Package ‘DAGassist’

September 21, 2025

Title Test Robustness with Directed Acyclic Graphs

Version 0.2.4

Description Provides robustness checks driven by directed acyclic graphs (DAGs). Given a 'dagitty' DAG object and a model specification, 'DAGassist' classifies variables by causal roles, flags problematic controls, and generates a report comparing the original model with minimal and canonical adjustment sets. Exports publication-grade reports in 'LaTeX', 'Word', 'Excel', or plain text. 'DAGassist' is built on 'dagitty', an 'R' package that uses the 'DAGitty' web tool (<https://dagitty.net/>) for creating and analyzing DAGs. Methods draw on Pearl (2009) <doi:10.1017/CBO9780511803161> and Textor et al. (2016) <doi:10.1093/ije/dyw341>.

License GPL (>= 2)

URL https://github.com/grahamgoff/DAGassist, https://grahamgoff.github.io/DAGassist/

BugReports https://github.com/grahamgoff/DAGassist/issues

Depends R (>= 3.5)

Imports broom, cli, crayon, dagitty, magrittr, stats, tools, utils, writexl

Suggests fixest, ggdag, knitr, modelsummary, rmarkdown, testthat (>= 3.0.0), tidyverse

VignetteBuilder knitr

Config/Needs/website pkgdown, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Graham Goff [aut, cre] (ORCID: <https://orcid.org/0000-0002-0717-6995>), Michael Denly [ctb] (ORCID: <https://orcid.org/0000-0002-7074-5011>)

Maintainer Graham Goff <goffgrahamc@gmail.com>

Repository CRAN

Date/Publication 2025-09-21 14:00:12 UTC

Contents

`bad_controls_in` *flag bad controls (mediator/collider/desc of Y) among a candidate set*

Description

flag bad controls (mediator/collider/desc of Y) among a candidate set

Usage

```
bad_controls_in(dag, controls, exposure, outcome)
```

Arguments

`dag`	A `dagitty` DAG object.
`controls`	Character vector of variable names.
`exposure`	Character; exposure node name (X).
`outcome`	Character; outcome node name (Y).

Value

A character vector (possibly empty) containing the elements of `controls` that are identified as "bad controls".

This is essentially the inverse of `pick_minimal_controls()`, as it returns bad controls, rather than the minimal/canonical set of good controls

Examples

```
d <- ggdag::dagify(
Y ~ X + M + Z,
M ~ X + Z,
C ~ X + Y,
exposure = "X",
outcome = "Y")
# M: mediator / Z: confounder / C: collider

# hypothetical candidate controls
controls <- c("Z", "M", "C")
```

```
# Flag controls that would bias the total effect of X on Y:
bad_controls_in(d, controls = c("Z","M","C"), exposure = "X", outcome = "Y")

# expected: c("M", "C")  # mediator & collider are "bad controls"; Z is OK
```

`classify_nodes` *Classify DAG nodes*

Description

Labels each node as one of: `exposure`, `outcome`, `confounder`, `mediator`, `collider`, `descendant_of_outcome`, or `other`.

Usage

```
classify_nodes(dag, exposure, outcome)
```

Arguments

`dag`	A `dagitty` DAG object.
`exposure`	Optional– inferred from DAG if not set; character; exposure node name (X).
`outcome`	Optional– inferred from DAG if not set; character; outcome node name (Y).

Details

label definitions *confounder* – ancestor of both X and Y, and not a descendant of X *mediator* – descendant of X and ancestor of Y *collider* – node with 2 or more parents on an X / Y path (non-structural) *descendant_of_outcome* – any descendant of Y `exposure` / `outcome` labeled explicitly in function call

Notes:

- in definitions, x is exposure and y is outcome
- structural colliders are calculated, but only to define non-structural. structural colliders are not included as a boolean flag
- A node may satisfy multiple properties; we also return boolean flags for each property. The `role` column gives a single "primary" label using the precedence defined below.

Value

A data.frame with one row per node and columns:

- `variable` (node name)
- logical flags: `is_exposure`, `is_outcome`, `is_confounder`, `is_mediator`, `is_collider`, `is_descendant_of_outcome` `is_descendant_of_exposure`
- `role` (a single primary label)

Examples

```
d1 <- dagitty::dagitty("dag { Z -> X; Z -> Y; X -> Y }") # confounder Z
classify_nodes(d1, exposure = "X", outcome = "Y")

d2 <- dagitty::dagitty("dag { X -> M -> Y }") # mediator M
classify_nodes(d2, "X", "Y")

d3 <- dagitty::dagitty("dag { X -> C <- Y }") # collider C
classify_nodes(d3, "X", "Y")
```

`DAGassist` *Produce a compact* `DAGassist` *report (console/LaTeX/Word/Excel/Text)*

Description

`DAGassist()` validates a DAG + model specification, classifies node roles, builds minimal and canonical adjustment sets, fits comparable models, and renders a compact report in several formats (console, LaTeX fragment, DOCX, XLSX, plain text). It also supports passing a **single engine call** (e.g. `feols(Y ~ X + Z | fe, data = df)`) instead of a plain formula.

