Package: flexCausal (via r-universe)

June 1, 2026

Title Causal Effect Estimation via Doubly Robust One-Step Estimators and TMLE in Graphical Models with Unmeasured Variables

Version 0.1.0

Date 2026-03-18

Description Provides doubly robust one-step and targeted maximum likelihood (TMLE) estimators for average causal effects in acyclic directed mixed graphs (ADMGs) with unmeasured variables. Automatically determines whether the treatment effect is identified via backdoor adjustment or the extended front-door functional, and dispatches to the appropriate estimator. Supports incorporation of machine learning algorithms via 'SuperLearner' and cross-fitting for nuisance estimation. Methods are described in Guo and Nabi (2024) <doi:10.48550/arXiv.2409.03962>.

License GPL-3

LazyData true

URL https://github.com/annaguo-bios/flexCausal

BugReports https://github.com/annaguo-bios/flexCausal/issues

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3.9000

Imports rlang, dplyr, SuperLearner, densratio, MASS, mvtnorm, stats, utils

Depends R (>= 4.1)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), earth, ranger

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://annaguo-bios.r-universe.dev

Date/Publication 2026-03-19 20:51:57 UTC

RemoteUrl https://github.com/annaguo-bios/flexcausal

RemoteRef HEAD

RemoteSha 229d80365c3e6500a36c55a686baae5c0b1fa433

Contents

`data_backdoor` *Data generated from a classic backdoor model, where the pre-treatment variable X cause A and Y, and the treatment A cause Y.*

Description

Data generated from a classic backdoor model, where the pre-treatment variable X cause A and Y, and the treatment A cause Y.

Usage

```
data_backdoor
```

Format

A data frame with 2000 rows and 3 variables: X, A, Y.

X a variable that follows uniform distribution at the interval of $[0, 1]$

A a binary treatment

Y a variable that follows normal distribution

References

https://github.com/annaguo-bios/flexCausal

---

data_example_a — *Data generated from the simulation model in Figure 4(a) of the paper* https://arxiv.org/abs/2409.03962. *The model can also be found in Figure (a) of the github repository:* https://github.com/annaguo-bios/flexCausal

---

Description

Data generated from the simulation model in Figure 4(a) of the paper https://arxiv.org/abs/2409.03962. The model can also be found in Figure (a) of the github repository: https://github.com/annaguo-bios/flexCausal

Usage

```
data_example_a
```

Format

A data frame with 2000 rows and 7 variables: X, A, U, M=(M.1,M.2), L, Y.

X a variable that follows uniform distribution at the interval of $[0, 1]$

A a binary treatment

U an unmeasured confounder following a normal distribution

M.1 first component of the bivariate mediator M, following a normal distribution

M.2 second component of the bivariate mediator M, following a normal distribution

L a variable that follows normal distribution

Y a variable that follows normal distribution

References

https://github.com/annaguo-bios/flexCausal

`data_example_b` *Data generated from the simulation model in Figure 4(b) of the paper* https://arxiv.org/abs/2409.03962. *The model can also be found in Figure (b) of the github repository:* https://github.com/annaguo-bios/flexCausal

---

Description

Data generated from the simulation model in Figure 4(b) of the paper https://arxiv.org/abs/2409.03962. The model can also be found in Figure (b) of the github repository: https://github.com/annaguo-bios/flexCausal

Usage

```
data_example_b
```

Format

A data frame with 2000 rows and 6 variables: X, A, U1, U2, M=(M.1,M.2), L, Y.

X a variable that follows uniform distribution at the interval of $[0, 1]$

A a binary treatment

U1 first unmeasured confounder following a normal distribution

U2 second unmeasured confounder following a normal distribution

M.1 first component of the bivariate mediator M, following a normal distribution

M.2 second component of the bivariate mediator M, following a normal distribution

L a variable that follows normal distribution

Y a variable that follows normal distribution

References

https://github.com/annaguo-bios/flexCausal

`data_frontdoor` *Data generated from a classic front-door model.*

Description

Data generated from a front-door model where a pre-treatment variable X and an unmeasured confounder U affect both the treatment A and outcome Y, and the treatment A affects Y through a binary mediator M.

