
  to = NULL,
  controlling_for = NULL,
  as_factor = TRUE,
  ...
)
```

```
node_dseparated(
  .tdy_dag,
  from = NULL,
  to = NULL,
  controlling_for = NULL,
  as_factor = TRUE,
  ...
)
```

```
node_drelationship(
  .tdy_dag,
```

```

    from = NULL,
    to = NULL,
    controlling_for = NULL,
    as_factor = TRUE,
    ...
)

ggdag_drelationship(
  .tdy_dag,
  from = NULL,
  to = NULL,
  controlling_for = NULL,
  ...,
  edge_type = ggdag_option("edge_type", "link_arc"),
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = draw_key_dag_point,
  label = NULL,
  text = NULL,
  node = deprecated(),
  stylized = deprecated(),
  collider_lines = TRUE
)

ggdag_dseparated(
  .tdy_dag,
  from = NULL,
  to = NULL,
  controlling_for = NULL,
  ...,
  edge_type = ggdag_option("edge_type", "link_arc"),
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),

```

```

label_size = ggdag_option("label_size", text_size),
text_col = ggdag_option("text_col", "white"),
label_col = ggdag_option("label_col", "black"),
edge_width = ggdag_option("edge_width", 0.6),
edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
arrow_length = ggdag_option("arrow_length", 5),
use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
unified_legend = TRUE,
key_glyph = draw_key_dag_point,
label = NULL,
text = NULL,
node = deprecated(),
stylized = deprecated(),
collider_lines = TRUE
)

ggdag_dconnected(
  .tdy_dag,
  from = NULL,
  to = NULL,
  controlling_for = NULL,
  ...,
  edge_type = ggdag_option("edge_type", "link_arc"),
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = draw_key_dag_point,
  label = NULL,
  text = NULL,
  node = deprecated(),

```

```

    stylized = deprecated(),
    collider_lines = TRUE
  )

```

Arguments

<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>from</code>	A character vector with starting node name(s), or NULL. If NULL, checks DAG for exposure variable.
<code>to</code>	A character vector with ending node name(s), or NULL. If NULL, checks DAG for outcome variable.
<code>controlling_for</code>	A set of variables to control for. This can be a character vector of variable names, a list of the form <code>list(c(...))</code> , or NULL. When NULL, no control is applied. Default is NULL.
<code>as_factor</code>	Logical. Should the column be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*</code> () function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.

<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.
<code>collider_lines</code>	Logical. Should paths opened by conditioning on colliders be shown? These paths are drawn as dashed <code>ggraph</code> curves whatever <code>edge_engine</code> is in use: they mark an association rather than an edge of the DAG, so they stay visibly apart from the arrows the engine draws.

Value

a `tidy_dagitty` with a `d_relationship` column for variable D relationship or a `ggplot`

Examples

```
library(ggplot2)
dag <- dagify(m ~ x + y)
dag |> ggdag_drelationship("x", "y")
dag |> ggdag_drelationship("x", "y", controlling_for = "m")

dag |>
  node_dseparated("x", "y") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend,
             col = d_relationship)) +
  geom_dag_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  theme_dag()

dag |>
  node_dconnected("x", "y", controlling_for = "m") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted,
             col = d_relationship)) +
  geom_dag_edges() +
  geom_dag_collider_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  theme_dag() +
  scale_adjusted(include_color = FALSE)

dagify(m ~ x + y, m_jr ~ m) |>
  tidy_dagitty(layout = "nicely") |>
  node_dconnected("x", "y", controlling_for = "m_jr") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted,
```

```

      col = d_relationship)) +
geom_dag_edges() +
geom_dag_collider_edges() +
geom_dag_node() +
geom_dag_text(col = "white") +
theme_dag() +
scale_adjusted(include_color = FALSE)

```

Assess familial relationships between variables

Familial relationships between variables

Description

Parents and children are those nodes that either directly cause or are caused by the variable, respectively. Ancestors and descendants are those nodes that are on the path to or descend from the variable. The `node_*()` functions label variables depending on their relationship. The `ggdag_*()` functions plot the results. See [dagitty::children](#) for details.

Usage

```

node_children(.tdy_dag, .var, as_factor = TRUE)

node_parents(.tdy_dag, .var, as_factor = TRUE)

node_ancestors(.tdy_dag, .var, as_factor = TRUE)

node_descendants(.tdy_dag, .var, as_factor = TRUE)

node_markov_blanket(.tdy_dag, .var, as_factor = TRUE)

node_adjacent(.tdy_dag, .var, as_factor = TRUE)

ggdag_children(
  .tdy_dag,
  .var,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),

```

```

use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
unified_legend = TRUE,
text = NULL,
label = NULL,
node = deprecated(),
stylized = deprecated()
)

```

```

ggdag_parents(
  .tdy_dag,
  .var,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

```

ggdag_ancestors(
  .tdy_dag,
  .var,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),

```

```

    label_size = ggdag_option("label_size", text_size),
    text_col = ggdag_option("text_col", "white"),
    label_col = ggdag_option("label_col", "black"),
    edge_width = ggdag_option("edge_width", 0.6),
    edge_cap = ggdag_option("edge_cap", 8),
    arrow_length = ggdag_option("arrow_length", 5),
    use_edges = ggdag_option("use_edges", TRUE),
    use_nodes = ggdag_option("use_nodes", TRUE),
    use_stylized = ggdag_option("use_stylized", FALSE),
    use_text = ggdag_option("use_text", TRUE),
    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    unified_legend = TRUE,
    text = NULL,
    label = NULL,
    node = deprecated(),
    stylized = deprecated()
)

ggdag_descendants(
  .tdy_dag,
  .var,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

ggdag_markov_blanket(
  .tdy_dag,

```

```

    .var,
    ...,
    size = 1,
    edge_type = c("link_arc", "link", "arc", "diagonal"),
    node_size = ggdag_option("node_size", 16),
    text_size = ggdag_option("text_size", 3.88),
    label_size = ggdag_option("label_size", text_size),
    text_col = ggdag_option("text_col", "white"),
    label_col = ggdag_option("label_col", "black"),
    edge_width = ggdag_option("edge_width", 0.6),
    edge_cap = ggdag_option("edge_cap", 8),
    arrow_length = ggdag_option("arrow_length", 5),
    use_edges = ggdag_option("use_edges", TRUE),
    use_nodes = ggdag_option("use_nodes", TRUE),
    use_stylized = ggdag_option("use_stylized", FALSE),
    use_text = ggdag_option("use_text", TRUE),
    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    unified_legend = TRUE,
    text = NULL,
    label = NULL,
    node = deprecated(),
    stylized = deprecated()
)

ggdag_adjacent(
  .tdy_dag,
  .var,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,

```

```

    node = deprecated(),
    stylized = deprecated()
  )

```

Arguments

<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>.var</code>	a character vector, the variable(s) to be assessed (must be in the DAG). When more than one variable is given, the relation is assessed against the set as a whole. <code>node_parents()</code> , <code>node_children()</code> , and <code>node_adjacent()</code> label a queried variable that is also a relative of another queried variable with the relation rather than as a queried variable. <code>node_ancestors()</code> , <code>node_descendants()</code> , and <code>node_markov_blanket()</code> instead drop every queried variable from the relation set, so a queried variable always carries the queried-set label and only variables outside the queried set carry the relation label.
<code>as_factor</code>	Logical. Should the column be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.

text	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use name.
label	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use label.
node	Deprecated.
stylized	Deprecated.

Value

a `tidy_dagitty` with an column related to the given relationship for variable D relationship or a `ggplot`

Examples

```
library(ggplot2)
dag <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ ~w2
)

ggdag_children(dag, "w1")

dag |>
  node_children("w1") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, color = children)) +
  geom_dag_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  geom_dag_label_repel(aes(label = children, fill = children),
    col = "white", show.legend = FALSE) +
  theme_dag() +
  scale_adjusted(include_color = FALSE) +
  scale_color_hue(breaks = c("parent", "child"))

ggdag_parents(dag, "y")

ggdag_ancestors(dag, "x")

ggdag_descendants(dag, "w1")

dag |>
  node_parents("y") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, color = parent)) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text(col = "white") +
  geom_dag_label_repel(aes(label = parent, fill = parent),
    col = "white", show.legend = FALSE) +
```

```
theme_dag() +
scale_adjusted(include_color = FALSE) +
scale_color_hue(breaks = c("parent", "child"))
```

Canonicalize DAGs

Canonicalize a DAG

Description

Takes an input graph with bidirected edges and replaces every bidirected edge $x \leftrightarrow y$ with a sub-structure $x \leftarrow L \rightarrow y$, where L is a latent variable. See [dagitty::canonicalize\(\)](#) for details. Undirected edges are not currently supported in ggdag.

Usage

```
node_canonical(.dag, ...)

ggdag_canonical(
  .tdy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)
```

Arguments

<code>.dag, .tidy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code> , which lays out the canonical DAG. <code>use_existing_coords</code> is not among them: the latent variables that the canonical form introduces have no coordinates to reuse.
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.

Value

a tidy_dagitty that includes L or a ggplot

Examples

```
dag <- dagify(y ~ x + z, x ~ ~z)

ggdag(dag)

node_canonical(dag)
ggdag_canonical(dag)
```

Colliders

Find colliders

Description

Detects any colliders given a DAG. `node_collider` tags colliders and `ggdag_collider` plots all colliders.

Usage

```
node_collider(.dag, as_factor = TRUE, ...)

ggdag_collider(
  .tidy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,
```

```

    node = deprecated(),
    stylized = deprecated()
  )

```

Arguments

<code>.dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>as_factor</code>	Logical. Should the column be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.

Value

a tidy_dagitty with a colliders column for colliders or a ggplot

Examples

```
dag <- dagify(m ~ x + y, y ~ x)

node_collider(dag)
ggdag_collider(dag)
```

coordinates	<i>Manipulate DAG coordinates</i>
-------------	-----------------------------------

Description

Manipulate DAG coordinates

Usage

```
coords2df(coord_list)

coords2list(coord_df)
```

Arguments

coord_list	a named list of coordinates
coord_df	a data.frame with columns x, y, and name

Value

either a list or a data.frame with DAG node coordinates

Examples

```
library(dagitty)
coords <- list(
  x = c(A = 1, B = 2, D = 3, C = 3, F = 3, E = 4, G = 5, H = 5, I = 5),
  y = c(A = 0, B = 0, D = 1, C = 0, F = -1, E = 0, G = 1, H = 0, I = -1)
)
coord_df <- coords2df(coords)
coords2list(coord_df)

x <- dagitty("dag{
  G <-> H <-> I <-> G
  D <- B -> C -> I <- F <- B <- A
  H <- E <- C -> G <- D
}")
coordinates(x) <- coords2list(coord_df)
```

Covariate Adjustment Sets

Covariate Adjustment Sets

Description

See `dagitty::adjustmentSets()` for details.

Usage

```
dag_adjustment_sets(.tdy_dag, exposure = NULL, outcome = NULL, ...)
```

```
ggdag_adjustment_set(
  .tdy_dag,
  exposure = NULL,
  outcome = NULL,
  ...,
  shadow = TRUE,
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = draw_key_dag_point,
  label = NULL,
  text = NULL,
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  node = deprecated(),
  stylized = deprecated(),
  expand_x = expansion(c(0.25, 0.25)),
  expand_y = expansion(c(0.2, 0.2))
)
```

Arguments

`.tdy_dag` A `tidy_dagitty` or `dagitty` object

exposure	A character vector, the exposure variable. Default is NULL, in which case it will be determined from the DAG.
outcome	A character vector, the outcome variable. Default is NULL, in which case it will be determined from the DAG.
...	additional arguments to <code>adjustmentSets</code>
shadow	logical. Show paths blocked by adjustment?
size	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
node_size	The size of the nodes.
text_size	The size of the text.
label_size	The size of the labels.
text_col	The color of the text.
label_col	The color of the labels.
edge_width	The width of the edges.
edge_cap	The size of edge caps (the distance between the arrowheads and the node borders).
arrow_length	The length of arrows on edges.
use_edges	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
use_nodes	A logical value. Include <code>geom_dag_point()</code> ?
use_stylized	A logical value. Include <code>geom_dag_node()</code> ?
use_text	A logical value. Include <code>geom_dag_text()</code> ?
use_labels	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
label_geom	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
unified_legend	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
key_glyph	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
label	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
text	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
edge_engine	The engine used to draw edges. Either <code>"ggraph"</code> (default) or <code>"ggarrow"</code> . When <code>"ggarrow"</code> , edges are drawn using <code>ggarrow</code> geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see <code>ggdag_options_set()</code>).

node	Deprecated.
stylized	Deprecated.
expand_x, expand_y	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 <code>ggplot2::expansion()</code> to generate the values for the <code>expand</code> argument.

Value

a `tidy_dagitty` with an `adjusted` column and `set` column, indicating adjustment status and DAG ID, respectively, for the adjustment sets or a `ggplot`

Edge layers of the composite plotters

The plotters that color or fade edges by an analysis column build their edge layers themselves, and which layers they build is settled from the DAG they are called with: a DAG with no bidirected edge is given no bidirected edge layer. Replacing the data of the returned plot afterwards, with `ggplot2`'s `%>%`, does not bring a layer back, so a plot built for one DAG is not a template for another.

Examples

```
dag <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ ~w2,
  exposure = "x",
  outcome = "y"
)

tidy_dagitty(dag) |> dag_adjustment_sets()

ggdag_adjustment_set(dag)

ggdag_adjustment_set(
  dagitty::randomDAG(10, 0.5),
  exposure = "x3",
  outcome = "x5"
)
```

curve_edge

Add or update curvature for a single edge

Description

`curve_edge()` sets the curvature for a single edge on a `dagitty` or `tidy_dagitty` object. Use `set_curve_edges()` to set multiple edges at once.

Usage

```
curve_edge(.dag, from, to, curvature = 0.3)
```

Arguments

.dag	A dagitty or tidy_dagitty object.
from	Character. The name of the source node.
to	Character. The name of the target node.
curvature	Numeric. The curvature value for the edge.

Value

The modified .dag object with updated curvature.

Curvature sign convention

Per-edge curvature is drawn by the **ggarrow** edge engine only. Set it with `edge_engine = "ggarrow"` on `geom_dag()` and the `ggdag_*()` quick plots, or globally with `ggdag_options_set(edge_engine = "ggarrow")`; the default **ggraph** engine draws every edge with the curvature of its own edge type and warns when it is handed a per-edge value it cannot draw.

`ggarrow` follows the `grid::curveGrob()` convention, so positive curvature curves *below* (to the right of) a left-to-right edge and negative curvature curves *above* it. `ggdag` passes the value through untouched.

A bidirected edge has no direction of its own, so `ggdag` draws it from the endpoint `dagitty` stores first. Naming its endpoints the other way round, as in `curve_edge(dag, "y", "x", 0.5)` for an edge stored as `x <-> y`, curves the edge to the same side of the page: the sign is flipped to match the direction the edge is drawn in.

Examples

```
dag <- dagify(y ~ x + m, m ~ x)
dag <- curve_edge(dag, from = "m", to = "y", curvature = 0.5)
```

curved

Mark an edge as curved in dagify formulas

Description

Use `curved()` inside `dagify()` formulas to specify per-edge curvature. This function should only be used inside `dagify()` formulas; calling it directly will result in an error, similar to `dplyr::n()`.

Usage

```
curved(var, curvature = 0.3)
```

Arguments

<code>var</code>	A variable name (unquoted) representing the parent node.
<code>curvature</code>	A numeric curvature value. Positive values curve edges in one direction, negative in the other. Default is 0.3.

Value

This function is not intended to be called directly. It is detected in the formula AST by `dagify()`.

Curvature sign convention

Per-edge curvature is drawn by the **ggarrow** edge engine only. Set it with `edge_engine = "ggarrow"` on `geom_dag()` and the `ggdag_*()` quick plots, or globally with `ggdag_options_set(edge_engine = "ggarrow")`; the default **ggraph** engine draws every edge with the curvature of its own edge type and warns when it is handed a per-edge value it cannot draw.

`ggarrow` follows the `grid::curveGrob()` convention, so positive curvature curves *below* (to the right of) a left-to-right edge and negative curvature curves *above* it. `ggdag` passes the value through untouched.

A bidirected edge has no direction of its own, so `ggdag` draws it from the endpoint `dagitty` stores first. Naming its endpoints the other way round, as in `curve_edge(dag, "y", "x", 0.5)` for an edge stored as `x <-> y`, curves the edge to the same side of the page: the sign is flipped to match the direction the edge is drawn in.

Examples

```
# Curve the edge from m to y
dagify(
  y ~ x + curved(m, 0.5),
  m ~ x
)
```

`dag`

Create a dagitty DAG

Description

A convenience wrapper for `dagitty::dagitty()`.

Usage

```
dag(...)
```

Arguments

`...` a character vector in the style of `dagitty`. See `dagitty::dagitty` for details.

