

Package: quickdag (via r-universe)

October 17, 2026

Title Make Beautiful Directed Acyclic Graphs

Version 0.2.1.9000

Description This package allows the user the save directed acyclic graphs (DAGs) generated in DiagrammeR and export them to PDF, PNG, SVG, or PS format. Users may optionally choose to view the DAG without saving or to embed it in a document. In addition, quickdag allows the user easily to reformat DAGs as single-world intervention graph (SWIG) templates.

License MIT + file LICENSE

URL <https://jrgant.github.io/quickdag/>,
<https://github.com/jrgant/quickdag>

BugReports <https://github.com/jrgant/quickdag/issues>

Depends R (>= 4.1)

Imports dagitty (>= 0.3.4), DiagrammeR (>= 1.0.11), dplyr (>= 1.1.4),
purrr (>= 1.1.0), rlang (>= 1.1.6), stringr (>= 1.5.1), tibble
(>= 3.3.0), tidyr (>= 1.3.1)

Suggests DiagrammeRsvg (>= 0.1), htmlTable (>= 2.4.3), knitr (>= 1.50), rsvg (>= 2.6.2), testthat (>= 3.2.3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs
cmake libglib-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev libnode-dev libx11-dev

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

Date/Publication 2026-04-20 02:36:54 UTC

RemoteUrl <https://github.com/jrgant/quickdag>

RemoteRef HEAD

RemoteSha 9dda719a188125e13068c83f18879cb76300a373

Contents

get_numids	2
parse_edgestring	2
parse_nodes	3
qd_adjustment_sets	3
qd_dag	4
qd_save	6
qd_set_node_attrs	7
qd_swig	7
qd_themes	8
sep_opts	10
Index	12

get_numids	<i>Retrieves numeric IDs given alphanumeric IDs</i>
------------	---

Description

Retrieves numeric IDs given alphanumeric IDs

Usage

get_numids(graph_obj, alpha_ids)

Arguments

- | | |
|-----------|---|
| graph_obj | A quickdag object output by qd_dag() or qd_swig() . |
| alpha_ids | A vector of alphanumeric node IDs upon which to operate. |

parse_edgestring	<i>Parse edges in a graph specification</i>
------------------	---

Description

Parse edges in a graph specification

Usage

parse_edgestring(edgestring)

parse_edges(edgelist)

Arguments

- | | |
|------------|--|
| edgestring | A single string containing some set of node-edge relationships |
| edgelist | A vector of node-edge relationships. |

parse_nodes	<i>Parse nodes</i>
-------------	--------------------

Description

Parse nodes

Usage

```
parse_nodes(edgelist)
```

Arguments

edgelist A vector of node-edge relationships.

qd_adjustment_sets	<i>Identify variables for adjustment</i>
--------------------	--

Description

Format an edgelist and send it to dagitty to identify variable adjustment sets.

Usage

```
qd_adjustment_sets(  
  edgelist,  
  diagram_type = "dag",  
  showplot = FALSE,  
  exposure,  
  outcome,  
  ...  
)
```

```
qd_todagitty(  
  edgelist,  
  diagram_type = "dag",  
  showplot = FALSE,  
  exposure,  
  outcome,  
  ...  
)
```

Arguments

edgelist	A vector of edge relationships.
diagram_type	Character identifying the diagram type. Defaults to "dag", but user can specify another graph type (see dagitty::adjustmentSets()).
showplot	Logical indicating whether to produce a dagitty plot. Defaults to FALSE.
exposure	Character. Specify exposure of interest. (Required)
outcome	Character. Specify outcome of interest. (Required)
...	Pass arguments to dagitty::adjustmentSets() .

Details

The exposure and outcome options map to dagitty parameters of the same name.

[qd_todagitty\(\)](#) remains an alias for [qd_adjustment_sets\(\)](#) to avoid breaking existing scripts, as it was the original name of this function.

Examples

```
# feed an edgelist to qd_adjustment_sets()
edges <- c("A -> { B C D }",
          "B -> C",
          "E -> { B C }",
          "Y <- L -> A")

qd_adjustment_sets(edges, exposure = "A", outcome = "C")
qd_adjustment_sets(edges, exposure = "A", outcome = "C", type = "minimal")

# if you've already created a qd_dag() object
dag <- qd_dag(edges)
qd_adjustment_sets(dag$qd_edgelist, exposure = "A", outcome = "Y")
```

qd_dag	<i>Generate a graph object</i>
--------	--------------------------------

Description

Provide simple syntax specifying paths between nodes to generate a graph object.