Usage

```
DAGassist(
  dag,
  formula,
  data,
  exposure,
  outcome,
  engine = stats::lm,
  labels = NULL,
  verbose = TRUE,
  type = c("console", "latex", "word", "docx", "excel", "xlsx", "text", "txt"),
  out = NULL,
  imply = FALSE,
  omit_intercept = TRUE,
  omit_factors = TRUE,
  engine_args = list()
)
```

Arguments

`dag`	A **dagitty** object (see `dagitty::dagitty()`).
`formula`	Either (a) a standard model formula `Y ~ X + ...`, or (b) a single **engine call** such as `feols(Y ~ X + Z | fe, data = df, ...)`. When an engine call is provided, `engine`, `data`, and extra arguments are automatically extracted from the call.

`data`	A `data.frame` (or compatible, e.g. tibble). Optional if supplied via the engine call in `formula`.
`exposure`	Optional character scalar; if missing/empty, inferred from the DAG (must be unique).
`outcome`	Optional character scalar; if missing/empty, inferred from the DAG (must be unique).
`engine`	Modeling function, default stats::lm. Ignored if `formula` is a single engine call (in that case the function is taken from the call).
`labels`	Optional variable labels (named character vector or data.frame).
`verbose`	Logical (default `TRUE`). Controls verbosity in the console printer (formulas + notes).
`type`	Output type. One of `"console"` (default), `"latex"`/`"docx"`/`"word"`, `"excel"`/`"xlsx"`, `"text"`/`"txt"`.
`out`	Output file path for the non-console types: • `type="latex"`: a **LaTeX fragment** written to `out` (must end with `.tex`). • `type="docx"`/`"word"`: a **Word (.docx)** file written to `out`. • `type="excel"`/`"xlsx"`: an **Excel (.xlsx)** file written to `out`. • `type="text"`/`"txt"`: a **plain-text** file written to `out`. Ignored for `type="console"`.
`imply`	Logical; default `FALSE`. **Evaluation scope.** • If `FALSE` (default): restrict DAG evaluation to variables **named in the formula** (prune the DAG to exposure, outcome, and RHS terms). Roles/sets/bad-controls are computed on this pruned graph, and the roles table **only** shows those variables. This is most useful if you want to refine your specific call. • If `TRUE`: evaluate on the **full DAG** and allow DAG-implied controls in the minimal/canonical sets; roles table shows all nodes. This is most useful if you want to refine your overall control variable selection.
`omit_intercept`	Logical; drop intercept rows from the model comparison (default `TRUE`).
`omit_factors`	Logical; drop factor-level rows from the model comparison (default `TRUE`).
`engine_args`	Named list of extra arguments forwarded to `engine(...)`. If `formula` is an engine call, arguments from the call are merged with `engine_args` (call values take precedence).

Details

Engine-call parsing. If `formula` is a call (e.g., `feols(Y ~ X | fe, data=df)`), DAGassist extracts the engine function, formula, data argument, and any additional engine arguments directly from that call; these are merged with `engine`/`engine_args` you pass explicitly (call arguments win).

Fixest tails. For engines like **fixest** that use `|` to denote FE/IV parts, DAGassist preserves any `| ...` tail when constructing minimal/canonical formulas (e.g., `Y ~ X + controls | fe | iv(...)`).

Roles grid. The roles table displays short headers:

- `X` (exposure), `Y` (outcome), `CON` (confounder), `MED` (mediator), `COL` (collider), `IO` (intermediate outcome = proper descendant of `Y`), `DMed` (proper descendant of any mediator), `DCol` (proper descendant of any collider). Descendants are **proper** (exclude the node itself) and can be any distance downstream. The internal `is_descendant_of_exposure` is retained for logic but hidden in displays.

Bad controls. For total-effect estimation, DAGassist flags as `bad controls` any variables that are `MED`, `COL`, `IO`, `DMed`, or `DCol`. These are warned in the console and omitted from the model-comparison table. Valid confounders (pre-treatment) are eligible for minimal/canonical adjustment sets.

Output types.