Value

a dagitty

Examples

```
dag("{x m} -> y")
```

DAG Edges

Directed DAG edges

Description

Directed DAG edges

Usage

```
geom_dag_edges_link(
  mapping = NULL,
  data = NULL,
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

```
geom_dag_edges_arc(
  mapping = NULL,
  data = NULL,
  curvature = 0.5,
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  fold = 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,
    ...
)

geom_dag_edges_diagonal(
  mapping = NULL,
  data = NULL,
  position = "identity",
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  curvature = 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,
  ...
)

geom_dag_edges_fan(
  mapping = NULL,
  data = NULL,
  position = "identity",
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  spread = 0.7,
  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,
    ...
  )

```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> ., and will be used as the layer data.
arrow	specification for arrow heads, as created by <code>arrow()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>TRUE</code> , the default, missing values are removed silently. A node with no outgoing edge has a missing edge end, so the edge layers drop those rows rather than warning about them. If <code>FALSE</code> , missing values are removed with a warning.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .
...	Other arguments passed to <code>ggraph::geom_edge_*</code>
curvature	The bend of the curve. 1 approximates a halfcircle while 0 will give a straight line. Negative number will change the direction of the curve. Only used if <code>layout_circular = FALSE</code> .
fold	Logical. Should arcs appear on the same side of the nodes despite different directions. Default to <code>FALSE</code> .
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 <code>NA</code> it will use the colour of the edge.
label_alpha	The opacity of the edge label. If <code>NA</code> it will use the opacity of the edge.
label_parse	If <code>TRUE</code> , 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 <code>grid::unit()</code> giving a fixed vertical shift to add to the label in case of angle_calc is either 'along' or 'across'
label_push	A <code>grid::unit()</code> giving a fixed horizontal shift to add to the label in case of angle_calc is either 'along' or 'across'
spread	Deprecated. Use strength instead.

Aesthetics

geom_dag_edges_link, geom_dag_edges_arc, geom_dag_edges_diagonal, and geom_dag_edges_fan understand the following aesthetics. Bold aesthetics are required.

- x
- y
- **xend**
- **yend**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

geom_dag_edges_arc and geom_dag_edges_diagonal also require **circular**, but this is automatically set.

geom_dag_edges_fan requires **to** and **from**, but these are also automatically set.

Examples

```

library(ggplot2)
p <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  L ~ w1 + w2
) |>
  ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()

p + geom_dag_edges_link()
p + geom_dag_edges_arc()
p + geom_dag_edges_diagonal()
p + geom_dag_edges_fan()

```

DAG Labels

DAG labels

Description

Label or otherwise retrieve labels from objects of either class `tidy_dagitty` or `dagitty`

Usage

```

label(x) <- value

## S3 replacement method for class 'dagitty'
label(x) <- value

## S3 replacement method for class 'tidy_dagitty'
label(x) <- value

dag_label(.tdy_dag, labels = NULL)

label(.tdy_dag)

has_labels(.tdy_dag)

```

Arguments

<code>x</code>	an object of either class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>value</code>	a named character vector, where the names are nodes in the DAG (a name that matches no node is an error), or <code>NULL</code> to remove labels

<code>.tdy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>labels</code>	a named character vector, where the names are node names, or <code>NULL</code> to reuse the labels already attached to the DAG

Value

`label` returns the label attribute of `x`

Examples

```
labelled_dag <- dagify(y ~ z, x ~ z) |>
  tidy_dagitty() |>
  dag_label(labels = c("x" = "exposure", "y" = "outcome", "z" = "confounder"))

has_labels(labelled_dag)
```

<code>dag_saturate</code>	<i>Saturate or prune an existing DAG</i>
---------------------------	--

Description

`dag_saturate()` takes a tidy DAG object and, optionally using existing coordinates, saturates the DAG based on time ordering of the nodes. To create a saturated DAG from scratch, see [as_tidy_dagitty.list\(\)](#). `dag_prune()` takes an existing DAG and removes edges. This is most useful when used together with saturated DAG.

Usage

```
dag_saturate(
  .tdy_dag,
  use_existing_coords = FALSE,
  layout = "time_ordered",
  seed = NULL,
  ...
)

dag_prune(.tdy_dag, edges)
```

Arguments

<code>.tdy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>use_existing_coords</code>	Logical, indicating whether to use existing node coordinates.
<code>layout</code>	a layout available in <code>ggraph</code> . See ggraph::create_layout() for details. Alternatively, <code>"time_ordered"</code> will use <code>time_ordered_coords()</code> to algorithmically sort the graph by time. You can also pass the result of <code>time_ordered_coords()</code> directly: either the function returned when called with no arguments, or the coordinate tibble returned when called with arguments.

seed	a numeric seed for reproducible layout generation
...	optional arguments passed to <code>ggraph::create_layout()</code>
edges	The edges to remove, in either of two forms. A named character vector where the name is the starting node and the value is the end node, e.g. <code>c("x" = "y")</code> removes the edge going from x to y. Or a data frame with a name and a to column, which says the same thing, and an optional direction column of " <code>></code> ", " <code><</code> ", or " <code>--</code> ", which names one of the edges a pair of nodes holds.

Details

Bidirected edges carry no time-ordering information, so `dag_saturate()` assigns time order from the directed edges alone and then passes the input's bidirected edges through to the saturated DAG unchanged. A saturated model therefore never implies an independence that the input denies.

`dag_prune()` errors if edges is empty, and if it names an edge the DAG does not contain, including an edge written in the reverse direction. A node whose every edge is pruned is kept as an isolated node.

A pair of nodes can hold a directed edge and a bidirected edge at the same time, and endpoints alone name both of them. `dag_prune()` errors on such a pair rather than pruning both; name the direction as well, with the data frame form of edges, to prune one of them.

Value

A `tidy_dagitty` object

See Also

[as_tidy_dagitty.list\(\)](#)

Examples

```
# Example usage:
dag <- dagify(y ~ x, x ~ z)
saturated_dag <- dag_saturate(dag)

saturated_dag |>
  ggdag(edge_type = "arc")

saturated_dag |>
  dag_prune(c("x" = "y")) |>
  ggdag(edge_type = "arc")
```

dagify

*Create a dagitty DAG using R-like syntax***Description**

`dagify()` creates dagitty DAGs using a more R-like syntax. It currently accepts formulas in the usual R style, e.g. $y \sim x + z$, which gets translated to $y \leftarrow \{x \ z\}$, as well as using a double tilde ($\sim\sim$) to graph bidirected variables, e.g. $x1 \sim\sim x2$ is translated to $x1 \leftrightarrow x2$.

Usage

```
dagify(
  ...,
  exposure = NULL,
  outcome = NULL,
  latent = NULL,
  labels = NULL,
  coords = NULL
)
```

Arguments

<code>...</code>	formulas, which are converted to dagitty syntax
<code>exposure</code>	a character vector for the exposure (must be a variable name in the DAG)
<code>outcome</code>	a character vector for the outcome (must be a variable name in the DAG)
<code>latent</code>	a character vector for any latent variables (must be a variable name in the DAG)
<code>labels</code>	a named character vector, labels for variables in the DAG
<code>coords</code>	coordinates for the DAG nodes. Can be a named list or a <code>data.frame</code> with columns <code>x</code> , <code>y</code> , and <code>name</code>

Details

A single formula can mix the two: $y \sim x + \sim z$ gives $x \rightarrow y$ and $y \leftrightarrow z$. R's parser lets a unary $\sim$ take in the rest of the right-hand side, so every term after the tilde is bidirected: $y \sim x + \sim z + w$ gives $x \rightarrow y$, $y \leftrightarrow z$, and $y \leftrightarrow w$, and $y \sim \sim x + z$ leaves both x and z bidirected. Parentheses limit how far the tilde reaches, so $y \sim x + (\sim z) + w$ gives $x \rightarrow y$, $y \leftrightarrow z$, and $w \rightarrow y$.

A term that is a call contributes one edge for each variable it mentions, so both $y \sim f(x, z)$ and $y \sim x : z$ give $x \rightarrow y$ and $z \rightarrow y$.

Value

a dagitty DAG

See Also

[dag\(\)](#), [coords2df\(\)](#), [coords2list\(\)](#)

Examples

```

dagify(y ~ x + z, x ~ z)

coords <- list(
  x = c(A = 1, B = 2, D = 3, C = 3, J = 3, E = 4, G = 5, H = 5, I = 5),
  y = c(A = 0, B = 0, D = 1, C = 0, J = -1, E = 0, G = 1, H = 0, I = -1)
)

dag <- dagify(
  G ~ ~H,
  G ~ ~I,
  I ~ ~G,
  H ~ ~I,
  D ~ B,
  C ~ B,
  I ~ C + J,
  J ~ B,
  B ~ A,
  H ~ E,
  C ~ E + G,
  G ~ D,
  coords = coords
)

dagitty::is.dagitty(dag)

ggdag(dag)

dag2 <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ ~w2,
  exposure = "x",
  outcome = "y"
)

ggdag(dag2)

```

Description

Dplyr verb methods for tidy_dagitty objects.

Usage

```
## S3 method for class 'tidy_dagitty'
select(.data, ...)

## S3 method for class 'tidy_dagitty'
filter(.data, ...)

## S3 method for class 'tidy_dagitty'
rename(.data, ...)

## S3 method for class 'tidy_dagitty'
mutate(.data, ...)

## S3 method for class 'tidy_dagitty'
summarise(.data, ...)

## S3 method for class 'tidy_dagitty'
arrange(.data, ...)

## S3 method for class 'tidy_dagitty'
group_by(.data, ...)

## S3 method for class 'tidy_dagitty'
ungroup(x, ...)

## S3 method for class 'tidy_dagitty'
transmute(.data, ...)

## S3 method for class 'tidy_dagitty'
distinct(.data, ..., .keep_all = FALSE)

## S3 method for class 'tidy_dagitty'
full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'tidy_dagitty'
inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'tidy_dagitty'
left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'tidy_dagitty'
right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'tidy_dagitty'
anti_join(x, y, by = NULL, copy = FALSE, ...)

## S3 method for class 'tidy_dagitty'
semi_join(x, y, by = NULL, copy = FALSE, ...)
```

```
## S3 method for class 'tidy_dagitty'
slice(.data, ..., .dots = list())

select_.tidy_dagitty(.data, ..., .dots = list())

filter_.tidy_dagitty(.data, ..., .dots = list())

mutate_.tidy_dagitty(.data, ..., .dots = list())

summarise_.tidy_dagitty(.data, ..., .dots = list())

arrange_.tidy_dagitty(.data, ..., .dots = list())

slice_.tidy_dagitty(.data, ..., .dots = list())
```

Arguments

.data	data object of class tidy_dagitty
...	other arguments passed to the dplyr function
.dots, x, y, by, copy, suffix, .keep_all	see corresponding function in package dplyr

Examples

```
library(dplyr)
tidy_dagitty(m_bias()) |>
  group_by(name) |>
  summarize(n = n())
```

draw_key_dag_collider *Collider pattern legend key (many-to-one)*

Description

A custom legend key function that displays a collider pattern with two nodes pointing to one central node. This is particularly useful for visualizing collider relationships in DAGs.

Usage

```
draw_key_dag_collider(data, params, size)
```

Arguments

data	A data frame containing aesthetic information for the legend key
params	Additional parameters (not currently used)
size	Legend key size (not currently used)

Value

A grob object for the legend key

draw_key_dag_combined *Combined DAG legend key (horizontal node-edge-node)*

Description

A custom legend key function that displays a complete DAG representation showing two nodes connected by an arrow. This provides a unified legend entry for plots that show both nodes and edges.

Usage

```
draw_key_dag_combined(data, params, size)
```

Arguments

data	A data frame containing aesthetic information for the legend key
params	Additional parameters (not currently used)
size	Legend key size (not currently used)

Value

A grob object for the legend key

draw_key_dag_edge *DAG edge legend key (arrow only)*

Description

A custom legend key function that displays only an arrow (edge) without nodes. This is appropriate for edge-specific legends where nodes are not relevant.

Usage

```
draw_key_dag_edge(data, params, size)
```

Arguments

data	A data frame containing aesthetic information for the legend key
params	Additional parameters (not currently used)
size	Legend key size (not currently used)

Value

A grob object for the legend key

draw_key_dag_point	<i>DAG point legend key (25% size)</i>
--------------------	--

Description

A custom legend key function that draws points at 25% of their normal size with proportionally sized legend boxes. This creates much more compact legends while maintaining visual clarity.

Usage

```
draw_key_dag_point(data, params, size)
```

Arguments

data	A data frame containing aesthetic information for the legend key
params	Additional parameters (not currently used)
size	Legend key size (not currently used)

Value

A grob object for the legend key

edge_backdoor	<i>Classify DAG edges as backdoor or direct</i>
---------------	---

Description

`edge_backdoor()` identifies edges as being on backdoor paths or direct causal paths between an exposure and outcome. This function adds edge-level information to the tidy DAG object, classifying each edge based on the types of paths it appears on.

Usage

```
edge_backdoor(  
  .dag,  
  from = NULL,  
  to = NULL,  
  adjust_for = NULL,  
  open_only = TRUE,  
  ...  
)
```

Arguments

<code>.dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>from</code>	A character vector with starting node name(s), or NULL. If NULL, checks DAG for exposure variable.
<code>to</code>	A character vector with ending node name(s), or NULL. If NULL, checks DAG for outcome variable.
<code>adjust_for</code>	character vector, a set of variables to control for. Default is NULL.
<code>open_only</code>	logical. If TRUE (default), only considers open paths. If FALSE, includes information about closed paths as well.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>

Details

Edges are classified by examining the paths between exposure and outcome:

- Direct edges appear only on directed causal paths
- Backdoor edges appear only on backdoor paths, whose first edge points into the exposure
- Other edges appear only on paths that are neither causal nor backdoor, such as a path through a collider
- Both edges appear on more than one of those kinds of path

When `open_only = TRUE` (default), `path_type` will be NA for edges that are only part of closed paths.

Value

A `tidy_dagitty` object with additional columns:

- `path_type`: "direct", "backdoor", "other", or "both" classification for each edge
- `open`: logical indicating if the edge is part of an open path

Examples

```
# Create a DAG with both direct and backdoor paths
dag <- dagify(
  y ~ x + z,
  x ~ z,
  exposure = "x",
  outcome = "y"
)

# Classify edges
edge_backdoor(dag)

# Include closed paths
edge_backdoor(dag, open_only = FALSE)
```

Equivalent DAGs and Classes

Generating Equivalent Models

Description

Analyze the Markov equivalence class of an input DAG: the DAGs that encode the same conditional independencies as the input graph. See `dagitty::equivalentDAGs()` and `dagitty::equivalenceClass()` for details. `node_equivalent_dags()` returns a set of DAGs, while `node_equivalent_class()` tags reversible edges. `ggdag_equivalent_dags()` plots all equivalent DAGs, while `ggdag_equivalent_class()` plots all reversible edges as undirected.

Usage

```
node_equivalent_dags(
  .dag,
  n = 100,
  layout = ggdag_option("layout", "nicely"),
  ...
)

ggdag_equivalent_dags(
  .tdy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = NULL,
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
```

```

)

node_equivalent_class(.dag, layout = ggdag_option("layout", "nicely"), ...)

ggdag_equivalent_class(
  .tidy_dag,
  ...,
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  key_glyph = NULL,
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

Arguments

<code>.dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>n</code>	maximal number of returned graphs.
<code>layout</code>	a layout available in <code>ggraph</code> . See ggraph::create_layout() for details. Alternatively, <code>"time_ordered"</code> will use <code>time_ordered_coords()</code> to algorithmically sort the graph by time. You can also pass the result of <code>time_ordered_coords()</code> directly: either the function returned when called with no arguments, or the coordinate tibble returned when called with arguments.
<code>...</code>	optional arguments passed to <code>ggraph::create_layout()</code>
<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of <code>"link_arc"</code> , <code>"link"</code> , <code>"arc"</code> , <code>"diagonal"</code> .
<code>node_size</code>	The size of the nodes.

<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
<code>edge_engine</code>	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using ggarrow geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see ggdag_options_set()).
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.

Details

`node_equivalent_dags()` restores columns that the input `tidy_dagitty` carries beyond the standard ones, such as `label` or `status`, by joining them back on node name. Only node-level columns survive: a column whose value varies across the edges of a node cannot be matched to the edges of the equivalent DAGs, so the value of its first edge is used for every row of that node.

Value

a tidy_dagitty with at least one DAG, including a dag column to identify graph set for equivalent DAGs or a reversable column for equivalent classes, or a ggplot

Edge layers of the composite plotters

The plotters that color or fade edges by an analysis column build their edge layers themselves, and which layers they build is settled from the DAG they are called with: a DAG with no bidirected edge is given no bidirected edge layer. Replacing the data of the returned plot afterwards, with ggplot2's `%>%`, does not bring a layer back, so a plot built for one DAG is not a template for another.

Examples

```
g_ex <- dagify(y ~ x + z, x ~ z)

g_ex |> node_equivalent_class()

g_ex |> ggdag_equivalent_dags()
```

Exogenous Variables *Find Exogenous Variables*

Description

`node_exogenous` tags exogenous variables, the variables with no directed arrows pointing at them. `ggdag_exogenous` plots all exogenous variables. See `dagitty::exogenousVariables()` for details.