Usage

```
qd_dag(
  edgelist,
  node_labs = NULL,
  node_aes_opts = list(),
  edge_aes_opts = list(),
  format_special = getOption("quickdag.format_special"),
  verbose = getOption("quickdag.verbose"),
  check_dag = getOption("quickdag.check_dag"),
```

```

    theme = getOption("quickdag.theme"),
    ...
  )

```

Arguments

edgelist	A vector of node-edge relationships.
node_labs	A named character vector containing label names. Defaults to NULL.
node_aes_opts	A list feeding aesthetic options for nodes to DiagrammeR::node_aes() . Defaults to empty list.
edge_aes_opts	A list feeding aesthetic options for edges to DiagrammeR::edge_aes() . Defaults to empty list.
format_special	Render numeric elements in an alphanumeric alpha_id as subscripts. Defaults to FALSE.
verbose	Indicate whether to print node and edge dataframes to the console. See Details below. Defaults to TRUE.
check_dag	Logical. Check whether the graph conforms to the rules of DAGs. Defaults to TRUE.
theme	Choose theme for plot output. Defaults to "base". Setting theme to NULL will use DiagrammeR's NULL attribute theme.
...	Pass optional conditioned argument to qd_themes() .

Details

Leaving the verbose set to TRUE may be advisable to ensure labels and IDs have not been mismatched. By default, [qd_dag\(\)](#) alphabetizes nodes included in edgelist and does the same for node_labs under a first assumption that labels will begin with the same letter as their corresponding alpha_id, which may not always be the case.

Examples

```

# Provide a list of edges, with nodes specified as letters or single words.
edges <- c("A -> { B C } <- L",
          "B -> C")

# Make a DAG object and render the graph using the default theme
g.obj <- qd_dag(edges)
DiagrammeR::render_graph(g.obj)

# Pass labels and aesthetic options for nodes or edges
g.obj2 <- qd_dag(edges,
  node_labs = c("A" = "Alcohol",
               "B" = "BP",
               "C" = "CVD",
               "L" = "State"),
  node_aes_opts = list(shape = "plaintext",
                       fillcolor = "none",
                       color = "black"),

```

```

edge_aes_opts = list(arrowsize = 0.5,
                     color = "gray"),
theme = NULL)
DiagrammeR::render_graph(g.obj2)

```

qd_save

Save graph objects as images

Description

`qd_save()` is a light wrapper around `DiagrammeR::export_graph()`, which enables exporting high-quality and/or scalable graphics for both print and online. For convenience in computational notebooks like quarto or RMarkdown documents, `qd_save()` also allows the user to embed the image via `knitr::include_graphics()`.

Usage

```
qd_save(graph, file_name, ..., embed = getOption("quickerdag.embed"), kg = NULL)
```

Arguments

<code>graph</code>	A graph object produced by <code>qd_dag()</code>
<code>file_name</code>	A filename (possibly including a path) to which to save the image. File type will be determined by the extension provided (pdf, svg, png, ps). See <code>DiagrammeR::export_graph()</code> for details.
<code>...</code>	Pass other arguments to <code>DiagrammeR::render_graph()</code> .
<code>embed</code>	Defaults to FALSE.
<code>kg</code>	A list allowing the user to set arguments for <code>knitr::include_graphics()</code> (except for path).

Examples

```

# redirect file to temporary directory (example only)
file <- file.path(tempdir(), "dag.pdf")

# save without embedding
dag <- qd_dag(c("L -> {A Y}", "A -> Y"),
             check_dag = FALSE, verbose = FALSE)
qd_save(dag, file)

# embed from code chunk using knitr::include_graphics()
qd_save(dag, file, embed = TRUE)

# `title` passed to DiagrammeR::export_graph()
qd_save(dag, file, title = "Demo")

# clean up temporary directory
file.remove(file)

```

qd_set_node_attrs	<i>Graph operations</i>
-------------------	-------------------------

Description

Graph operations

Usage

```
qd_set_node_attrs(graph_obj, ..., alpha_ids)

qd_set_edge_attrs(graph_obj, ..., from_alpha = NULL, to_alpha = NULL)

select_nodes_by_alpha_id(graph_obj, alpha_ids, set_op = "union")

qd_select_nodes(graph_obj, alpha_ids, set_op = "union")

select_edges_by_node_alpha_id(graph_obj, alpha_ids, set_op = "union")

qd_select_edges(graph_obj, alpha_ids, set_op = "union")
```