Usage

```
data_frontdoor
```

Format

A data frame with 2000 rows and 5 variables:

X a pre-treatment variable following a uniform distribution on $[0, 1]$

U an unmeasured confounder following a normal distribution

A a binary treatment variable (0/1)

M a binary mediator variable (0/1)

Y a continuous outcome variable following a normal distribution

References

https://github.com/annaguo-bios/flexCausal

---

estADMG — *Estimate the average causal effect (ACE) or the average counterfactual outcome E(Y(a)) from observational data under a DAG with hidden variables.*

Description

The main user-facing function of the package. Given a causal graph specified as an acyclic directed mixed graph (ADMG), this function automatically determines the identifiability status of the treatment effect and dispatches to the appropriate estimator:

- If the treatment is *fixable* (i.e., backdoor-adjustable), estimation proceeds via `.call_backdoor`, returning G-computation, IPW, one-step (AIPW), and TMLE estimators.
- If the treatment is *primal fixable* (extended front-door functional), estimation proceeds via `.call_nps`, returning one-step and TMLE estimators.
- If the treatment is neither fixable nor primal fixable, the function stops with an error.

A message is also printed indicating whether the graph is nonparametrically saturated, in which case the returned estimators are semiparametrically efficient.

Usage

```
estADMG(
  a = NULL,
  data = NULL,
  vertices = NULL,
  di_edges = NULL,
  bi_edges = NULL,
  treatment = NULL,
```

```
  outcome = NULL,
  multivariate.variables = NULL,
  graph = NULL,
  superlearner.seq = F,
  superlearner.Y = F,
  superlearner.A = F,
  superlearner.M = F,
  superlearner.L = F,
  crossfit = F,
  K = 5,
  ratio.method.L = "bayes",
  ratio.method.M = "bayes",
  dnorm.formula.L = NULL,
  dnorm.formula.M = NULL,
  lib.seq = c("SL.glm", "SL.earth", "SL.ranger", "SL.mean"),
  lib.L = c("SL.glm", "SL.earth", "SL.ranger", "SL.mean"),
  lib.M = c("SL.glm", "SL.earth", "SL.ranger", "SL.mean"),
  lib.Y = c("SL.glm", "SL.earth", "SL.ranger", "SL.mean"),
  lib.A = c("SL.glm", "SL.earth", "SL.ranger", "SL.mean"),
  formulaY = "Y ~ .",
  formulaA = "A ~ .",
  linkY_binary = "logit",
  linkA = "logit",
  n.iter = 500,
  cvg.criteria = 0.01,
  truncate_lower = 0,
  truncate_upper = 1,
  zerodiv.avoid = 0
)
```

Arguments

a	Numeric scalar or length-two numeric vector specifying the treatment level(s) of interest. The treatment must be coded as 0/1. If a scalar, the function returns $E\{Y(a)\}$. If a length-two vector c(a1, a0), the function returns the contrast $E\{Y(a1)\} - E\{Y(a0)\}$.
data	A data frame containing all variables listed in vertices.
vertices	A character vector of variable names in the causal graph. Ignored if graph is provided.
di_edges	A list of length-two character vectors specifying directed edges. For example, list(c('A', 'B')) encodes A -> B. Ignored if graph is provided.
bi_edges	A list of length-two character vectors specifying bidirected edges. For example, list(c('A', 'B')) encodes A <-> B. Ignored if graph is provided.
treatment	A character string naming the binary (0/1) treatment variable in data.
outcome	A character string naming the outcome variable in data.
multivariate.variables	A named list mapping compound vertex names to their column names in data.

For example, `list(M = c('M.1', 'M.2'))` indicates M is bivariate with columns M.1 and M.2. Ignored if `graph` is provided.

`graph` A graph object created by `make.graph`. If supplied, `vertices`, `di_edges`, `bi_edges`, and `multivariate.variables` are ignored.

`superlearner.seq`
Logical. If `TRUE`, SuperLearner is used for sequential regression of intermediate variables (primal fixable case only). Default is `FALSE`.

`superlearner.Y` Logical. If `TRUE`, SuperLearner is used for outcome regression. Default is `FALSE`.

`superlearner.A` Logical. If `TRUE`, SuperLearner is used for propensity score estimation. Default is `FALSE`.