Usage

```
node_exogenous(.dag, ...)

ggdag_exogenous(
  .tidy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
```

```

    use_nodes = ggdag_option("use_nodes", TRUE),
    use_stylized = ggdag_option("use_stylized", FALSE),
    use_text = ggdag_option("use_text", TRUE),
    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    unified_legend = TRUE,
    text = NULL,
    label = NULL,
    node = deprecated(),
    stylized = deprecated()
  )

```

Arguments

<code>.dag</code> , <code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*</code> () function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.

text	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
label	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
node	Deprecated.
stylized	Deprecated.

Value

a `tidy_dagitty` with an exogenous column for exogenous variables or a `ggplot`

Examples

```
dag <- dagify(y ~ x1 + x2 + x3, b ~ x1 + x2)
ggdag_exogenous(dag)
node_exogenous(dag)
```

expand_plot	<i>Quickly scale the size of a ggplot</i>
-------------	---

Description

`expand_plot()` is a convenience function that expands the scales of a `ggplot`, as the large node sizes in a DAG will often get clipped in themes that don't have DAGs in mind.

Usage

```
expand_plot(
  expand_x = expansion(c(0.1, 0.1)),
  expand_y = expansion(c(0.1, 0.1))
)
```

Arguments

`expand_x`, `expand_y`

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 `ggplot2::expansion()` to generate the values for the `expand` argument.

fortify	<i>Fortify a tidy_dagitty object for ggplot2</i>
---------	--

Description

Fortify a tidy_dagitty object for ggplot2

Usage

```
## S3 method for class 'tidy_dagitty'
fortify(model, data = NULL, ...)

## S3 method for class 'dagitty'
fortify(model, data = NULL, ...)
```

Arguments

model	an object of class tidy_dagitty or dagitty
data	(not used)
...	(not used)

geom_dag	<i>Add common DAG layers to a ggplot</i>
----------	--

Description

geom_dag() is a helper function that adds common DAG layers to a ggplot. The purpose of geom_dag() is to simplify making custom DAGs. Most custom DAGs need the same basic layers, and so this function greatly reduces typing. It is not a true geom in that it adds many types of geoms to the plot (by default, edges, nodes, and text). While the underlying layers, all available in ggdag, are true geoms, we usually need a consistent set of layers to make a DAG. geom_dag() provides this. Because geom_dag() is not a true geom, you'll find that it is awkward for sophisticated customization. When you hit that point, you should use the underlying geoms directly.

Usage

```
geom_dag(
  data = NULL,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
```

```

label_col = ggdag_option("label_col", "black"),
edge_width = ggdag_option("edge_width", 0.6),
edge_cap = ggdag_option("edge_cap", 8),
arrow_length = ggdag_option("arrow_length", 5),
use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
n_edge_points = NULL,
n_node_points = NULL,
unified_legend = TRUE,
key_glyph = NULL,
label = NULL,
text = NULL,
node = deprecated(),
stylized = deprecated()
)

```

Arguments

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 <code>ggplot()</code>. A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> 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. <code>~ head(.x, 10)</code>).
size	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
edge_type	The type of edge, one of "link_arc", "link", "arc", "diagonal".
edge_engine	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using <code>ggarrow</code> geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see <code>ggdag_options_set()</code>).
node_size	The size of the nodes.
text_size	The size of the text.
label_size	The size of the labels.
text_col	The color of the text.
label_col	The color of the labels.
edge_width	The width of the edges.
edge_cap	The size of edge caps (the distance between the arrowheads and the node borders).

arrow_length	The length of arrows on edges.
use_edges	A logical value. Include a geom_dag_edges*() function? If TRUE, which is determined by edge_type.
use_nodes	A logical value. Include geom_dag_point()?
use_stylized	A logical value. Include geom_dag_node()?
use_text	A logical value. Include geom_dag_text()?
use_labels	A logical value. Include a label geom? The specific geom used is controlled by label_geom.
label_geom	A geom function to use for drawing labels when use_labels = TRUE. Default is geom_dag_label_repel. Other options include geom_dag_label, geom_dag_text_repel, geom_dag_label_repel2, and geom_dag_text_repel2.
n_edge_points	Number of invisible points to interpolate along each edge for label repulsion. Passed to repel label geoms. Defaults to NULL (uses StatNodesRepel default of 50). Set to 0 to disable.
n_node_points	Target number of invisible skeleton points filling the disc that covers each node for label repulsion: a center point plus four concentric rings, each holding at least six points, so every value from 1 to 16 gives the same 25 points per node. Passed to repel label geoms. Defaults to NULL (uses StatNodesRepel default of 12). Set to 0 to disable. The disc is measured in data units, so it matches the drawn node only on a panel about 180 mm wide; see geom_dag_label_repel() .
unified_legend	A logical value. When TRUE and both use_edges and use_nodes are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
key_glyph	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the unified_legend setting. When provided, this overrides the automatic selection. Common options include draw_key_dag_point, draw_key_dag_combined, and draw_key_dag_collider.
label	The bare name of a column to use for labels. If use_labels = TRUE, the default is to use label.
text	The bare name of a column to use for geom_dag_text(). If use_text = TRUE, the default is to use name.
node	Deprecated.
stylized	Deprecated.

Value

A list of ggplot2 layer elements

Examples

```
# Basic usage with ggdag
library(ggplot2)
dag <- dagify(y ~ x, z ~ y)
ggplot(dag, aes_dag()) +
```

```

    geom_dag()
  ggplot(dag, aes_dag()) +
    geom_dag(size = 1.5)
  ggplot(dag, aes_dag()) +
    geom_dag(size = 1.5, text_size = 8)

# Using different label geoms
dag_labeled <- dagify(
  y ~ x,
  z ~ y,
  labels = c(x = "Exposure", y = "Outcome", z = "Mediator")
)

# Default: repelling labels
ggplot(dag_labeled, aes_dag()) +
  geom_dag(use_labels = TRUE)

# Static labels
ggplot(dag_labeled, aes_dag()) +
  geom_dag(use_labels = TRUE, label_geom = geom_dag_label)

# Repelling text instead of labels
ggplot(dag_labeled, aes_dag()) +
  geom_dag(use_labels = TRUE, label_geom = geom_dag_text_repel)

```

geom_dag_arrow

Directed DAG edges using ggarrow

Description

These geoms draw DAG edges using the ggarrow package for rendering, providing richer arrow styling than the default ggraph-based edge geoms. `geom_dag_arrow()` draws straight directed edges, `geom_dag_arrow_arc()` draws curved edges (typically for bidirected relationships), and `geom_dag_arrows()` is a convenience wrapper that draws both directed and bidirected edges.

Usage

```

geom_dag_arrow(
  mapping = NULL,
  data = NULL,
  arrow_head = ggarrow::arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,

```

```

    force_arrow = FALSE,
    mid_place = 0.5,
    resect = NULL,
    resect_head = NULL,
    resect_fins = NULL,
    lineend = "butt",
    linejoin = "round",
    linemitre = 10,
    position = "identity",
    na.rm = TRUE,
    show.legend = NA,
    inherit.aes = TRUE,
    ...
  )

geom_dag_arrow_arc(
  mapping = NULL,
  data = NULL,
  curvature = 0.3,
  angle = 90,
  ncp = 5,
  arrow_head = ggarrow::arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  force_arrow = FALSE,
  mid_place = 0.5,
  resect = NULL,
  resect_head = NULL,
  resect_fins = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

geom_dag_arrows(
  mapping = NULL,
  data_directed = filter_direction("->"),
  data_bidirected = filter_direction("<->"),

```

```

    curvature = 0.3,
    arrow_head = ggarrow::arrow_head_wings(),
    arrow_fins = NULL,
    arrow_mid = NULL,
    resect = NULL,
    resect_head = NULL,
    resect_fins = NULL,
    position = "identity",
    na.rm = TRUE,
    show.legend = NA,
    inherit.aes = TRUE,
    ...
  )

```

Arguments

mapping	Set of aesthetic mappings created by <code>ggplot2::aes()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot2::ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data.
arrow_head, arrow_fins, arrow_mid	Arrow ornament functions from <code>ggarrow</code> (e.g., <code>ggarrow::arrow_head_wings()</code> , <code>ggarrow::arrow_head_line()</code>). Set to <code>NULL</code> to suppress an ornament.
length, length_head, length_fins, length_mid	Size of arrow ornaments. A numeric value sets the size relative to <code>linewidth</code> ; a <code>grid::unit()</code> sets an absolute size.
justify	A numeric value between 0 and 1 controlling where the arrow is drawn relative to the path endpoints. 0 (default) places the tip at the endpoint; 1 places the base at the endpoint.
force_arrow	If <code>TRUE</code> , draw arrows even when the path is shorter than the arrow ornaments. Default <code>FALSE</code> .
mid_place	Numeric vector with values between 0 and 1 setting positions for interior arrows, or a <code>grid::unit()</code> for spacing.
resect	A numeric value in millimetres to shorten the arrow from both ends, or <code>NULL</code> (the default) to leave both ends to auto-resection. Overridden by <code>resect_head/resect_fins</code> if set. 0 is a value like any other: it turns auto-resection off and draws the edge up to the node.
resect_head, resect_fins	Numeric values in millimetres to shorten the arrow from the head or fins end respectively, or <code>NULL</code> (the default) to leave that end to auto-resection.
lineend	Line end style: "butt" (default), "round", or "square".
linejoin	Line join style: "round" (default), "mitre", or "bevel".

linemitre	Line mitre limit (default 10).
position	Position adjustment, either as a string or the result of a call to a position adjustment function.
na.rm	If FALSE, removes missing values with a warning. If TRUE (the default for DAG geoms), silently removes missing values.
show.legend	Logical. Should this layer be included in the legends?
inherit.aes	If FALSE, overrides the default aesthetics rather than combining with them.
...	Other arguments passed on to the layer.
curvature	A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.
angle	A numeric value between 0 and 180, giving an amount to skew the control points of the curve.
ncp	The number of control points used to draw the curve. More control points creates a smoother curve.
data_directed, data_bidirected	The data to be displayed for directed and bidirected edges respectively. By default, these filter the plot data by edge direction.

Details

These geoms require the ggarrow package to be installed. Unlike the ggraph-based edge geoms, these use ggarrow's native parameter names (resect_head/resect_fins instead of start_cap/end_cap, arrow_head/arrow_fins instead of arrow).

Per-edge curvature:

geom_dag_arrow_arc() supports per-edge curvature via the edge_curvature aesthetic. Map a numeric column to aes(edge_curvature = ...) to give each edge its own curvature value. Edges with edge_curvature = 0 are drawn as straight lines; positive values curve right, negative values curve left. Any NA values fall back to the scalar curvature parameter. This is useful in time-ordered DAGs where some edges need to curve around intermediate nodes while adjacent edges stay straight.

Auto-resection:

Edges are automatically shortened so that they do not run underneath the nodes. Resection is decided one end at a time: an end you set, through resect or through resect_head/resect_fins, keeps the value you gave it, and every end you leave unset is shortened automatically. Setting only resect_head, for instance, leaves the fins end to the automatic value. Pass 0 to an end to draw the edge all the way to the node.

The automatic value comes from the node size when the plot has a node layer (geom_dag_point() or geom_dag_node()), whichever order the two layers were added in, and from the ggdag.edge_cap option (default: 8mm) when the plot has none.

Value

A `ggplot2::layer()` object that can be added to a plot.

Examples

```

library(ggplot2)
p <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ w2
) |>
ggplot(aes(
  x = .data$x, y = .data$y,
  xend = .data$xend, yend = .data$yend
))

# Straight directed edges
p + geom_dag_arrow() + geom_dag_point() + geom_dag_text() + theme_dag()

# Both directed and bidirected edges
p + geom_dag_arrows() + geom_dag_point() + geom_dag_text() + theme_dag()

# Custom arrow ornaments
p +
  geom_dag_arrow(arrow_head = ggarrow::arrow_head_line()) +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()

# Per-edge curvature: curve long-span edges around intermediate nodes
time_dag <- dagify(
  y ~ x + m,
  m ~ x + c,
  x ~ c,
  coords = time_ordered_coords(force_y = FALSE)
)

add_curvature <- function(x) {
  x <- dplyr::filter(x, !is.na(.data$xend))
  span <- abs(x$x - x$xend)
  x$edge_curvature <- ifelse(span > min(span) + 0.01, 0.5, 0)
  x
}

time_dag |>
ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_dag_arrow_arc(
    aes(edge_curvature = edge_curvature),
    data = add_curvature,
    arrow_fins = NULL
  ) +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()

```

`geom_dag_collider_edges`*Edges for paths activated by stratification on colliders*

Description

Adjusting for a collider activates pathways between the parent of the collider. This geom adds a curved edge between any such parent nodes.

Usage

```
geom_dag_collider_edges(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  linewidth = 0.6,  
  size = NULL,  
  curvature = 0.5,  
  angle = 90,  
  ncp = 5,  
  arrow = NULL,  
  lineend = "butt",  
  na.rm = FALSE,  
  show.legend = NA,  
  inherit.aes = TRUE  
)
```

Arguments

- | | |
|----------------------|---|
| <code>mapping</code> | Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (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. |
| <code>data</code> | <p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot().</p> <p>A <code>data.frame</code>, 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.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p> |

stat	The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following: • A Stat ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following: • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as <code>"jitter"</code>. • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
linewidth	a numeric vector of length 1. Edge width
size	deprecated. Please use <code>linewidth</code> .

curvature	A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.
angle	A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.
ncp	The number of control points used to draw the curve. More control points creates a smoother curve.
arrow	specification for arrow heads, as created by <code>grid::arrow()</code> .
lineend	Line end style (round, butt, square).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>annotation_borders()</code> .

Examples

```
library(dagitty)
library(ggplot2)
dagify(m ~ a + b, x ~ a, y ~ b) |>
  tidy_dagitty() |>
  control_for("m") |>
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted)) +
  geom_dag_edges() +
  geom_dag_collider_edges() +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag() +
  scale_adjusted()
```

geom_dag_edges

Directed and bidirected DAG edges

Description

Directed and bidirected DAG edges

Usage

```
geom_dag_edges(
  mapping = NULL,
  data_directed = filter_direction(">"),
  data_bidirected = filter_direction("<>"),
  curvature = 0.3,
  arrow_directed = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  arrow_bidirected = grid::arrow(length = grid::unit(5, "pt"), ends = "both", type =
    "closed"),
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  fold = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes</code> = <code>TRUE</code> (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_directed, data_bidirected	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> ., and will be used as the layer data.
curvature	The bend of the curve. 1 approximates a halfcircle while 0 will give a straight line. Negative number will change the direction of the curve. Only used if <code>layout_circular</code> = <code>FALSE</code> .
arrow_directed, arrow_bidirected	specification for arrow heads, as created by <code>arrow()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>TRUE</code> , the default, missing values are removed silently. A node with no outgoing edge has a missing edge end, so the edge layers drop those rows rather than warning about them. If <code>FALSE</code> , missing values are removed with a warning.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .
fold	Logical. Should arcs appear on the same side of the nodes despite different directions. Default to <code>FALSE</code> .

... Other arguments passed to `ggraph::geom_edge_*`()

Aesthetics

`geom_dag_edges` understand the following aesthetics. Bold aesthetics are required.

- **x**
- **y**
- **xend**
- **yend**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- start_cap
- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

`geom_dag_edges` also uses `geom_dag_edges_arc`, which requires the **circular** aesthetic, but this is automatically set.

Examples

```
library(ggplot2)
dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ ~w2
) |>
ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()
```

geom_dag_label	<i>Node text labels</i>
----------------	-------------------------

Description

Node text labels

Usage

```
geom_dag_label(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- | | |
|---------|---|
| mapping | Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (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 | <p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot().</p> <p>A <code>data.frame</code>, 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.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p> |
| stat | <p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> • A Stat ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. |

	For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: - The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. - A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter". - For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. - Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. - When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. - Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. - The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code> .
nudge_x, nudge_y	Horizontal and vertical adjustment to nudge labels by.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Aesthetics

`geom_dag_label` understand the following aesthetics (required aesthetics are in bold):

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

Examples

```
library(ggplot2)
library(ggraph)
g <- dagify(m ~ x + y, y ~ x)

ggdag(g, text = FALSE) + geom_dag_label()

g |>
  tidy_dagitty() |>
  ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_edges(aes(
    start_cap = label_rect(name, padding = margin(2.5, 2.5, 2.5, 2.5, "mm")),
    end_cap = label_rect(name, padding = margin(2.5, 2.5, 2.5, 2.5, "mm"))
  )) +
  geom_dag_label(size = 5, fill = "black", color = "white") +
  theme_dag()
```

geom_dag_text	<i>Node text</i>
---------------	------------------

Description

Node text

Usage

```
geom_dag_text(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- | | |
|---------|---|
| mapping | Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (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 | <p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot().</p> <p>A <code>data.frame</code>, 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.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p> |
| stat | <p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> • A Stat ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. |

	For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: - The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. - A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter". - For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. - Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. - When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. - Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. - The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code> .
nudge_x, nudge_y	Horizontal and vertical adjustment to nudge labels by.
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted. <code>check_overlap</code> happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling <code>geom_text()</code> . Note that this argument is not supported by <code>geom_label()</code> .
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. [annotation_borders\(\)](#).

Aesthetics

`geom_dag_text` understand the following aesthetics (required aesthetics are in bold):

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

Examples