Arguments

graph_obj	A quickdag object output by qd_dag() or qd_swig() .
...	Passed to DiagrammeR::set_node_attrs() or DiagrammeR::set_edge_attrs() argument node_attr or edge_attr, respectively. DiagrammeR::set_edge_attrs() argument of the same name.
alpha_ids	A vector of alphanumeric node IDs upon which to operate.
from_alpha	A vector of alphanumeric source node IDs.
to_alpha	A vector of alphanumeric destination node IDs.
set_op	Passed to the set_op argument of DiagrammeR::select_nodes_by_id() or DiagrammeR::select_edges_by_node_id() . Defaults to "union".

qd_swig	<i>Generate a single-world intervention graph (SWIG)</i>
---------	--

Description

Take a DAG graph object and, in the simplest case, create a single-world intervention template corresponding to a world in which the fixed nodes are set to a given value. Alternatively, tell [qd_swig\(\)](#) which values fixed nodes will be set to.

Usage

```
qd_swig(
  graph_obj,
  fixed_nodes,
  custom_values = NULL,
  fixed_sep = getOption("quickdag.swig.fixedsep"),
  sep_point_size = getOption("quickdag.swig.sepsize")
)
```

Arguments

graph_obj	A DAG object created by <code>qd_dag()</code> .
fixed_nodes	A vector containing the nodes to be intervened upon.
custom_values	A named vector containing alternative labels identifying explicit values for fixed nodes (e.g., <code>c("A" = "1")</code>).
fixed_sep	A character string indicating which character to use as a separator in fixed nodes. Defaults to "vlin". Run <code>sep_opts(TRUE)</code> for available options.
sep_point_size	A numerical value specifying the point size for fixed node separators.

Examples

```
# Provide a DAG object and a list of nodes to be fixed
edges <- c("A -> Y",
          "L -> { A Y }")

dag <- qd_dag(edges)

swig <- dag |>
  qd_swig(fixed_nodes = "A",
          custom_values = c("A" = "1"))

swig |> DiagrammeR::render_graph()
```

qd_themes

Diagram themes

Description

Themes and theming utilities.

Usage

```
qd_themes(graph_obj, theme, conditioned = NULL)

theme_qd_base(
  graph_obj,
```

```

    rankdir = getOption("quickdag.base_rankdir"),
    layout = getOption("quickdag.base_layout"),
    shape = getOption("quickdag.base_shape"),
    nodepen = getOption("quickdag.base_nodepen"),
    nodewidth = getOption("quickdag.base_nodewidth"),
    nodeheight = getOption("quickdag.base_nodeheight"),
    edgepen = getOption("quickdag.base_edgepen"),
    arrowhead = getOption("quickdag.base_arrowhead"),
    arrowsize = getOption("quickdag.base_arrowsize"),
    fontname = getOption("quickdag.base_fontname"),
    fontsize = getOption("quickdag.base_fontsize"),
    fontcolor = getOption("quickdag.base_fontcolor"),
    conditioned = NULL
)

theme_qd_circles(
  graph_obj,
  nodepen = choose_option("quickdag.circles_nodepen"),
  edgepen = choose_option("quickdag.circles_edgepen"),
  fontname = choose_option("quickdag.circles_fontname"),
  fontsize = choose_option("quickdag.circles_fontsize"),
  fontcolor = choose_option("quickdag.circles_fontcolor"),
  conditioned = NULL
)

theme_qd_pearl(
  graph_obj,
  pointsize = getOption("quickdag.pearl_pointsize"),
  pointcolor = getOption("quickdag.pearl_pointcolor"),
  pointfill = getOption("quickdag.pearl_pointfill"),
  edgepen = choose_option("quickdag.pearl_edgepen"),
  arrowhead = choose_option("quickdag.pearl_arrowhead"),
  arrowsize = choose_option("quickdag.pearl_arrowsize"),
  fontname = choose_option("quickdag.pearl_fontname"),
  fontsize = choose_option("quickdag.pearl_fontsize"),
  fontcolor = choose_option("quickdag.pearl_fontcolor"),
  conditioned = NULL
)

get_conditioned_nodes(graph_obj, conditioned = NULL)

cleanup_existing_theme(graph_obj)

```

Arguments

<code>graph_obj</code>	A DAG object created by <code>qd_dag()</code> .
<code>theme</code>	A character string indicating the theme to use. Defaults to "base". Set to NULL to use GraphViz defaults.