`superlearner.M` Logical. If `TRUE`, SuperLearner is used for density ratio estimation for variables in M via the Bayes method (primal fixable case only). Default is `FALSE`.

`superlearner.L` Logical. If `TRUE`, SuperLearner is used for density ratio estimation for variables in L via the Bayes method (primal fixable case only). Default is `FALSE`.

`crossfit` Logical. If `TRUE`, cross-fitting with `K` folds is applied to all SuperLearner fits. Default is `FALSE`.

`K` A positive integer specifying the number of cross-fitting folds. Used only when `crossfit = TRUE`. Default is 5.

`ratio.method.L` A character string specifying the method for estimating density ratios for variables in L (primal fixable case only). Options are:

`"bayes"` (Default) Rewrites the ratio via Bayes' rule as $[p(A = a_0|L, mp(L))/p(A = a_1|L, mp(L))]/[p(A = a_0|mp(L))/p(A = a_1|mp(L))]$ and estimates each factor via logistic regression or SuperLearner.

`"dnorm"` Assumes $L|mp(L), A$ is Gaussian (continuous L) or Bernoulli (binary L), estimated via linear or logistic regression with linear terms only.

`"densratio"` Uses the `densratio` package. Supports only numeric/integer variables and is computationally expensive; not recommended for graphs with many variables.

`ratio.method.M` A character string specifying the method for estimating density ratios for variables in M (primal fixable case only). Same options as `ratio.method.L`. Default is `"bayes"`.

`dnorm.formula.L`
An optional named list of regression formulas for variables in L, used when `ratio.method.L = "dnorm"`. Names are variable names; values are formula strings. Variables omitted from the list are regressed on all Markov pillow variables. For multivariate L, specify one formula per component, e.g. `list(L.1 = "L.1 ~ A + X", L.2 = "L.2 ~ A + X + I(M^2)")`.

`dnorm.formula.M`
An optional named list of regression formulas for variables in M, used when `ratio.method.M = "dnorm"`. Same structure as `dnorm.formula.L`.

`lib.seq` SuperLearner library for sequential regression. Default is `c("SL.glm", "SL.earth", "SL.ranger", "SL.mean")`.

`lib.L` SuperLearner library for density ratio estimation for L. Default is `c("SL.glm", "SL.earth", "SL.ranger", "SL.mean")`.

lib.M	SuperLearner library for density ratio estimation for M. Default is c("SL.glm", "SL.earth", "SL.ranger", "SL.mean").
lib.Y	SuperLearner library for outcome regression. Default is c("SL.glm", "SL.earth", "SL.ranger", "SL.mean").
lib.A	SuperLearner library for propensity score estimation. Default is c("SL.glm", "SL.earth", "SL.ranger", "SL.mean").
formulaY	A formula or character string for outcome regression of Y on its Markov pillow. Used only when superlearner.Y = FALSE. Default is "Y ~ .".
formulaA	A formula or character string for propensity score regression of A on its Markov pillow. Used only when superlearner.A = FALSE. Default is "A ~ .".
linkY_binary	A character string specifying the link function for outcome regression when Y is binary and superlearner.Y = FALSE. Default is "logit".
linkA	A character string specifying the link function for propensity score regression when superlearner.A = FALSE. Default is "logit".
n.iter	Maximum number of TMLE iterations. Default is 500.
cvg.criteria	Numeric. TMLE convergence threshold. The iterative update stops when $\lvert\text{mean}(D^*)\rvert <$ cvg.criteria. Default is 0.01.
truncate_lower	Numeric. Propensity score values below this threshold are clipped. Default is 0 (no clipping).
truncate_upper	Numeric. Propensity score values above this threshold are clipped. Default is 1 (no clipping).
zerodiv.avoid	Numeric. Density ratio or propensity score values below this threshold are clipped to prevent division by zero. Default is 0 (no clipping).

Value

The return structure depends on the identifiability path:

Fixable (backdoor) A named list with components TMLE, Onestep, IPW, and Gcomp, plus per-treatment-level sub-lists.

Primal fixable (front-door) A named list with components TMLE and Onestep.