```
library(ggplot2)
g <- dagify(m ~ x + y, y ~ x)
g |>
  tidy_dagitty() |>
  ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_point() +
  geom_dag_edges() +
  geom_dag_text() +
  theme_dag()
```

ggdag

*Quickly plot a DAG in ggplot2***Description**

ggdag() is a wrapper to quickly plot DAGs.

Usage

```
ggdag(
  .tidy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  n_edge_points = NULL,
  n_node_points = NULL,
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)
```

Arguments

.tidy_dag	A tidy_dagitty or dagitty object
...	additional arguments passed to tidy_dagitty()
size	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
edge_type	The type of edge, one of "link_arc", "link", "arc", "diagonal".

edge_engine	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using ggarrow geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see ggdag_options_set()).
node_size	The size of the nodes.
text_size	The size of the text.
label_size	The size of the labels.
text_col	The color of the text.
label_col	The color of the labels.
edge_width	The width of the edges.
edge_cap	The size of edge caps (the distance between the arrowheads and the node borders).
arrow_length	The length of arrows on edges.
use_edges	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
use_nodes	A logical value. Include <code>geom_dag_point()</code> ?
use_stylized	A logical value. Include <code>geom_dag_node()</code> ?
use_text	A logical value. Include <code>geom_dag_text()</code> ?
use_labels	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
label_geom	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
n_edge_points	Number of invisible points to interpolate along each edge for label repulsion. Passed to <code>repel</code> label geoms. Defaults to NULL (uses <code>StatNodesRepel</code> default of 50). Set to 0 to disable.
n_node_points	Target number of invisible skeleton points filling the disc that covers each node for label repulsion: a center point plus four concentric rings, each holding at least six points, so every value from 1 to 16 gives the same 25 points per node. Passed to <code>repel</code> label geoms. Defaults to NULL (uses <code>StatNodesRepel</code> default of 12). Set to 0 to disable. The disc is measured in data units, so it matches the drawn node only on a panel about 180 mm wide; see geom_dag_label_repel() .
unified_legend	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
key_glyph	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .
text	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use name.

label	The bare name of a column to use for labels. If use_labels = TRUE, the default is to use label.
node	Deprecated.
stylized	Deprecated.

Value

a ggplot

See Also

[ggdag_classic\(\)](#)

Examples

```
dag <- dagify(
  y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~ ~w2
)

ggdag(dag)
ggdag(dag) + theme_dag()

ggdag(dagitty::randomDAG(5, 0.5))
```

ggdag_classic

Quickly plot a DAG in ggplot2

Description

ggdag_classic() is a wrapper to quickly plot DAGs in a more traditional style.

Usage

```
ggdag_classic(
  .tdy_dag,
  ...,
  size = 8,
  label_rect_size = NULL,
  text_label = "name",
  text_col = "black",
  use_edges = TRUE
)
```

Arguments

<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>size</code>	text size, with a default of 8.
<code>label_rect_size</code>	specify the <code>fontsize</code> argument in <code>ggraph::label_rect</code> ; default is <code>NULL</code> , in which case it is scaled relative to <code>size</code>
<code>text_label</code>	text variable, with a default of "name"
<code>text_col</code>	text color, with a default of "black"
<code>use_edges</code>	logical value whether to include edges

Value

a `ggplot`

See Also

[ggdag\(\)](#)

Examples

```
dag <- dagify(  
  y ~ x + z2 + w2 + w1,  
  x ~ z1 + w1,  
  z1 ~ w1 + v,  
  z2 ~ w2 + v,  
  w1 ~ ~w2  
)  
  
ggdag_classic(dag)  
ggdag_classic(dag) + theme_dag_blank()  
  
ggdag_classic(dagitty::randomDAG(5, 0.5))
```

<code>ggdag_defaults</code>	<i>Global DAG Options</i>
-----------------------------	---------------------------

Description

Set, get, and reset global default options for DAG appearance. These options are used as defaults by all `geom_dag()`, `ggdag_*`, and related functions.

Usage

```

ggdag_defaults

ggdag_options_set(...)

ggdag_options_get(name = NULL)

ggdag_options_reset()

ggdag_option(name, default)

ggdag_option_proportional(name, base_default, override_default)

```

Arguments

<code>...</code>	Named option values to set. See <code>ggdag_defaults</code> for valid names and types.
<code>name</code>	Character string. The option name (without the <code>ggdag.</code> prefix). If <code>NULL</code> , returns all currently-set <code>ggdag</code> options.
<code>default</code>	Default value to return if the option is not set.
<code>base_default</code>	The base default for this option (e.g., 8 for <code>edge_cap</code>).
<code>override_default</code>	The override default used by certain functions (e.g., 10 for <code>edge_cap</code> in adjustment set functions).

Details

Options are stored in R's global `options()` as `ggdag.<name>`. When an option is `NULL` (the default), each function uses its own built-in default. Setting a global option overrides the built-in default for all functions that use it. Passing `NULL` to `ggdag_options_set()` leaves that one option unset, which returns it to the built-in default and leaves every other option alone. That also makes the previous values `ggdag_options_set()` returns safe to restore with `do.call(ggdag_options_set, old)`.

Functions that normally use `edge_cap = 10` (e.g., `ggdag_adjustment_set()`, `ggdag_drelationship()`) maintain a proportional offset. If you set `ggdag.edge_cap` to a custom value, these functions scale it by $10/8$.

`debug_repel_points` is a diagnostic rather than an appearance setting. When it is `TRUE`, every repelling label geom (see `geom_dag_label_repel()`) adds a layer of purple points showing the invisible geometry that labels are repelled from: the points traced along each edge and the disc filling each node. It is useful for understanding why a label came to rest where it did.

Value

- `ggdag_options_set()`: Invisibly returns a named list of the previous option values.
- `ggdag_options_get()`: The option value, or a named list of all set options if `name` is `NULL`.
- `ggdag_options_reset()`: Called for its side effect; returns `NULL` invisibly.
- `ggdag_option()`: The option value if set, otherwise default.
- `ggdag_option_proportional()`: The scaled option value if set, otherwise `override_default`.

Examples

```
# Set global options
old <- ggdag_options_set(node_size = 20, text_size = 5)

# Check current value
ggdag_options_get("node_size")

# Reset to defaults
ggdag_options_reset()
```

ggplot.tidy_dagitty *Create a new ggplot*

Description

Create a new ggplot

Usage

```
## S3 method for class 'tidy_dagitty'
ggplot(data = NULL, mapping = aes(), ...)

## S3 method for class 'dagitty'
ggplot(data = NULL, mapping = aes(), ...)
```

Arguments

data	Default dataset to use for plot. If not already a data.frame, will be converted to one by <code>fortify()</code> . If not specified, must be supplied in each layer added to the plot.
mapping	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
...	Other arguments passed on to methods. Not currently used.

ggrepel functions *Repulsive textual annotations*

Description

These functions are minor modifications of those in the `ggrepel` package. `geom_dag_text_repel()` adds text directly to the plot. `geom_dag_label_repel()` draws a rectangle underneath the text, making it easier to read. The text labels repel away from each other and away from the data points. `geom_dag_label_repel2()` is a slightly stylized version of `geom_dag_label_repel()` that often looks better on DAGs. `geom_dag_text_repel2()` is a slightly stylized version of `geom_dag_text_repel()` that often looks better on DAGs.

Usage

```

geom_dag_text_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  node_size = NULL,
  n_edge_points = NULL,
  n_node_points = NULL,
  box.padding = 1.25,
  point.padding = 1,
  min.segment.length = 0.5,
  segment.color = "#666666",
  segment.alpha = 1,
  fontface = "bold",
  segment.size = 0.5,
  arrow = NULL,
  force = 1,
  force_pull = 1,
  max.time = 0.5,
  max.iter = 2000,
  max.overlaps = Inf,
  nudge_x = 0,
  nudge_y = 0,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  na.rm = FALSE,
  show.legend = NA,
  direction = c("both", "y", "x"),
  seed = NA,
  verbose = getOption("verbose", default = FALSE),
  inherit.aes = TRUE
)

geom_dag_label_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  node_size = NULL,
  n_edge_points = NULL,
  n_node_points = NULL,
  box.padding = grid::unit(1.25, "lines"),
  label.padding = grid::unit(0.25, "lines"),

```

```

    point.padding = grid::unit(1, "lines"),
    label.r = grid::unit(0.15, "lines"),
    label.size = 0.25,
    min.segment.length = 0.5,
    segment.color = "grey50",
    segment.alpha = 1,
    segment.size = 0.5,
    arrow = NULL,
    force = 1,
    force_pull = 1,
    max.time = 0.5,
    max.iter = 2000,
    max.overlaps = Inf,
    nudge_x = 0,
    nudge_y = 0,
    xlim = c(NA, NA),
    ylim = c(NA, NA),
    na.rm = FALSE,
    show.legend = NA,
    direction = c("both", "y", "x"),
    seed = NA,
    verbose = getOption("verbose", default = FALSE),
    inherit.aes = TRUE
  )

  geom_dag_label_repel2(
    mapping = NULL,
    data = NULL,
    box.padding = 2,
    max.overlaps = Inf,
    label.size = NA,
    linewidth = 0,
    ...
  )

  geom_dag_text_repel2(
    mapping = NULL,
    data = NULL,
    box.padding = 2,
    max.overlaps = Inf,
    ...
  )

```

Arguments

mapping Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes` = TRUE (the default), is combined with the default mapping at the top level of the plot. You only need to supply mapping if there isn't a mapping defined for the plot.

<code>data</code>	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
<code>stat</code>	The statistical transformation to use on the data for this layer, as a string.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>parse</code>	If TRUE, the labels will be parsed into expressions and displayed as described in <code>?plotmath</code>
<code>...</code>	other arguments passed on to <code>layer</code> . There are three types of arguments you can use here: • Aesthetics: to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code>. • Other arguments to the layer, for example you override the default <code>stat</code> associated with the layer. • Other arguments passed on to the <code>stat</code>.
<code>node_size</code>	The size of the DAG nodes, used to compute the <code>point.size</code> aesthetic so that labels repel from the node boundary rather than the node center, and to size the skeleton discs described under Details. Defaults to NULL, which auto-discovers the size from a node layer (<code>geom_dag_node()</code> or <code>geom_dag_point()</code>) already added to the plot. Falls back to 16 if no node layer is found.
<code>n_edge_points</code>	Number of invisible points to interpolate along each edge. These "fake" points participate in ggrepel's repulsion calculation so that labels avoid overlapping edges. Defaults to NULL, which uses the <code>StatNodesRepel</code> default of 50. Set to 0 to disable edge-aware repulsion.
<code>n_node_points</code>	Target number of invisible points filling the disc that covers each node: a center point plus four concentric rings. Each ring holds at least six points, so every value from 1 to 16 gives the same 25 points per node and the parameter only starts to take effect above 16. Defaults to NULL, which uses the <code>StatNodesRepel</code> default of 12. Set to 0 to disable node skeleton repulsion.
<code>box.padding</code>	Amount of padding around bounding box, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>point.padding</code>	Amount of padding around labeled point, as unit or number. Defaults to 0. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>min.segment.length</code>	Skip drawing segments shorter than this, as unit or number. Defaults to 0.5. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>segment.color</code> , <code>segment.size</code>	See <code>ggrepel::geom_text_repel()</code>
<code>segment.alpha</code>	Transparency of the line segment. Set to NULL (default) to use ggrepel's default behavior, or provide a value between 0 and 1
<code>fontface</code>	A character vector. Default is "bold"
<code>arrow</code>	specification for arrow heads, as created by <code>arrow</code>
<code>force</code>	Force of repulsion between overlapping text labels. Defaults to 1.
<code>force_pull</code>	Force of attraction between a text label and its corresponding data point. Defaults to 1.

<code>max.time</code>	Maximum number of seconds to try to resolve overlaps. Defaults to 0.5.
<code>max.iter</code>	Maximum number of iterations to try to resolve overlaps. Defaults to 10000.
<code>max.overlaps</code>	Exclude text labels when they overlap too many other things. For each text label, we count how many other text labels or other data points it overlaps, and exclude the text label if it has too many overlaps. Defaults to 10.
<code>nudge_x, nudge_y</code>	Horizontal and vertical adjustments to nudge the starting position of each text label. The units for <code>nudge_x</code> and <code>nudge_y</code> are the same as for the data units on the x-axis and y-axis.
<code>xlim, ylim</code>	Limits for the x and y axes. Text labels will be constrained to these limits. By default, text labels are constrained to the entire plot area.
<code>na.rm</code>	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
<code>show.legend</code>	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.
<code>direction</code>	"both", "x", or "y" – direction in which to adjust position of labels
<code>seed</code>	Random seed passed to <code>set.seed</code> . Defaults to NA, which means that <code>set.seed</code> will not be called.
<code>verbose</code>	If TRUE, some diagnostics of the repel algorithm are printed
<code>inherit.aes</code>	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders</code> .
<code>label.padding</code>	Amount of padding around label, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>label.r</code>	Radius of rounded corners, as unit or number. Defaults to 0.15. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>label.size</code>	Size of label border, in mm.
<code>linewidth</code>	Width of the label border in <code>geom_dag_label_repel2()</code> . Default is 0 (no border). Set to a positive value to show borders.

Details

These geoms are wrappers around `ggrepel::geom_text_repel()` and `ggrepel::geom_label_repel()` that use the custom `StatNodesRepel` for better handling of DAG data. All arguments available in `ggrepel` functions are supported.

Labels are kept off nodes and edges by two mechanisms. The `point.size` aesthetic, computed from `node_size`, is converted by `ggrepel` at draw time and so describes the same circle at every device size. The invisible points along edges and the disc filling each node, on the other hand, are placed in data units: the disc radius is `node_size` times the average spread of the nodes divided by 400 (the same divisor sizes the debug overlay described in `ggdag_options_set()`). Because the panel converts data units to millimetres at drawing time, that disc is congruent with the drawn node only on a panel about 180 mm wide. On a narrower device the disc is smaller than the node it stands for and a label may come to rest on the node; on a wider one it is larger and labels are pushed further away than they need to be. Set `n_node_points = 0` to rely on `point.size` alone.

Points along an edge trace the path that edge is drawn along, including the arc of a bidirected edge and of `geom_dag_edges_arc()`. Edges drawn by `geom_dag_edges_diagonal()`, `geom_dag_edges_fan()`, and the `ggarrow` engine are traced along the straight line between their nodes.

Additional segment parameters can be passed through `...`, including:

- `segment.linetype`: Line style
- `segment.alpha`: Line transparency
- `segment.curvature`: Curve amount
- `segment.angle`: Curve angle
- `segment.ncp`: Number of control points
- `segment.shape`: Control point position
- `segment.square`: Square formation control points
- `segment.squareShape`: Square formation shape
- `segment.inflect`: Add inflection point
- `segment.debug`: Show debug information

You can also pass `point.size` and `point.colour` through `....`

Examples

```
library(ggplot2)
g <- dagify(
  m ~ x + y,
  y ~ x,
  exposure = "x",
  outcome = "y",
  latent = "m",
  labels = c("x" = "Exposure", "y" = "Outcome", "m" = "Collider")
)

g |>
  tidy_dagitty() |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text_repel(aes(label = name), show.legend = FALSE) +
  theme_dag()

# Use nudge_x and nudge_y to push labels away from nodes
g |>
  tidy_dagitty() |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text_repel(
    aes(label = name),
    nudge_x = 0.1,
    nudge_y = 0.1
  ) +
```

```

    theme_dag()

# Use position_nudge_repel for the same effect
g |>
  tidy_dagitty() |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text_repel(
    aes(label = name),
    position = ggrepel::position_nudge_repel(x = 0.1, y = 0.1)
  ) +
  theme_dag()

g |>
  tidy_dagitty() |>
  dag_label(labels = c(
    "x" = "This is the exposure",
    "y" = "Here's the outcome",
    "m" = "Here is where they collide"
  )) |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text() +
  geom_dag_label_repel(
    aes(label = label, fill = label),
    col = "white",
    show.legend = FALSE
  ) +
  theme_dag()

# Use directional repulsion
g |>
  tidy_dagitty() |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text_repel(
    aes(label = name),
    direction = "y",
    seed = 1234
  ) +
  theme_dag()

# Customize segment appearance
g |>
  tidy_dagitty() |>
  ggplot(aes_dag()) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text_repel(
    aes(label = name),

```

```

    segment.linetype = 2,
    segment.alpha = 0.5,
    segment.curvature = -0.3
  ) +
  theme_dag()

```

Instrumental Variables

Find Instrumental Variables

Description

`node_instrumental` tags instrumental variables given an exposure and outcome. `ggdag_instrumental` plots all instrumental variables. See `dagitty::instrumentalVariables()` for details.

Usage

```
node_instrumental(.dag, exposure = NULL, outcome = NULL, ...)
```

```

ggdag_instrumental(
  .tdy_dag,
  exposure = NULL,
  outcome = NULL,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option_proportional("edge_cap", 8, 10),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

Arguments

<code>.dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>exposure</code>	A character vector, the exposure variable. Default is <code>NULL</code> , in which case it will be determined from the DAG.
<code>outcome</code>	A character vector, the outcome variable. Default is <code>NULL</code> , in which case it will be determined from the DAG.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>.tidy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If <code>TRUE</code> , which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When <code>TRUE</code> and both <code>use_edges</code> and <code>use_nodes</code> are <code>TRUE</code> , creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is <code>TRUE</code> .
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.