conditioned	A character vector indicating which nodes are conditioned upon. The shape for these nodes will be set to "rectangle".
rankdir	Direction of node layout. Defaults to "LR", for left-to-right.
layout	Layout engine. Defaults to "dot", which is the only engine explicitly used by quickdag . Other options provided by DiagrammeR::DiagrammeR may or may not work well.
shape	Standard node shape for a given graph. Defaults for each theme: "base" = plain-text, "circles" = circle, "pearl" = point. See Graphviz documentation for other options.
nodepen	Size (in points) of node outlines. By default, controlled by package options specific to each theme, each of which defaults to 0.2.
nodewidth	Width (in inches) of nodes. By default, controlled by package options specific to each theme. Defaults to 0.
nodeheight	Height (in inches) of nodes. By default, controlled by package options specific to each theme. Defaults to 0.
edgepen	Size (in points) of edges. By default, controlled by package options specific to each theme, each of which defaults to 0.2.
arrowhead	Shape of the arrowhead. See DiagrammeR::edge_aes() .
arrowsize	Size of the arrowhead. See DiagrammeR::edge_aes() .
fontname	Font name for text elements of the graph. By default, controlled by package options specific to each theme, each of which defaults to "Helvetica". This is the default in DiagrammeR.
fontsize	Size of text (in points). By default, controlled by package options specific to each theme, each of which defaults to 5.
fontcolor	Text color. By default, controlled by package options specific to each theme, each of which defaults to "black".
pointsize	Size of "point" shape (in points). Applies to "pearl" theme only and defaults to 0.02.
pointcolor	Border color for the "point" shape. Applies to "pearl" theme only and defaults to "black".
pointfill	Fill color for the "point" shape. Applies to "pearl" theme only and defaults to "black".

sep_opts

View options for fixed node separator characters

Description

Preview character options for use as the fixed node separator in SWIGs.

Usage

```
sep_opts(entity = NULL, table = FALSE)
```

Arguments

entity	The separator name or an HTML entity of the user's choosing. Defaults to NULL.
table	Logical to show or hide HTML table display of built-in characters. Defaults to FALSE.

Details

By default `sep_opts()` returns a named vector listing the built-in separators. These built-in HTML entities are attached to `nicenames` for convenience. If the user provides the `nickname` for a built-in entity to the `entity` argument, the function will return the appropriate HTML code. If the user provides an HTML entity that does not appear in the built-in menu, the input will be returned verbatim. Users will typically access this latter functionality via `qd_swig()`.

Index

`cleanup_existing_theme (qd_themes)`, 8

`dagitty::adjustmentSets()`, 4

`DiagrammeR::DiagrammeR`, 10

`DiagrammeR::edge_aes()`, 5, 10

`DiagrammeR::export_graph()`, 6

`DiagrammeR::node_aes()`, 5

`DiagrammeR::render_graph()`, 6

`DiagrammeR::select_edges_by_node_id()`, 7

`DiagrammeR::select_nodes_by_id()`, 7

`DiagrammeR::set_edge_attrs()`, 7

`DiagrammeR::set_node_attrs()`, 7

`get_conditioned_nodes (qd_themes)`, 8

`get_numids`, 2

`knitr::include_graphics()`, 6

`parse_edges (parse_edgestring)`, 2

`parse_edgestring`, 2

`parse_nodes`, 3

`qd_adjustment_sets`, 3

`qd_adjustment_sets()`, 4

`qd_dag`, 4

`qd_dag()`, 2, 5–9

`qd_save`, 6

`qd_save()`, 6

`qd_select_edges (qd_set_node_attrs)`, 7

`qd_select_nodes (qd_set_node_attrs)`, 7

`qd_set_edge_attrs (qd_set_node_attrs)`, 7

`qd_set_node_attrs`, 7

`qd_swig`, 7

`qd_swig()`, 2, 7, 11

`qd_themes`, 8

`qd_themes()`, 5

`qd_todagitty (qd_adjustment_sets)`, 3

`qd_todagitty()`, 4

`quickdag`, 10

`select_edges_by_node_alpha_id (qd_set_node_attrs)`, 7

`select_nodes_by_alpha_id (qd_set_node_attrs)`, 7

`sep_opts`, 10

`sep_opts()`, 11

`theme_qd_base (qd_themes)`, 8

`theme_qd_circles (qd_themes)`, 8

`theme_qd_pearl (qd_themes)`, 8