- `console` prints roles, sets, formulas (if `verbose`), and a compact model comparison with `{modelsummary}` if available (falls back gracefully otherwise).
- `latex` writes a **LaTeX fragment** you can `\\input{}` into a paper.
- `docx`/`word` writes a **Word** doc (uses `options(DAGassist.ref_docx=...)` if set).
- `excel`/`xlsx` writes an **Excel** workbook with tidy tables.
- `text`/`txt` writes a **plain-text** report for logs/notes.

Dependencies. Core requires `{dagitty}`. Optional enhancements: `{modelsummary}` (pretty tables), `{broom}` (fallback tidying), `{rmarkdown}` + **Pandoc** (DOCX), `{writexl}` (XLSX).

Value

An object of class `"DAGassist_report"`, invisibly for file outputs, and printed for `type="console"`. The list contains:

- `validation` - result from `validate_spec(...)` which verifies acyclicity and X/Y declarations.
- `roles` - raw roles data.frame from `classify_nodes(...)` (logic columns).
- `roles_display` - roles grid after labeling/renaming for exporters.
- `bad_in_user` - variables in the user's RHS that are `MED`/`COL`/`IO`/`DMed`/`DCol`.
- `controls_minimal` - (legacy) one minimal set (character vector).
- `controls_minimal_all` - list of all minimal sets (character vectors).
- `controls_canonical` - canonical set (character vector; may be empty).
- `formulas` - list with `original`, `minimal`, `minimal_list`, `canonical`.
- `models` - list with fitted models `original`, `minimal`, `minimal_list`, `canonical`.
- `verbose`, `imply` - flags as provided.

Interpreting the output

ROLES. Variables in your formula are classified by DAG-based causal role:

- `X` - treatment / exposure.
- `Y` - outcome / dependent variable.
- `CON` - confounder (common cause of `X` and `Y`); adjust for these.
- `MED` - mediator (on a path from `X` to `Y`); do **not** adjust when estimating total effects.
- `COL` - collider (direct descendant of `X` and `Y`); adjusting opens a spurious path, so do **not** adjust.
- `IO` - intermediate outcome (descendant of `Y`); do **not** adjust.
- `DMed` - descendant of a mediator; do **not** adjust when estimating total effects.

- `DCol` - descendant of a collider; adjusting opens a spurious path, so do **not** adjust.
- `other` - safe, non-confounding predictors (e.g., affect `Y` only). Included in the canonical model but omitted from the minimal set because they're not required for identification.

MODEL COMPARISON.

- **Minimal** - the smallest adjustment set that blocks all back-door paths (confounders only).
- **Canonical** - the largest permissible set: includes all controls that are not `MED`, `COL`, `IO`, `DMed`, or `DCol`. `other` variables may appear here.

Errors and edge cases

- If exposure/outcome cannot be inferred uniquely, the function stops with a clear message.
- Fitting errors (e.g., FE collinearity) are captured and displayed in comparisons without aborting the whole pipeline.

See Also

`print.DAGassist_report()` for the console printer, and the helper exporters in `report_*` modules.

Examples

```
# generate a console DAGassist report
DAGassist(dag = g, formula = lm(Y ~ X + Z + C + M, data = df))

# generate a LaTeX DAGassist report

DAGassist(dag = g, formula = lm(Y ~ X + Z + C + M, data = df),
          type = "latex", out = file.path(tempdir(), "frag.tex"))
```

`print.DAGassist_report`

Print method for DAGassist reports

Description

Nicely prints the roles table, highlights potential bad controls, shows minimal/canonical adjustment sets, optionally shows formulas, and renders a compact model comparison (using `{modelsummary}` if available, falling back to `{broom}` or basic `coef()` preview).

Usage

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

Arguments

`x`	A `"DAGassist_report"` object returned by `DAGassist()`.
`...`	Additional arguments (currently unused; present for S3 compatibility).

Details

The printer respects the `verbose` flag in the report: when `TRUE`, it includes formulas and a brief note on variables added by DAG logic (minimal and canonical sets). Fitting errors are shown inline per model column and do not abort printing.

Value

Invisibly returns `x`.

`print.DAGassist_roles` *Print node classifications (aligned)*

Description

Print node classifications (aligned)

Usage

```
## S3 method for class 'DAGassist_roles'
print(x, n = Inf, ...)
```

Arguments

`x`	Output of classify_nodes() (class "DAGassist_roles")
`n`	Max rows to print (default all)
`...`	(ignored)

Value

Invisibly returns x

`print.DAGassist_validation` *Minimal, clean printout for validation results with color coding*

Description

Minimal, clean printout for validation results with color coding

Usage

```
## S3 method for class 'DAGassist_validation'
print(x, n = 10, ...)
```

Arguments

`x`	the list (class `out`) from validate_spec
`n`	Max number of issues to show (default 10).
`...`	Ignored.

Value

Invisibly returns `x`.

Index