Value

a tidy_dagitty with an instrumental column for instrumental variables or a ggplot

Examples

```
library(dagitty)

node_instrumental(dagitty("dag{ i->x->y; x<->y }"), "x", "y")
ggdag_instrumental(dagitty("dag{ i->x->y; i2->x->y; x<->y }"), "x", "y")
```

is.tidy_dagitty	<i>Test for object class for tidy_dagitty</i>
-----------------	---

Description

Test for object class for tidy_dagitty

Usage

```
is.tidy_dagitty(x)
```

Arguments

x object to be tested

is_confounder	<i>Assess if a variable confounds a relationship</i>
---------------	--

Description

Assess if a variable confounds a relationship

Usage

```
is_confounder(.tdy_dag, z, x, y, direct = FALSE)
```

Arguments

.tdy_dag A tidy_dagitty or dagitty object
z a character vector, the potential confounder
x, y a character vector, the variables z may confound.
direct logical. Only consider direct confounding? Default is FALSE

Details

A confounder is a common cause of x and y . z therefore has to reach x by a directed path that does not run through y , and reach y by a directed path that does not run through x . Being a descendant of z is not enough: descent is transitive through x , so every upstream cause of the exposure, such as an instrument, would qualify even though it opens no backdoor path.

Value

Logical. Is the variable a confounder?

Examples

```
dag <- dagify(y ~ z, x ~ z)

is_confounder(dag, "z", "x", "y")
is_confounder(dag, "x", "z", "y")
```

is_dag_properties	<i>Test DAG properties</i>
-------------------	----------------------------

Description

These functions test various properties of DAGs:

- `is_acyclic()` tests whether a DAG is acyclic
- `is_adjustment_set()` tests whether a set of variables is a valid adjustment set
- `is_d_separated()` tests whether two sets of variables are d-separated
- `is_d_connected()` tests whether two sets of variables are d-connected

Usage

```
is_acyclic(.dag)

is_adjustment_set(.dag, Z, exposure = NULL, outcome = NULL)

is_d_separated(.dag, from = NULL, to = NULL, controlling_for = NULL)

is_d_connected(.dag, from = NULL, to = NULL, controlling_for = NULL)
```

Arguments

<code>.dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>Z</code>	A set of variables to test or condition on. This can be a character vector of variable names, a list of the form <code>list(c(...))</code> , or <code>NULL</code> .
<code>exposure</code>	A character vector, the exposure variable. Default is <code>NULL</code> , in which case it will be determined from the DAG.

outcome	A character vector, the outcome variable. Default is NULL, in which case it will be determined from the DAG.
from	A character vector with starting node name(s), or NULL. If NULL, checks DAG for exposure variable.
to	A character vector with ending node name(s), or NULL. If NULL, checks DAG for outcome variable.
controlling_for	A set of variables to control for. This can be a character vector of variable names, a list of the form <code>list(c(...))</code> , or NULL. When NULL, no control is applied. Default is NULL.

Value

A logical value indicating whether the tested property holds

Examples

```
dag <- dagify(
  y ~ x + z,
  x ~ z,
  exposure = "x",
  outcome = "y"
)

is_acyclic(dag)
is_adjustment_set(dag, "z")
is_d_separated(dag, "x", "y", "z")
is_d_connected(dag, "x", "y")
```

is_node_properties	<i>Test node properties</i>
--------------------	-----------------------------

Description

These functions test various properties of nodes in a DAG:

- `is_exogenous()` tests whether a variable is exogenous (has no parents)
- `is_instrumental()` tests whether a variable is instrumental
- `is_exposure()`, `is_outcome()`, `is_latent()` test variable status

Usage

```
is_exogenous(.dag, .var)
```

```
is_instrumental(.dag, .var, exposure = NULL, outcome = NULL)
```

```
is_exposure(.dag, .var)
```

```
is_outcome(.dag, .var)
```

```
is_latent(.dag, .var)
```

Arguments

<code>.dag</code>	A tidy_dagitty or dagitty object
<code>.var</code>	A character string specifying the variable to test
<code>exposure</code>	A character vector, the exposure variable. Default is NULL, in which case it will be determined from the DAG.
<code>outcome</code>	A character vector, the outcome variable. Default is NULL, in which case it will be determined from the DAG.

Value

A logical value indicating whether the tested property holds

Examples

```
dag <- dagify(  
  y ~ x + z,  
  x ~ z,  
  exposure = "x",  
  outcome = "y",  
  latent = "z"  
)  
  
is_exogenous(dag, "z")  
is_exposure(dag, "x")  
is_outcome(dag, "y")  
is_latent(dag, "z")
```

is_node_relationships *Test node relationships*

Description

These functions test relationships between nodes in a DAG:

- `is_parent()` tests whether one node is a parent of another
- `is_child()` tests whether one node is a child of another
- `is_ancestor()` tests whether one node is an ancestor of another
- `is_descendant()` tests whether one node is a descendant of another
- `is_adjacent()` tests whether two nodes are adjacent (connected by an edge)

Usage

```
is_parent(.dag, .var, .node)

is_child(.dag, .var, .node)

is_ancestor(.dag, .var, .node)

is_descendant(.dag, .var, .node)

is_adjacent(.dag, .var, .node)
```

Arguments

<code>.dag</code>	A tidy_dagitty or dagitty object
<code>.var</code>	A character string specifying the variable to test
<code>.node</code>	A character string specifying the reference node

Value

A logical value indicating whether the relationship holds

Examples

```
dag <- dagify(
  y ~ x + z,
  x ~ z
)

is_parent(dag, "z", "x")
is_child(dag, "x", "z")
is_ancestor(dag, "z", "y")
is_descendant(dag, "y", "z")
is_adjacent(dag, "x", "y")
```

Nodes

DAG Nodes

Description

`geom_dag_node` and `geom_dag_point` are very similar to [ggplot2::geom_point](#) but with a few defaults changed. `geom_dag_node` is slightly stylized and includes an internal white circle, while `geom_dag_point` plots a single point.

Usage

```
geom_dag_node(
  mapping = NULL,
  data = NULL,
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  key_glyph = NULL
)

geom_dag_point(
  mapping = NULL,
  data = NULL,
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  key_glyph = NULL
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot(). A <code>data.frame</code>, 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 <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.

...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
<code>inherit.aes</code>	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>annotation_borders()</code> .
<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .

Aesthetics

`geom_dag_node` and `geom_dag_point` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour

- fill
- shape
- size
- stroke

`geom_dag_node` also accepts:

- `internal_colour`

Examples

```
library(ggplot2)
g <- dagify(m ~ x + y, y ~ x)
p <- g |>
  tidy_dagitty() |>
  ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_edges() +
  theme_dag()

p +
  geom_dag_node() +
  geom_dag_text()

p +
  geom_dag_point() +
  geom_dag_text()
```

Pathways

Find Open Paths Between Variables

Description

`dag_paths` finds open paths between a given exposure and outcome. `ggdag_paths` and `ggdag_paths_fan` plot all open paths. See [dagitty::paths\(\)](#) for details.

Usage

```
dag_paths(
  .dag,
  from = NULL,
  to = NULL,
  adjust_for = NULL,
  limit = 100,
  directed = FALSE,
  paths_only = FALSE,
  ...
)
```

```

ggdag_paths(
  .tdy_dag,
  from = NULL,
  to = NULL,
  adjust_for = NULL,
  limit = 100,
  directed = FALSE,
  shadow = TRUE,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

```

ggdag_paths_fan(
  .tdy_dag,
  from = NULL,
  to = NULL,
  adjust_for = NULL,
  limit = 100,
  directed = FALSE,
  ...,
  shadow = TRUE,
  spread = 0.7,
  size = 1,
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),

```

```

edge_width = ggdag_option("edge_width", 0.6),
edge_cap = ggdag_option("edge_cap", 8),
arrow_length = ggdag_option("arrow_length", 5),
use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
unified_legend = TRUE,
key_glyph = NULL,
edge_engine = ggdag_option("edge_engine", "ggraph"),
text = NULL,
label = NULL,
node = deprecated(),
stylized = deprecated()
)

```

Arguments

<code>.dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>from</code>	A character vector with starting node name(s), or <code>NULL</code> . If <code>NULL</code> , checks DAG for exposure variable.
<code>to</code>	A character vector with ending node name(s), or <code>NULL</code> . If <code>NULL</code> , checks DAG for outcome variable.
<code>adjust_for</code>	character vector, a set of variables to control for. Default is <code>NULL</code> .
<code>limit</code>	maximum amount of paths to show. In general, the number of paths grows exponentially with the number of variables in the graph, such that path inspection is not useful except for the most simple models.
<code>directed</code>	logical. Should only directed paths be shown?
<code>paths_only</code>	logical. Should only open paths be returned? Default is <code>FALSE</code> , which includes every variable and edge in the DAG regardless if they are part of the path.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>.tdy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>shadow</code>	logical. Show edges which are not on an open path?
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.

edge_cap	The size of edge caps (the distance between the arrowheads and the node borders).
arrow_length	The length of arrows on edges.
use_edges	A logical value. Include a <code>geom_dag_edges*</code> () function? If TRUE, which is determined by <code>edge_type</code> .
use_nodes	A logical value. Include <code>geom_dag_point</code> ()?
use_stylized	A logical value. Include <code>geom_dag_node</code> ()?
use_text	A logical value. Include <code>geom_dag_text</code> ()?
use_labels	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
label_geom	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
edge_engine	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using <code>ggarrow</code> geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see <code>ggdag_options_set</code> ()).
text	The bare name of a column to use for <code>geom_dag_text</code> (). If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
label	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
node	Deprecated.
stylized	Deprecated.
spread	the width of the fan spread
unified_legend	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
key_glyph	A function to use for drawing the legend key glyph for nodes. If NULL (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .

Value

a `tidy_dagitty` with a `path` column for path variables, a `set` grouping column, and a `path_type` column classifying paths as "direct" (a directed causal path), "backdoor" (a path whose first edge points into the exposure), or "other" (any other path, such as one through a collider), or a `ggplot`.

Edge layers of the composite plotters

The plotters that color or fade edges by an analysis column build their edge layers themselves, and which layers they build is settled from the DAG they are called with: a DAG with no bidirected edge is given no bidirected edge layer. Replacing the data of the returned plot afterwards, with `ggplot2`'s `%>%`, does not bring a layer back, so a plot built for one DAG is not a template for another.

Examples

```

confounder_triangle(x_y_associated = TRUE) |>
  dag_paths(from = "x", to = "y")

confounder_triangle(x_y_associated = TRUE) |>
  ggdag_paths(from = "x", to = "y")

butterfly_bias(x_y_associated = TRUE) |>
  ggdag_paths_fan(shadow = TRUE)

```

print.tidy_dagitty	<i>Print a tidy_dagitty</i>
--------------------	-----------------------------

Description

Print a tidy_dagitty

Usage

```

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

```

Arguments

x	an object of class tidy_dagitty
...	optional arguments passed to format()

pull_dag	<i>Pull components from DAG objects</i>
----------	---

Description

pull_dag() and pull_dag_data() are generic methods to pull components of DAG objects, e.g. tidy_dagitty, such as the dagitty object or the data frame associated with it. These methods are recommended over extracting components manually, e.g. my_dag\$data, because the internal structure of these objects may change over time. Similarly, use update_dag() if you want to sync the data back to the DAG object or override it with another DAG; use update_dag_data() to do update the data frame. This is useful with pull_dag_data().

Usage

```

pull_dag(x, ...)

## S3 method for class 'tidy_dagitty'
pull_dag(x, ...)

## S3 method for class 'dagitty'
pull_dag(x, ...)

pull_dag_data(x, ...)

## S3 method for class 'tidy_dagitty'
pull_dag_data(x, ...)

## S3 method for class 'dagitty'
pull_dag_data(x, ...)

update_dag_data(x) <- value

## S3 replacement method for class 'tidy_dagitty'
update_dag_data(x) <- value

update_dag(x, ...)

update_dag(x) <- value

## S3 method for class 'tidy_dagitty'
update_dag(x, ...)

## S3 replacement method for class 'tidy_dagitty'
update_dag(x) <- value

```

Arguments

<code>x</code>	a <code>tidy_dagitty</code> or <code>dagitty</code> object.
<code>...</code>	For <code>dagitty</code> objects, passed to <code>tidy_dagitty()</code> if needed, otherwise currently unused.
<code>value</code>	a value to set, either a <code>dagitty</code> or <code>data.frame</code> object, depending on the function.

Value

a DAG object, e.g. `dagitty`, or data frame

Examples

```

tidy_dagitty_obj <- dagify(y ~ x + z, x ~ z) |>
  tidy_dagitty()

```

```

dag <- pull_dag(tidy_dagitty_obj)
dag_data <- pull_dag_data(tidy_dagitty_obj)

tidy_dagitty_obj |>
  # rename both endpoints of every edge, or the recompiled DAG will hold a
  # mix of the old and new names
  dplyr::mutate(name = toupper(name), to = toupper(to)) |>
  # recreate the DAG component
  update_dag()

dag_data$label <- paste0(dag_data$name, "(observed)")
update_dag_data(tidy_dagitty_obj) <- dag_data

```

query_adjustment_sets *Query Adjustment Sets*

Description

Find adjustment sets that close backdoor paths between exposure and outcome. Unlike `dag_adjustment_sets()`, this function returns a tibble with the adjustment sets as list columns rather than a `tidy_dagitty` object.

Usage

```

query_adjustment_sets(
  .tdy_dag,
  exposure = NULL,
  outcome = NULL,
  type = c("minimal", "canonical", "all"),
  effect = c("total", "direct"),
  max.results = Inf
)

```

Arguments

<code>.tdy_dag</code>	A <code>tidy_dagitty</code> or <code>dagitty</code> object
<code>exposure</code>	A character vector of exposure variable names. If <code>NULL</code> , uses the exposure defined in the DAG.
<code>outcome</code>	A character vector of outcome variable names. If <code>NULL</code> , uses the outcome defined in the DAG.
<code>type</code>	Character string specifying the type of adjustment sets to find. Options are "minimal" (default), "canonical", or "all".
<code>effect</code>	Character string specifying the effect type. Options are "total" (default) or "direct".
<code>max.results</code>	Maximum number of adjustment sets to return. Default is <code>Inf</code> .

Value

A tibble with columns:

- `set_id`: Integer identifier for each adjustment set
- `type`: Type of adjustment set (minimal, canonical, or all)
- `effect`: Effect type (total or direct)
- `set`: String representation of the adjustment set (e.g., "{a, b, c}")
- `variables`: List column containing the variables in each set

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ z,
  exposure = "x",
  outcome = "y"
)

query_adjustment_sets(dag)
```

query_ancestors

Query Node Ancestors

Description

Find ancestor nodes for specified variables in a DAG.

Usage

```
query_ancestors(.tdy_dag, .var = NULL)
```

Arguments

`.tdy_dag` A tidy_dagitty or dagitty object

`.var` Character vector of variables to query. If NULL, returns parents for all nodes.

Value

A tibble with columns:

- `node`: The node
- `ancestor_set`: String representation of ancestor nodes
- `ancestors`: List column containing ancestor nodes
- `n_ancestors`: Number of ancestors

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w
)

query_ancestors(dag)
query_ancestors(dag, .var = "y")
```

query_children	<i>Query Node Children</i>
----------------	----------------------------

Description

Find child nodes for specified variables in a DAG.

Usage

```
query_children(.tdy_dag, .var = NULL)
```

Arguments

.tdy_dag	A tidy_dagitty or dagitty object
.var	Character vector of variables to query. If NULL, returns parents for all nodes.