Examples

```
# Fixable graph: simple backdoor adjustment
test <- estADMG(
  a = 1,
  data = data_backdoor,
  vertices = c('A', 'Y', 'X'),
  di_edges = list(c('X', 'A'), c('X', 'Y'), c('A', 'Y')),
  treatment = 'A',
  outcome = 'Y'
)

# Primal fixable graph: extended front-door functional
test <- estADMG(
  a = 1,
```

```
  data = data_example_a,
  vertices = c('A', 'M', 'L', 'Y', 'X'),
  bi_edges = list(c('A', 'Y')),
  di_edges = list(c('X', 'A'), c('X', 'M'), c('X', 'L'),
                  c('X', 'Y'), c('M', 'Y'), c('A', 'M'),
                  c('A', 'L'), c('M', 'L'), c('L', 'Y')),
  treatment = 'A',
  outcome = 'Y',
  multivariate.variables = list(M = c('M.1', 'M.2'))
)

# ACE estimation E(Y(1)) - E(Y(0))
test <- estADMG(
  a = c(1, 0),
  data = data_example_a,
  vertices = c('A', 'M', 'L', 'Y', 'X'),
  bi_edges = list(c('A', 'Y')),
  di_edges = list(c('X', 'A'), c('X', 'M'), c('X', 'L'),
                  c('X', 'Y'), c('M', 'Y'), c('A', 'M'),
                  c('A', 'L'), c('M', 'L'), c('L', 'Y')),
  treatment = 'A',
  outcome = 'Y',
  multivariate.variables = list(M = c('M.1', 'M.2'))
)
```

`f.adj_matrix` *Build an adjacency matrix from a graph.*

Description

Construct the adjacency matrix implied by the directed edges stored in the graph object.

Usage

```
f.adj_matrix(graph)
```

Arguments

`graph`	A graph object generated by the `make.graph()` function.

Value

An adjacency matrix of the graph.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.adj_matrix(graph)
```

f.children *Get the children of a node OR nodes in a graph.*

Description

Function to extract the children of a node OR nodes in a graph object. Children of a node are the nodes that have edges from the given node.

Usage

```
f.children(graph, nodes)
```

Arguments

graph	A graph object generated by the make.graph() function.
nodes	A character vector of nodes for which to extract children.

Value

A vector of vertices contains children set of the given nodes.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.children(graph, c('A'))
```

f.descendants *Get the descendants of a node OR nodes in a graph.*

Description

Function to extract the descendants of a node OR nodes in a graph object. Descendants of a node Vi are set Vj such that there is a directed path Vi->...->Vj. Descendants set including Vi itself by convention.

Usage

```
f.descendants(graph, nodes)
```

Arguments

graph	A graph object generated by the make.graph() function.
nodes	A character vector of nodes for which to extract children.

Value

A vector of vertices contains descendants set of the given nodes.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.descendants(graph, c('A'))
```

`f.district` *Get the district of a vertex in a graph.*

Description

Function to extract the name of vertices that is in the district of a given vertex in a graph object. District of a unfixed vertex Vi is the set of vertices that are connected to Vi by bidirected edges, including Vi itself by convention.

Usage

```
f.district(graph, node)
```

Arguments

`graph`	A graph object generated by the `make.graph()` function.
`node`	A character string of a vertex for which to extract district.

Value

A vector of vertices that is in the district of the given vertex.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.district(graph, c('A'))
```

f.markov_blanket *Get the Markov blanket of a vertex in a graph.*

Description

Function to get the Markov blanket of a vertex in a graph object. Markov blanket of a vertex Vi is the union of vertices that is in the district of Vi and their parents set. Mb= union(district(Vi), parents(district(Vi))).

Usage

```
f.markov_blanket(graph, node)
```

Arguments

graph	A graph object generated by the make.graph() function.
node	A character string of a vertex for which to extract Markov blanket.