Value

A tibble with columns:

- node: The node
- child_set: String representation of child nodes
- children: List column containing child nodes
- n_children: Number of children

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w
)

query_children(dag)
query_children(dag, .var = "x")
```

query_colliders	<i>Query Collider Nodes</i>
-----------------	-----------------------------

Description

Identify all collider nodes in a DAG. A collider is a node that two or more edges point into. Bidirected edges count toward that total, so a node with one directed parent and one bidirected partner is a collider.

Usage

```
query_colliders(.tdy_dag)
```

Arguments

`.tdy_dag` A tidy_dagitty or dagitty object

Value

A tibble with columns:

- `node`: The collider node
- `parent_set`: String representation of the directed parents
- `parents`: List column containing the directed parents. A bidirected partner contributes an arrowhead but is not a parent, so a collider formed by bidirected edges can have fewer than two parents here.
- `is_activated`: Logical indicating if the collider is conditioned on

Examples

```
library(ggdag)
dag <- dagify(
  z ~ x + y,
  w ~ z
)

query_colliders(dag)
```

`query_conditional_independence`*Query and Test Conditional Independence in a DAG*

Description

`query_conditional_independence()` queries conditional independencies implied by a given DAG. These serve as potential robustness checks for your DAG. `test_conditional_independence()` runs the tests of independence implied by the DAG on a given dataset. `ggdag_conditional_independence()` plots the results as a forest plot.

Usage

```
query_conditional_independence(  
  .tdy_dag,  
  type = "missing.edge",  
  max.results = Inf  
)  
  
test_conditional_independence(  
  .tdy_dag,  
  data = NULL,  
  type = c("cis", "cis.loess", "cis.chisq", "cis.pillai", "tetrads", "tetrads.within",  
    "tetrads.between", "tetrads.epistemic"),  
  tests = NULL,  
  sample.cov = NULL,  
  sample.nobs = NULL,  
  conf.level = 0.95,  
  R = NULL,  
  max.conditioning.variables = NULL,  
  abbreviate.names = FALSE,  
  tol = NULL,  
  loess.pars = NULL  
)  
  
ggdag_conditional_independence(  
  .test_result,  
  sort = TRUE,  
  vline_linewidth = 0.8,  
  vline_color = "grey70",  
  point_size = NULL,  
  pointrange_fatten = deprecated()  
)
```

Arguments

`.tdy_dag` A `tidy_dagitty` or `dagitty` object

<code>type</code>	can be one of "missing.edge", "basis.set", or "all.pairs". With the first, one or more minimal testable implication (with the smallest possible conditioning set) is returned per missing edge of the graph. With "basis.set", one testable implication is returned per vertex of the graph that has non-descendants other than its parents. Basis sets can be smaller, but they involve higher-dimensional independencies, whereas missing edge sets involve only independencies between two variables at a time. With "all.pairs", the function will return a list of all implied conditional independencies between two variables at a time. Beware, because this can be a very long list and it may not be feasible to compute this except for small graphs.
<code>max.results</code>	integer. The listing of conditional independencies is stopped once this many results have been found. Use Inf to generate them all. This applies only when <code>type="missing.edge"</code> or <code>type="all"</code> .
<code>data</code>	matrix or data frame containing the data.
<code>tests</code>	list of the precise tests to perform. If not given, the list of tests is automatically derived from the input graph. Can be used to restrict testing to only a certain subset of tests (for instance, to test only those conditional independencies for which the conditioning set is of a reasonably low dimension, such as shown in the example).
<code>sample.cov</code>	the sample covariance matrix; ignored if data is supplied. Either data or <code>sample.cov</code> and <code>sample.nobs</code> must be supplied.
<code>sample.nobs</code>	number of observations; ignored if data is supplied.
<code>conf.level</code>	determines the size of confidence intervals for test statistics.
<code>R</code>	how many bootstrap replicates for estimating confidence intervals. If NULL, then confidence intervals are based on normal approximation. For tetrads, the normal approximation is only valid in large samples even if the data are normally distributed.
<code>max.conditioning.variables</code>	for conditional independence testing, this parameter can be used to perform only those tests where the number of conditioning variables does not exceed the given value. High-dimensional conditional independence tests can be very unreliable.
<code>abbreviate.names</code>	logical. Whether to abbreviate variable names (these are used as row names in the returned data frame).
<code>tol</code>	bound value for tolerated deviation from local test value. By default, we perform a two-sided test of the hypothesis $\theta=0$. If this parameter is given, the test changes to $\text{abs}(\theta)=\text{tol}$ versus $\text{abs}(\theta)>\text{tol}$.
<code>loess.pars</code>	list of parameter to be passed on to loess (for <code>type="cis.loess"</code>), for example the smoothing range. <code>ciTest(X,Y,Z,data)</code> is a convenience function to test a single conditional independence independently of a DAG.
<code>.test_result</code>	A data frame containing the results of conditional independence tests created by <code>test_conditional_independence()</code> .
<code>sort</code>	Logical indicating whether to sort the results by estimate value. Default is TRUE.

vline_linewidth	Line width for the vertical line indicating no effect.
vline_color	Color of the vertical line.
point_size	Size of the point in the point range. Default is NULL, which uses the ggplot2 theme default.
pointrange_fatten	[Deprecated] Use point_size instead.

Value

Either a tibble summarizing the conditional independencies in the DAG or test results, or a ggplot of the results.

query_dconnected	<i>Query D-connection</i>
------------------	---------------------------

Description

Test whether sets of variables are d-connected in a DAG given a conditioning set. This is the complement of d-separation.

Usage

```
query_dconnected(.tdy_dag, from, to, conditioned_on = NULL)
```

Arguments

.tdy_dag	A tidy_dagitty or dagitty object
from	Character vector of node names.
to	Character vector of node names.
conditioned_on	Character vector of conditioning variables.

Value

A tibble with columns:

- from_set: String representation of source nodes
- from: List column of source nodes
- to_set: String representation of target nodes
- to: List column of target nodes
- conditioning_set: String representation of conditioning variables
- conditioned_on: List column of conditioning variables
- dconnected: Logical indicating d-connection

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w,
  z ~ w
)

query_dconnected(dag, from = "x", to = "z")
query_dconnected(dag, from = "x", to = "z", conditioned_on = "w")
```

query_descendants	<i>Query Node Descendants</i>
-------------------	-------------------------------

Description

Find descendant nodes for specified variables in a DAG.

Usage

```
query_descendants(.tidy_dag, .var = NULL)
```

Arguments

.tidy_dag	A tidy_dagitty or dagitty object
.var	Character vector of variables to query. If NULL, returns parents for all nodes.

Value

A tibble with columns:

- node: The node
- descendant_set: String representation of descendant nodes
- descendants: List column containing descendant nodes
- n_descendants: Number of descendants

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w
)

query_descendants(dag)
query_descendants(dag, .var = "w")
```

query_dseparated	<i>Query D-separation</i>
------------------	---------------------------

Description

Test whether sets of variables are d-separated in a DAG given a conditioning set.

Usage

```
query_dseparated(.tdy_dag, from, to, conditioned_on = NULL)
```

Arguments

.tdy_dag	A tidy_dagitty or dagitty object
from	Character vector of node names.
to	Character vector of node names.
conditioned_on	Character vector of conditioning variables.

Value

A tibble with columns:

- from_set: String representation of source nodes
- from: List column of source nodes
- to_set: String representation of target nodes
- to: List column of target nodes
- conditioning_set: String representation of conditioning variables
- conditioned_on: List column of conditioning variables
- dseparated: Logical indicating d-separation

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w,
  z ~ w
)

query_dseparated(dag, from = "x", to = "z")
query_dseparated(dag, from = "x", to = "z", conditioned_on = "w")
```

query_exogenous	<i>Query Exogenous Variables</i>
-----------------	----------------------------------

Description

Identify exogenous (parentless) variables in a DAG.

Usage

```
query_exogenous(.tdy_dag)
```

Arguments

.tdy_dag A tidy_dagitty or dagitty object

Value

A tibble with columns:

- node: The exogenous variable
- n_descendants: Number of descendant nodes

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w
)

query_exogenous(dag)
```

query_instrumental	<i>Query Instrumental Variables</i>
--------------------	-------------------------------------

Description

Identify instrumental variables for a given exposure-outcome pair.

Usage

```
query_instrumental(
  .tdy_dag,
  exposure = NULL,
  outcome = NULL,
  conditioned_on = NULL
)
```

Arguments

.tdy_dag	A tidy_dagitty or dagitty object
exposure	Character vector of exposure variable names. If NULL, uses the exposure defined in the DAG.
outcome	Character vector of outcome variable names. If NULL, uses the outcome defined in the DAG.
conditioned_on	Defunct. dagitty::instrumentalVariables() works out the conditioning set an instrument requires on its own and reports it in the conditioning_set and conditioned_on columns, so there is nothing for this argument to do. Supplying it is an error.

Value

A tibble with columns:

- instrument: The instrumental variable
- exposure: The exposure variable
- outcome: The outcome variable
- conditioning_set: String representation of conditioning variables
- conditioned_on: List column of required conditioning variables

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + u,
  x ~ z + u,
  exposure = "x",
  outcome = "y",
  latent = "u"
)

query_instrumental(dag)
```

query_markov_blanket *Query Markov Blanket*

Description

Find the Markov blanket for specified variables in a DAG. The Markov blanket includes parents, children, and parents of children (co-parents).

Usage

```
query_markov_blanket(.tdy_dag, .var = NULL)
```

Arguments

`.tidy_dag` A `tidy_dagitty` or `dagitty` object
`.var` Character vector of variables to query. If NULL, returns parents for all nodes.

Value

A tibble with columns:

- `node`: The node
- `blanket`: String representation of Markov blanket nodes
- `blanket_vars`: List column containing Markov blanket nodes
- `blanket_size`: Size of the Markov blanket

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w,
  z ~ w
)

query_markov_blanket(dag)
query_markov_blanket(dag, .var = "x")
```

query_parents

Query Node Parents

Description

Find parent nodes for specified variables in a DAG.

Usage

```
query_parents(.tidy_dag, .var = NULL)
```

Arguments

`.tidy_dag` A `tidy_dagitty` or `dagitty` object
`.var` Character vector of variables to query. If NULL, returns parents for all nodes.

Value

A tibble with columns:

- `node`: The node
- `parent_set`: String representation of parent nodes
- `parents`: List column containing parent nodes
- `n_parents`: Number of parents

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w
)

query_parents(dag)
query_parents(dag, .var = "y")
```

query_paths	<i>Query Paths in a DAG</i>
-------------	-----------------------------

Description

Find all paths between specified nodes in a DAG and determine if they are open or closed given a conditioning set.

Usage

```
query_paths(
  .tidy_dag,
  from = NULL,
  to = NULL,
  directed = FALSE,
  limit = 100,
  conditioned_on = NULL
)
```

Arguments

.tidy_dag	A tidy_dagitty or dagitty object
from	Character vector of starting nodes. If NULL, uses exposure from DAG.
to	Character vector of ending nodes. If NULL, uses outcome from DAG.
directed	Logical. If TRUE, only considers directed paths.
limit	Maximum number of paths to return. Default is 100.
conditioned_on	Character vector of variables to condition on.

Details

When from or to has more than one element, paths are enumerated once for each ordered pair of endpoints and the results are stacked, so every row names the pair its path runs between.

Value

A tibble with columns:

- path_id: Integer identifier for each path
- from: Starting node
- to: Ending node
- path: Character string representation of the path
- path_type: Character classification as "direct" (a directed causal path), "backdoor" (a path whose first edge points into from), or "other" (any other path, such as one through a collider)
- variables: List column containing all variables in the path
- open: Logical indicating if the path is open

Examples

```
library(ggdag)
dag <- dagify(
  y ~ x + z,
  x ~ w,
  z ~ w,
  exposure = "x",
  outcome = "y"
)

query_paths(dag)
query_paths(dag, conditioned_on = "z")
```

query_status

Query Variable Status

Description

Query the status of variables in a DAG (exposure, outcome, or latent).

Usage

```
query_status(.tdy_dag, .var = NULL)
```

Arguments

.tdy_dag A tidy_dagitty or dagitty object
 .var Character vector of variables to query. If NULL, returns status for all nodes.

Value

A tibble with columns:

- name: The variable name
- status: The variable status (exposure, outcome, latent, or NA)

Examples

```
library(ggdag)
dag <- dagify(
  l ~ x + y,
  y ~ x,
  exposure = "x",
  outcome = "y",
  latent = "l"
)

query_status(dag)
query_status(dag, .var = "x")
```

Quick Plots for Common DAGs

Quickly create a DAGs with common structures of bias

Description

base functions create an object of class dagitty; ggdag_* functions are wrappers that also call ggdag() on the dagitty object.

Usage

```
m_bias(
  x = NULL,
  y = NULL,
  a = NULL,
  b = NULL,
  m = NULL,
  x_y_associated = FALSE
)
```

```
butterfly_bias(
  x = NULL,
  y = NULL,
  a = NULL,
  b = NULL,
  m = NULL,
  x_y_associated = FALSE
)
```

```
confounder_triangle(x = NULL, y = NULL, z = NULL, x_y_associated = FALSE)
```

```
collider_triangle(x = NULL, y = NULL, m = NULL, x_y_associated = FALSE)
```

```
mediation_triangle(x = NULL, y = NULL, m = NULL, x_y_associated = FALSE)
```

```

quartet_collider(x = NULL, y = NULL, z = NULL, x_y_associated = TRUE)

quartet_confounder(x = NULL, y = NULL, z = NULL, x_y_associated = TRUE)

quartet_mediator(x = NULL, y = NULL, z = NULL, x_y_associated = FALSE)

quartet_m_bias(
  x = NULL,
  y = NULL,
  z = NULL,
  u1 = NULL,
  u2 = NULL,
  x_y_associated = TRUE
)

quartet_time_collider(
  x0 = NULL,
  x1 = NULL,
  x2 = NULL,
  x3 = NULL,
  y1 = NULL,
  y2 = NULL,
  y3 = NULL,
  z1 = NULL,
  z2 = NULL,
  z3 = NULL
)

ggdag_m_bias(
  x = NULL,
  y = NULL,
  a = NULL,
  b = NULL,
  m = NULL,
  x_y_associated = FALSE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),

```

```

    use_stylized = ggdag_option("use_stylized", FALSE),
    use_text = ggdag_option("use_text", TRUE),
    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    edge_engine = ggdag_option("edge_engine", "ggraph"),
    unified_legend = TRUE,
    key_glyph = NULL,
    text = NULL,
    label = NULL,
    node = deprecated(),
    stylized = deprecated()
)

```

```

ggdag_butterfly_bias(
  x = NULL,
  y = NULL,
  a = NULL,
  b = NULL,
  m = NULL,
  x_y_associated = FALSE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

```

ggdag_confounder_triangle(
  x = NULL,
  y = NULL,

```

```

z = NULL,
x_y_associated = FALSE,
size = 1,
edge_type = ggdag_option("edge_type", "link_arc"),
node_size = ggdag_option("node_size", 16),
text_size = ggdag_option("text_size", 3.88),
label_size = ggdag_option("label_size", text_size),
text_col = ggdag_option("text_col", "white"),
label_col = ggdag_option("label_col", "black"),
edge_width = ggdag_option("edge_width", 0.6),
edge_cap = ggdag_option("edge_cap", 8),
arrow_length = ggdag_option("arrow_length", 5),
use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
edge_engine = ggdag_option("edge_engine", "ggraph"),
unified_legend = TRUE,
key_glyph = NULL,
text = NULL,
label = NULL,
node = deprecated(),
stylized = deprecated()
)

ggdag_collider_triangle(
  x = NULL,
  y = NULL,
  m = NULL,
  x_y_associated = FALSE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),

```

```

    edge_engine = ggdag_option("edge_engine", "ggraph"),
    unified_legend = TRUE,
    key_glyph = NULL,
    text = NULL,
    label = NULL,
    node = deprecated(),
    stylized = deprecated()
)

ggdag_mediation_triangle(
  x = NULL,
  y = NULL,
  m = NULL,
  x_y_associated = FALSE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

ggdag_quartet_collider(
  x = NULL,
  y = NULL,
  z = NULL,
  x_y_associated = TRUE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),

```

```

    label_size = ggdag_option("label_size", text_size),
    text_col = ggdag_option("text_col", "white"),
    label_col = ggdag_option("label_col", "black"),
    edge_width = ggdag_option("edge_width", 0.6),
    edge_cap = ggdag_option("edge_cap", 8),
    arrow_length = ggdag_option("arrow_length", 5),
    use_edges = ggdag_option("use_edges", TRUE),
    use_nodes = ggdag_option("use_nodes", TRUE),
    use_stylized = ggdag_option("use_stylized", FALSE),
    use_text = ggdag_option("use_text", TRUE),
    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    edge_engine = ggdag_option("edge_engine", "ggraph"),
    unified_legend = TRUE,
    key_glyph = NULL,
    text = NULL,
    label = NULL
  )

ggdag_quartet_confounder(
  x = NULL,
  y = NULL,
  z = NULL,
  x_y_associated = TRUE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL
)

ggdag_quartet_mediator(

```

```

x = NULL,
y = NULL,
z = NULL,
x_y_associated = FALSE,
size = 1,
edge_type = ggdag_option("edge_type", "link_arc"),
node_size = ggdag_option("node_size", 16),
text_size = ggdag_option("text_size", 3.88),
label_size = ggdag_option("label_size", text_size),
text_col = ggdag_option("text_col", "white"),
label_col = ggdag_option("label_col", "black"),
edge_width = ggdag_option("edge_width", 0.6),
edge_cap = ggdag_option("edge_cap", 8),
arrow_length = ggdag_option("arrow_length", 5),
use_edges = ggdag_option("use_edges", TRUE),
use_nodes = ggdag_option("use_nodes", TRUE),
use_stylized = ggdag_option("use_stylized", FALSE),
use_text = ggdag_option("use_text", TRUE),
use_labels = ggdag_option("use_labels", FALSE),
label_geom = ggdag_option("label_geom", geom_dag_label_repel),
edge_engine = ggdag_option("edge_engine", "ggraph"),
unified_legend = TRUE,
key_glyph = NULL,
text = NULL,
label = NULL
)

```

```

ggdag_quartet_m_bias(
  x = NULL,
  y = NULL,
  z = NULL,
  u1 = NULL,
  u2 = NULL,
  x_y_associated = TRUE,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),

```

```

    use_labels = ggdag_option("use_labels", FALSE),
    label_geom = ggdag_option("label_geom", geom_dag_label_repel),
    edge_engine = ggdag_option("edge_engine", "ggraph"),
    unified_legend = TRUE,
    key_glyph = NULL,
    text = NULL,
    label = NULL
  )

ggdag_quartet_time_collider(
  x0 = NULL,
  x1 = NULL,
  x2 = NULL,
  x3 = NULL,
  y1 = NULL,
  y2 = NULL,
  y3 = NULL,
  z1 = NULL,
  z2 = NULL,
  z3 = NULL,
  size = 1,
  edge_type = ggdag_option("edge_type", "link_arc"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  edge_engine = ggdag_option("edge_engine", "ggraph"),
  unified_legend = TRUE,
  key_glyph = NULL,
  text = NULL,
  label = NULL
)

```

Arguments

- x, y, a, b, m, z** Character vector. Optional label. Default is NULL
- x_y_associated** Logical. Are x and y associated? Default is FALSE for the bias-structure DAGs and for `quartet_mediator()`, and TRUE for `quartet_collider()`, `quartet_confounder()`,

	and <code>quartet_m_bias()</code> , whose datasets have <code>x</code> and <code>y</code> associated by construction. <code>quartet_time_collider()</code> has no such argument.
<code>u1, u2</code>	Character vector. Optional label for unmeasured nodes, used in <code>quartet_m_bias()</code> . Default is <code>NULL</code> .
<code>x0, x1, x2, x3, y1, y2, y3, z1, z2, z3</code>	Character vector. Optional labels for time-indexed nodes, used in <code>quartet_time_collider()</code> . Default is <code>NULL</code> . The DAG holds six nodes, <code>x1</code> , <code>x2</code> , <code>y2</code> , <code>y3</code> , <code>z2</code> , and <code>z3</code> ; <code>x0</code> , <code>x3</code> , <code>y1</code> , and <code>z1</code> name none of them and are accepted only so that code written against the older signature still runs. Labels given for them are ignored.
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If <code>TRUE</code> , which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>edge_engine</code>	The engine used to draw edges. Either "ggraph" (default) or "ggarrow". When "ggarrow", edges are drawn using ggarrow geoms, which support additional customization via the <code>arrow_head</code> , <code>arrow_fins</code> , <code>arrow_mid</code> , and <code>curvature</code> global options (see ggdag_options_set()).
<code>unified_legend</code>	A logical value. When <code>TRUE</code> and both <code>use_edges</code> and <code>use_nodes</code> are <code>TRUE</code> , creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is <code>TRUE</code> .
<code>key_glyph</code>	A function to use for drawing the legend key glyph for nodes. If <code>NULL</code> (the default), the glyph is chosen automatically based on the <code>unified_legend</code> setting. When provided, this overrides the automatic selection. Common options include <code>draw_key_dag_point</code> , <code>draw_key_dag_combined</code> , and <code>draw_key_dag_collider</code> .

text	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use name.
label	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use label.
node	Deprecated.
stylized	Deprecated.

Details

The `quartet_*` functions create DAGs that represent the causal quartet, which are four example datasets with identical statistical properties but different causal structures. These are inspired by Anscombe's quartet and demonstrate that statistical summaries alone cannot determine causal relationships. See [Causal Inference in R](#)

The four structures represent different relationships between exposure (x), outcome (y), and a co-variate (z):

- Collider: z is caused by both x and y (should not adjust for z)
- Confounder: z causes both x and y (must adjust for z)
- Mediator: z is on the causal path from x to y (adjust for direct effect only)
- M-bias: z is a collider with unmeasured confounders u1 and u2 (should not adjust for z)

The time-varying collider (`quartet_time_collider()`) demonstrates how time-ordering can help identify causal relationships when variables are measured at multiple time points.

Value

a DAG of class `dagitty` or a `ggplot`

References

D'Agostino McGowan L, Gerke T, Barrett M (2023). "Causal inference is not just a statistics problem." *Journal of Statistics and Data Science Education*, 32(1), 1-4. doi:[10.1080/26939169.2023.2276446](https://doi.org/10.1080/26939169.2023.2276446)

Examples

```
m_bias() |> ggdag_adjust("m")
ggdag_confounder_triangle()

# Causal Quartets
ggdag_quartet_collider()
ggdag_quartet_confounder()
ggdag_quartet_mediator()
ggdag_quartet_m_bias()

# Time-varying collider
ggdag_quartet_time_collider()
```

remove_axes

*Quickly remove plot axes and grids***Description**

`remove_axes()` and `remove_grid()` are convenience functions that removes the axes and grids from a ggplot, respectively. This is useful when you want to use an existing theme, e.g. those included in `ggplot2`, for a DAG.

Usage

```
remove_axes()
```

```
remove_grid()
```

Examples

```
library(ggplot2)
ggdag(confounder_triangle()) +
  theme_bw() +
  remove_axes()
```

scale_adjusted

*Common scale adjustments for DAGs***Description**

`scale_adjusted()` is a convenience function that implements ways of visualizing adjustment for a variable. By convention, a square shape is used to indicate adjustment and a circle when not adjusted. Arrows out of adjusted variables are often eliminated or de-emphasized, and `scale_adjusted()` uses a lower alpha for these arrows. When adjusting a collider, a dashed line is sometimes used to demarcate opened pathways, and `scale_adjusted()` does this whenever `geom_dag_collider_edges()` is used. `scale_dag()` is deprecated in favor of `scale_adjusted()`.

Usage

```
scale_adjusted(
  include_linetype = TRUE,
  include_shape = TRUE,
  include_color = TRUE,
  include_alpha = FALSE
)
```

```
scale_dag(breaks = ggplot2::waiver())
```

Arguments

include_linetype	Logical. Include linetype scale for dashed lines on collider edges? Default is TRUE.
include_shape	Logical. Include shape scale for adjustment status (squares for adjusted, circles for unadjusted)? Default is TRUE.
include_color	Logical. Include color scale for adjustment status? Default is TRUE.
include_alpha	Logical. Include alpha scales for de-emphasizing edges from adjusted variables? Default is FALSE.
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

set_curve_edges	<i>Set curvature for multiple edges at once</i>
-----------------	---

Description

set_curve_edges() replaces all edge curvatures on a dagitty or tidy_dagitty object from a data frame. Use [curve_edge\(\)](#) to set a single edge.

Usage

```
set_curve_edges(.dag, edges)
```

Arguments

.dag	A dagitty or tidy_dagitty object.
edges	A data frame with columns from, to, and curvature.

Value

The modified .dag object with updated curvatures.

Curvature sign convention

Per-edge curvature is drawn by the **ggarrow** edge engine only. Set it with edge_engine = "ggarrow" on [geom_dag\(\)](#) and the ggdag_*() quick plots, or globally with ggdag_options_set(edge_engine = "ggarrow"); the default **ggraph** engine draws every edge with the curvature of its own edge type and warns when it is handed a per-edge value it cannot draw.

ggarrow follows the grid::curveGrob() convention, so positive curvature curves *below* (to the right of) a left-to-right edge and negative curvature curves *above* it. ggdag passes the value through untouched.

A bidirected edge has no direction of its own, so ggdag draws it from the endpoint dagitty stores first. Naming its endpoints the other way round, as in `curve_edge(dag, "y", "x", 0.5)` for an edge stored as $x \leftrightarrow y$, curves the edge to the same side of the page: the sign is flipped to match the direction the edge is drawn in.

Examples

```
dag <- dagify(y ~ x + m, m ~ x)
edges <- data.frame(
  from = c("x", "m"),
  to = c("y", "y"),
  curvature = c(0.3, -0.4)
)
dag <- set_curve_edges(dag, edges)
```

simulate_data

Simulate Data from Structural Equation Model

Description

This is a thin wrapper for the `simulateSEM()` function in dagitty that works with tidied dagitty objects. It treats the input DAG as a structural equation model, generating random path coefficients and simulating corresponding data. See [dagitty::simulateSEM\(\)](#) for details.

Usage

```
simulate_data(
  .tdy_dag,
  b.default = NULL,
  b.lower = -0.6,
  b.upper = 0.6,
  eps = 1,
  N = 500,
  standardized = TRUE
)
```

Arguments

<code>.tdy_dag</code>	A tidy_dagitty or dagitty object
<code>b.default</code>	default path coefficient applied to arrows for which no coefficient is defined in the model syntax.
<code>b.lower</code>	lower bound for random path coefficients, applied if <code>b.default = NULL</code> .
<code>b.upper</code>	upper bound for path coefficients.
<code>eps</code>	residual variance (only meaningful if <code>standardized=FALSE</code>).
<code>N</code>	number of samples to generate.
<code>standardized</code>	whether a standardized output is desired (all variables have variance 1).

Value

a tbl with N values for each variable in .tidy_dag

Examples

```
dagify(y ~ z, x ~ z) |>
  tidy_dagitty() |>
  simulate_data()
```

<code>tbl_df.tidy_dagitty</code>	<i>Convert a tidy_dagitty object to tbl_df</i>
----------------------------------	--

Description

Convert a tidy_dagitty object to tbl_df

Usage

```
tbl_df.tidy_dagitty(.tidy_dag)
```

Arguments

.tidy_dag A tidy_dagitty or dagitty object

Test if Variable Is Collider
<i>Detecting colliders in DAGs</i>

Description

A collider is a variable that two edges point into. Bidirected edges count, so a variable with one directed parent and one bidirected partner is a collider, as is a variable with two bidirected partners and no parents.

Usage

```
is_collider(.dag, .var, downstream = TRUE)

is_downstream_collider(.dag, .var)
```

Arguments

.dag an input graph, an object of class tidy_dagitty or dagitty
.var a character vector of length 1, the potential collider to check
downstream Logical. Check for downstream colliders? Default is TRUE.

Value

Logical. Is the variable a collider or downstream collider?

Examples

```
dag <- dagify(m ~ x + y, m_jr ~ m)
is_collider(dag, "m")
is_downstream_collider(dag, "m_jr")

# a downstream collider is also treated as a collider
is_collider(dag, "m_jr")

# but a direct collider is not treated as a downstream collider
is_downstream_collider(dag, "m")
```

theme_dag_blank	<i>Minimalist DAG themes</i>
-----------------	------------------------------

Description

Minimalist DAG themes

Usage

```
theme_dag_blank(base_size = 12, base_family = "", ...)
```

```
theme_dag(base_size = 12, base_family = "", ...)
```

```
theme_dag_grid(base_size = 12, base_family = "", ...)
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
...	additional arguments passed to theme(). A value given here replaces the theme's own setting for that element. complete is the one exception: the result takes its completeness from the base theme, so a complete passed here has no effect.

Examples

```
ggdag(m_bias()) + theme_dag_blank() # the default
```

theme_dag_grey	<i>Simple grey themes for DAGs</i>
----------------	------------------------------------

Description

Simple grey themes for DAGs

Usage

```
theme_dag_grey(base_size = 12, base_family = "", ...)
```

```
theme_dag_gray(base_size = 12, base_family = "", ...)
```

```
theme_dag_grey_grid(base_size = 12, base_family = "", ...)
```

```
theme_dag_gray_grid(base_size = 12, base_family = "", ...)
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
...	additional arguments passed to theme(). A value given here replaces the theme's own setting for that element. complete is the one exception: the result takes its completeness from the base theme, so a complete passed here has no effect.

Examples

```
ggdag(m_bias()) + theme_dag_grey()
```

tidy_dagitty	<i>Tidy a dagitty object</i>
--------------	------------------------------

Description

Tidy a dagitty object

Usage

```
tidy_dagitty(
  .dagitty,
  seed = NULL,
  layout = ggdag_option("layout", "nicely"),
  ...,
  use_existing_coords = TRUE
)
```

Arguments

<code>.dagitty</code>	a dagitty
<code>seed</code>	a numeric seed for reproducible layout generation
<code>layout</code>	a layout available in ggraph. See <code>ggraph::create_layout()</code> for details. Alternatively, "time_ordered" will use <code>time_ordered_coords()</code> to algorithmically sort the graph by time. You can also pass the result of <code>time_ordered_coords()</code> directly: either the function returned when called with no arguments, or the coordinate tibble returned when called with arguments.
<code>...</code>	optional arguments passed to <code>ggraph::create_layout()</code>
<code>use_existing_coords</code>	(Advanced). Logical. Use the coordinates produced by <code>dagitty::coordinates(.dagitty)</code> ? If the coordinates are empty, <code>tidy_dagitty()</code> will generate a layout. Generally, setting this to FALSE is thus only useful when there is a difference in the variables coordinates and the variables in the DAG, as sometimes happens when recompiling a DAG.

Value

a tidy_dagitty object

Examples

```
library(dagitty)
library(ggplot2)

dag <- dagitty("dag {
  Y <- X <- Z1 <- V -> Z2 -> Y
  Z1 <- W1 <-> W2 -> Z2
  X <- W1 -> Y
  X <- W2 -> Y
  X [exposure]
  Y [outcome]
}")

tidy_dagitty(dag)

tidy_dagitty(dag, layout = "fr") |>
  ggplot(aes(x = .data$x, y = .data$y, xend = .data$xend, yend = .data$yend)) +
  geom_dag_node() +
  geom_dag_text() +
  geom_dag_edges() +
  theme_dag()
```

time_ordered_coords *Create a time-ordered coordinate data frame*

Description

time_ordered_coords() is a helper function to create time-ordered DAGs. Pass the results to the coords argument of dagify(). If .vars is not specified, these coordinates will be determined automatically. If you want to be specific, you can also use a list or data frame. The default is to assume you want variables to go from left to right in order by time. Variables are spread along the y-axis using a simple algorithm to stack them. You can also work along the y-axis by setting direction = "y".

Usage

```
time_ordered_coords(
  .vars = NULL,
  time_points = NULL,
  direction = c("x", "y"),
  auto_sort_direction = c("right", "left"),
  fixed_time = NULL,
  adjust_exposure_outcome = TRUE,
  force_y = TRUE
)
```

Arguments

- | | |
|---------------------|---|
| .vars | A list of character vectors, where each vector represents a single time period. Alternatively, a data frame where the first column is the variable name and the second column is the time period. |
| time_points | A vector of time points, one element per time period in .vars. Default is NULL, which creates a sequence from 1 to the number of time periods (the length of .vars). A data frame carries its own time points in its second column, so supplying both is an error. |
| direction | A character string indicating the axis along which the variables should be time-ordered. Either "x" or "y". Default is "x". |
| auto_sort_direction | If .vars is NULL: nodes will be placed as far "left" or "right" of in the graph as is reasonable. Default is right, meaning the nodes will be as close as possible in time to their descendants. |
| fixed_time | A named numeric vector pinning specific nodes to time points (e.g., c(x = 3, z = 1)). Only used in auto mode (.vars = NULL). Other nodes are placed automatically while respecting these constraints. Pinned times are 1-based and preserved in the output, so they must be whole numbers of at least 1. A pin that no ordering can satisfy, such as a time earlier than the node's own ancestors allow, is an error. |

adjust_exposure_outcome	If TRUE (default), automatically shift the outcome forward by one time point when it shares a layer with the exposure. All descendants of the outcome are also shifted. Only applies in auto mode and when the DAG has exposure and outcome set.
force_y	If TRUE (default), run force-directed Y optimization to minimize node-edge overlaps. If FALSE, nodes are evenly spaced within each layer using barycenter ordering only. Setting to FALSE is useful when edges will be curved or auto-routed, where tight Y positioning is less important. Only used in auto mode (<code>.vars = NULL</code>).

Value

A tibble with three columns: name, x, and y.

See Also

[dagify\(\)](#), [coords2df\(\)](#), [coords2list\(\)](#)

Examples

```
dagify(
  d ~ c1 + c2 + c3,
  c1 ~ b1 + b2,
  c3 ~ a,
  b1 ~ a,
  coords = time_ordered_coords()
) |> ggdag()

coords <- time_ordered_coords(list(
  # time point 1
  "a",
  # time point 2
  c("b1", "b2"),
  # time point 3
  c("c1", "c2", "c3"),
  # time point 4
  "d"
))

dagify(
  d ~ c1 + c2 + c3,
  c1 ~ b1 + b2,
  c3 ~ a,
  b1 ~ a,
  coords = coords
) |> ggdag()

# or use a data frame
x <- data.frame(
  name = c("x1", "x2", "y", "z1", "z2", "z3", "a"),
  time = c(1, 1, 2, 3, 3, 3, 4)
```

```

)
dagify(
  z3 ~ y,
  y ~ x1 + x2,
  a ~ z1 + z2 + z3,
  coords = time_ordered_coords(x)
) |>
ggdag()

```

Variable Status

Find variable status

Description

Detects variable status given a DAG (exposure, outcome, latent). See [dagitty::VariableStatus\(\)](#) for details.

Usage

```
node_status(.dag, as_factor = TRUE, ...)
```

```

ggdag_status(
  .tdy_dag,
  ...,
  size = 1,
  edge_type = c("link_arc", "link", "arc", "diagonal"),
  node_size = ggdag_option("node_size", 16),
  text_size = ggdag_option("text_size", 3.88),
  label_size = ggdag_option("label_size", text_size),
  text_col = ggdag_option("text_col", "white"),
  label_col = ggdag_option("label_col", "black"),
  edge_width = ggdag_option("edge_width", 0.6),
  edge_cap = ggdag_option("edge_cap", 8),
  arrow_length = ggdag_option("arrow_length", 5),
  use_edges = ggdag_option("use_edges", TRUE),
  use_nodes = ggdag_option("use_nodes", TRUE),
  use_stylized = ggdag_option("use_stylized", FALSE),
  use_text = ggdag_option("use_text", TRUE),
  use_labels = ggdag_option("use_labels", FALSE),
  label_geom = ggdag_option("label_geom", geom_dag_label_repel),
  unified_legend = TRUE,
  text = NULL,
  label = NULL,
  node = deprecated(),
  stylized = deprecated()
)

```

Arguments

<code>.dag, .tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>as_factor</code>	treat status variable as factor
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>size</code>	A numeric value scaling the size of all elements in the DAG. This allows you to change the scale of the DAG without changing the proportions.
<code>edge_type</code>	The type of edge, one of "link_arc", "link", "arc", "diagonal".
<code>node_size</code>	The size of the nodes.
<code>text_size</code>	The size of the text.
<code>label_size</code>	The size of the labels.
<code>text_col</code>	The color of the text.
<code>label_col</code>	The color of the labels.
<code>edge_width</code>	The width of the edges.
<code>edge_cap</code>	The size of edge caps (the distance between the arrowheads and the node borders).
<code>arrow_length</code>	The length of arrows on edges.
<code>use_edges</code>	A logical value. Include a <code>geom_dag_edges*()</code> function? If TRUE, which is determined by <code>edge_type</code> .
<code>use_nodes</code>	A logical value. Include <code>geom_dag_point()</code> ?
<code>use_stylized</code>	A logical value. Include <code>geom_dag_node()</code> ?
<code>use_text</code>	A logical value. Include <code>geom_dag_text()</code> ?
<code>use_labels</code>	A logical value. Include a label geom? The specific geom used is controlled by <code>label_geom</code> .
<code>label_geom</code>	A geom function to use for drawing labels when <code>use_labels = TRUE</code> . Default is <code>geom_dag_label_repel</code> . Other options include <code>geom_dag_label</code> , <code>geom_dag_text_repel</code> , <code>geom_dag_label_repel2</code> , and <code>geom_dag_text_repel2</code> .
<code>unified_legend</code>	A logical value. When TRUE and both <code>use_edges</code> and <code>use_nodes</code> are TRUE, creates a unified legend entry showing both nodes and edges in a single key, and hides the separate edge legend. This creates a single, more compact legend. Default is TRUE.
<code>text</code>	The bare name of a column to use for <code>geom_dag_text()</code> . If <code>use_text = TRUE</code> , the default is to use <code>name</code> .
<code>label</code>	The bare name of a column to use for labels. If <code>use_labels = TRUE</code> , the default is to use <code>label</code> .
<code>node</code>	Deprecated.
<code>stylized</code>	Deprecated.

Details

`node_status` tags variable status and `ggdag_status` plots all variable statuses.

Value

a tidy_dagitty with a status column for variable status or a ggplot

Examples

```
dag <- dagify(  
  l ~ x + y,  
  y ~ x,  
  exposure = "x",  
  outcome = "y",  
  latent = "l"  
)  
  
node_status(dag)  
ggdag_status(dag)
```

Index

activate_collider_paths, 4
Adjust for variables, 5
adjust_for (Adjust for variables), 5
aes, 79
aes(), 61, 66, 69, 91
aes_, 79
aes_dag, 7
annotation_borders(), 63, 68, 71, 92
anti_join.tidy_dagitty (dplyr), 41
arrange.tidy_dagitty (dplyr), 41
arrange_.tidy_dagitty (dplyr), 41
arrow, 80
as.data.frame.tidy_dagitty, 8
as.tbl.tidy_dagitty, 9
as_tbl_graph, 9
as_tibble.tidy_dagitty
 (as.tbl.tidy_dagitty), 9
as_tidy_dagitty, 10
as_tidy_dagitty.list(), 38, 39
Assess d-separation between variables,
 12
Assess familial relationships between
 variables, 17

borders, 81
butterfly_bias (Quick Plots for Common
 DAGs), 113

Canonicalize DAGs, 23
collider_triangle (Quick Plots for
 Common DAGs), 113
Colliders, 25
confounder_triangle (Quick Plots for
 Common DAGs), 113
control_for (Adjust for variables), 5
control_for(), 4
coordinates, 27
coords2df (coordinates), 27
coords2df(), 40, 131
coords2list (coordinates), 27

coords2list(), 40, 131
Covariate Adjustment Sets, 28
curve_edge, 30
curve_edge(), 124
curved, 31

dag, 32
DAG Edges, 33
DAG Labels, 37
dag(), 40
dag_adjustment_sets (Covariate
 Adjustment Sets), 28
dag_label (DAG Labels), 37
dag_paths (Pathways), 93
dag_prune (dag_saturate), 38
dag_saturate, 38
dagify, 40
dagify(), 31, 32, 131
dagitty, 32
dagitty::adjustmentSets(), 28
dagitty::canonicalize(), 23
dagitty::children, 17
dagitty::dseparated(), 12
dagitty::equivalenceClass(), 47
dagitty::equivalentDAGs(), 47
dagitty::exogenousVariables(), 50
dagitty::instrumentalVariables(), 84
dagitty::paths(), 93
dagitty::simulateSEM(), 125
dagitty::VariableStatus(), 132
distinct.tidy_dagitty (dplyr), 41
dplyr, 41
dplyr::as_tibble(), 9
dplyr::n(), 31
draw_key_dag_collider, 43
draw_key_dag_combined, 44
draw_key_dag_edge, 44
draw_key_dag_point, 45

edge_backdoor, 45

- Equivalent DAGs and Classes, [47](#)
- Exogenous Variables, [50](#)
- expand_plot, [52](#)
- filter.tidy_dagitty(dplyr), [41](#)
- filter_.tidy_dagitty(dplyr), [41](#)
- fortify, [53](#)
- fortify(), [54](#), [61](#), [66](#), [69](#), [77](#), [91](#)
- full_join.tidy_dagitty(dplyr), [41](#)
- geom_dag, [53](#)
- geom_dag(), [31](#), [32](#), [124](#)
- geom_dag_arrow, [56](#)
- geom_dag_arrow_arc(geom_dag_arrow), [56](#)
- geom_dag_arrows(geom_dag_arrow), [56](#)
- geom_dag_collider_edges, [61](#)
- geom_dag_collider_edges(), [4](#), [123](#)
- geom_dag_edges, [63](#)
- geom_dag_edges_arc(DAG Edges), [33](#)
- geom_dag_edges_arc(), [82](#)
- geom_dag_edges_diagonal(DAG Edges), [33](#)
- geom_dag_edges_diagonal(), [82](#)
- geom_dag_edges_fan(DAG Edges), [33](#)
- geom_dag_edges_fan(), [82](#)
- geom_dag_edges_link(DAG Edges), [33](#)
- geom_dag_label, [66](#)
- geom_dag_label_repel(ggrepel functions), [77](#)
- geom_dag_label_repel(), [55](#), [73](#), [76](#)
- geom_dag_label_repel2(ggrepel functions), [77](#)
- geom_dag_node(Nodes), [90](#)
- geom_dag_point(Nodes), [90](#)
- geom_dag_text, [69](#)
- geom_dag_text_repel(ggrepel functions), [77](#)
- geom_dag_text_repel2(ggrepel functions), [77](#)
- ggarrow, [6](#), [29](#), [49](#), [54](#), [73](#), [96](#), [121](#)
- ggdag, [72](#)
- ggdag(), [75](#)
- ggdag_adjacent(Assess familial relationships between variables), [17](#)
- ggdag_adjust(Adjust for variables), [5](#)
- ggdag_adjust(), [4](#)
- ggdag_adjustment_set(Covariate Adjustment Sets), [28](#)
- ggdag_adjustment_set(), [76](#)
- ggdag_ancestors(Assess familial relationships between variables), [17](#)
- ggdag_butterfly_bias(Quick Plots for Common DAGs), [113](#)
- ggdag_canonical(Canonicalize DAGs), [23](#)
- ggdag_children(Assess familial relationships between variables), [17](#)
- ggdag_classic, [74](#)
- ggdag_classic(), [74](#)
- ggdag_collider(Colliders), [25](#)
- ggdag_collider_triangle(Quick Plots for Common DAGs), [113](#)
- ggdag_conditional_independence(query_conditional_independence), [103](#)
- ggdag_confounder_triangle(Quick Plots for Common DAGs), [113](#)
- ggdag_dconnected(Assess d-separation between variables), [12](#)
- ggdag_defaults, [75](#)
- ggdag_descendants(Assess familial relationships between variables), [17](#)
- ggdag_drelationship(Assess d-separation between variables), [12](#)
- ggdag_drelationship(), [76](#)
- ggdag_dseparated(Assess d-separation between variables), [12](#)
- ggdag_equivalent_class(Equivalent DAGs and Classes), [47](#)
- ggdag_equivalent_dags(Equivalent DAGs and Classes), [47](#)
- ggdag_exogenous(Exogenous Variables), [50](#)
- ggdag_instrumental(Instrumental Variables), [84](#)
- ggdag_m_bias(Quick Plots for Common DAGs), [113](#)
- ggdag_markov_blanket(Assess familial relationships between variables), [17](#)
- ggdag_mediation_triangle(Quick Plots for Common DAGs), [113](#)
- ggdag_option(ggdag_defaults), [75](#)
- ggdag_option_proportional

- (ggdag_defaults), 75
- ggdag_options_get (ggdag_defaults), 75
- ggdag_options_reset (ggdag_defaults), 75
- ggdag_options_set (ggdag_defaults), 75
- ggdag_options_set(), 6, 29, 49, 54, 73, 81, 96, 121
- ggdag_parents (Assess familial relationships between variables), 17
- ggdag_paths (Pathways), 93
- ggdag_paths_fan (Pathways), 93
- ggdag_quartet_collider (Quick Plots for Common DAGs), 113
- ggdag_quartet_confounder (Quick Plots for Common DAGs), 113
- ggdag_quartet_m_bias (Quick Plots for Common DAGs), 113
- ggdag_quartet_mediator (Quick Plots for Common DAGs), 113
- ggdag_quartet_time_collider (Quick Plots for Common DAGs), 113
- ggdag_status (Variable Status), 132
- ggplot(), 54, 61, 66, 69, 91
- ggplot.dagitty (ggplot.tidy_dagitty), 77
- ggplot.tidy_dagitty, 77
- ggplot2::aes(), 58
- ggplot2::geom_point, 90
- ggplot2::ggplot(), 58
- ggplot2::layer(), 59
- ggraph::create_layout(), 11, 38, 48, 129
- ggrepel functions, 77
- ggrepel::geom_label_repel(), 81
- ggrepel::geom_text_repel(), 80, 81
- grDevices::plotmath(), 35
- grid::arrow(), 63
- grid::unit(), 36, 58
- group_by.tidy_dagitty (dplyr), 41
- has_labels (DAG Labels), 37
- inner_join.tidy_dagitty (dplyr), 41
- Instrumental Variables, 84
- is.tidy_dagitty, 86
- is_acyclic (is_dag_properties), 87
- is_adjacent (is_node_relationships), 89
- is_adjustment_set (is_dag_properties), 87
- is_ancestor (is_node_relationships), 89
- is_child (is_node_relationships), 89
- is_collider (Test if Variable Is Collider), 126
- is_confounder, 86
- is_d_connected (is_dag_properties), 87
- is_d_separated (is_dag_properties), 87
- is_dag_properties, 87
- is_descendant (is_node_relationships), 89
- is_downstream_collider (Test if Variable Is Collider), 126
- is_exogenous (is_node_properties), 88
- is_exposure (is_node_properties), 88
- is_instrumental (is_node_properties), 88
- is_latent (is_node_properties), 88
- is_node_properties, 88
- is_node_relationships, 89
- is_outcome (is_node_properties), 88
- is_parent (is_node_relationships), 89
- key glyphs, 62, 67, 70, 92
- label (DAG Labels), 37
- label<- (DAG Labels), 37
- layer, 80
- layer position, 62, 67, 70, 91
- layer stat, 62, 67, 70
- layer(), 62, 67, 70, 92
- left_join.tidy_dagitty (dplyr), 41
- loess, 104
- m_bias (Quick Plots for Common DAGs), 113
- mediation_triangle (Quick Plots for Common DAGs), 113
- mutate.tidy_dagitty (dplyr), 41
- mutate_.tidy_dagitty (dplyr), 41
- node_adjacent (Assess familial relationships between variables), 17
- node_ancestors (Assess familial relationships between variables), 17
- node_canonical (Canonicalize DAGs), 23
- node_children (Assess familial relationships between variables), 17
- node_collider (Colliders), 25
- node_dconnected (Assess d-separation between variables), 12

- node_descendants (Assess familial relationships between variables), 17
- node_drelationship (Assess d-separation between variables), 12
- node_dseparated (Assess d-separation between variables), 12
- node_equivalent_class (Equivalent DAGs and Classes), 47
- node_equivalent_dags (Equivalent DAGs and Classes), 47
- node_exogenous (Exogenous Variables), 50
- node_instrumental (Instrumental Variables), 84
- node_markov_blanket (Assess familial relationships between variables), 17
- node_parents (Assess familial relationships between variables), 17
- node_status (Variable Status), 132
- Nodes, 90
- options(), 76
- Pathways, 93
- print.tidy_dagitty, 97
- pull_dag, 97
- pull_dag(), 11
- pull_dag.dagitty (pull_dag), 97
- pull_dag.tidy_dagitty (pull_dag), 97
- pull_dag_data (pull_dag), 97
- quartet_collider (Quick Plots for Common DAGs), 113
- quartet_confounder (Quick Plots for Common DAGs), 113
- quartet_m_bias (Quick Plots for Common DAGs), 113
- quartet_mediator (Quick Plots for Common DAGs), 113
- quartet_time_collider (Quick Plots for Common DAGs), 113
- query_adjustment_sets, 99
- query_ancestors, 100
- query_children, 101
- query_colliders, 102
- query_conditional_independence, 103
- query_dconnected, 105
- query_descendants, 106
- query_dseparated, 107
- query_exogenous, 108
- query_instrumental, 108
- query_markov_blanket, 109
- query_parents, 110
- query_paths, 111
- query_status, 112
- Quick Plots for Common DAGs, 113
- remove_axes, 123
- remove_grid (remove_axes), 123
- rename.tidy_dagitty (dplyr), 41
- right_join.tidy_dagitty (dplyr), 41
- scale_adjusted, 123
- scale_dag (scale_adjusted), 123
- select.tidy_dagitty (dplyr), 41
- select_.tidy_dagitty (dplyr), 41
- semi_join.tidy_dagitty (dplyr), 41
- set.seed, 87
- set_curve_edges, 124
- set_curve_edges(), 30
- simulate_data, 125
- slice.tidy_dagitty (dplyr), 41
- slice_.tidy_dagitty (dplyr), 41
- summarise.tidy_dagitty (dplyr), 41
- summarise_.tidy_dagitty (dplyr), 41
- tbl_df.tidy_dagitty, 126
- Test if Variable Is Collider, 126
- test_conditional_independence (query_conditional_independence), 103
- theme_dag (theme_dag_blank), 127
- theme_dag_blank, 127
- theme_dag_gray (theme_dag_grey), 128
- theme_dag_gray_grid (theme_dag_grey), 128
- theme_dag_grey, 128
- theme_dag_grey_grid (theme_dag_grey), 128
- theme_dag_grid (theme_dag_blank), 127
- tidy_dagitty, 10, 128
- tidy_dagitty(), 10, 11
- tidygraph::as_tbl_graph(), 9
- time_ordered_coords, 130
- transmute.tidy_dagitty (dplyr), 41

`ungroup.tidy_dagitty(dplyr)`, [41](#)
`update_dag(pull_dag)`, [97](#)
`update_dag.tidy_dagitty(pull_dag)`, [97](#)
`update_dag<- (pull_dag)`, [97](#)
`update_dag_data<- (pull_dag)`, [97](#)

Variable Status, [132](#)