Value

A vector of vertices that is in the Markov blanket of the given vertex.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.markov_blanket(graph, 'A')
```

f.markov_pillow *Get the Markov pillow of a vertex in a graph.*

Description

Function to get the Markov pillow of a vertex in a graph object. Markov pillow of a vertex Vi is the subset of the Markov blanket of Vi that proceed Vi in the topological ordering of the graph. Mp= {{Vj in union(district(Vi), parents(district(Vi))): Vj proceed Vi}}.

Usage

```
f.markov_pillow(graph, node, treatment = NULL)
```

Arguments

graph	A graph object generated by the make.graph() function.
node	A character string of a vertex for which to extract Markov pillow.
treatment	A character string specifying the treatment variable in the graph.

Value

A vector of vertices that is in the Markov pillow of the given vertex.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.markov_pillow(graph, 'A')
```

---

f.parents *Get the parents of a node OR nodes in a graph.*

Description

Function to extract the parents of a node OR nodes in a graph object. Parents of a node are the nodes that have directed edges pointing to the node.

Usage

```
f.parents(graph, nodes)
```

Arguments

graph	A graph object generated by the make.graph() function.
nodes	A character vector of nodes for which to extract parents.

Value

A vector of vertices contains parents set of the given nodes.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.parents(graph, c('Y','L'))
```

`f.reachable_closure` *Reachable closure of a set of vertices in a graph.*

Description

Function to return the reachable closure of a set of vertices in a graph object. First obtain a Conditional ADMG (CADMG) via recursively fixing as many vertices as possible in the set of all vertices (V) excluding the set of vertices specified by the `nodes` parameter (S), i.e. V \ S. The reachable closure is the subset of V \ S, where each vertex is not fixable even upon fixing other vertices.

Usage

```
f.reachable_closure(graph, nodes)
```

Arguments

`graph`	A graph object generated by the `make.graph()` function.
`nodes`	A character vector of vertices.

Value

A list containing the following components:

reachable_closure A character vector containing the reachable closure of the given vertices.

fixing_order A character vector of vertices telling the order in which the vertices were fixed.

graph The CADMG obtained via recursively fixing as many vertices as possible in the set of all vertices (V) excluding the set of vertices specified by the `nodes` parameter (S), i.e. V \ S.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.reachable_closure(graph, 'A')
```

f.top_order *Get the topological ordering of a graph from a graph object.*

Description

Wrapper around `f.top_orderMAT()` that first builds the adjacency matrix from the graph.

Usage

```
f.top_order(graph, treatment = NULL)
```

Arguments

`graph`	A graph object generated by the `make.graph()` function.
`treatment`	A character string indicating the treatment variable. If NULL, this function will rank vertices according to their input order in the vertices vector when there are ties.

Value

A vector of vertices ranked with rank from small to large.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
f.top_order(graph)
```

is.fix *Fixability of a treatment variable in a graph.*

Description

Function to check if a treatment variable is fixable in a graph object. If the treatment is fixable, then the average causal effect of the treatment on any choice of the outcome in the given graph is always identified via backdoor adjustment.

Usage

```
is.fix(graph, treatment)
```

Arguments

graph	A graph object generated by the make.graph() function.
treatment	A character string specifying the treatment variable in the graph.

Value

A logical value indicating whether the treatment is primal fixable.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
is.p.fix(graph, 'A')
```

---

is.mb.shielded *Check if a graph is mb-shielded.*

Description

Function to check if a graph is mb-shielded. A graph being mb-shielded means that the graph only implies ordinary equality constraints on the observed data distribution.

Usage

```
is.mb.shielded(graph)
```

Arguments

graph	A graph object generated by the make.graph() function.

Value

A logical value indicating whether the graph is mb-shielded.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
is.mb.shielded(graph)
```

is.np.saturated *Check if a graph is nonparametrically saturated.*

Description

Function to check if a graph is nonparametrically saturated. A graph being nonparametrically saturated means that the graph implies NO equality constraints on the observed data distribution

Usage

```
is.np.saturated(graph)
```

Arguments

graph	A graph object generated by the make.graph() function.

Value

A logical value indicating whether the graph is nonparametrically saturated.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
is.np.saturated(graph)
```

is.p.fix *Primal fixability of a treatment variable in a graph.*

Description

Function to check if a treatment variable is primal fixable in a graph object. If the treatment is primal fixable, then the average causal effect of the treatment on any choice of the outcome in the given graph is always identified.

Usage

```
is.p.fix(graph, treatment)
```

Arguments

graph	A graph object generated by the make.graph() function.
treatment	A character string specifying the treatment variable in the graph.

Value

A logical value indicating whether the treatment is primal fixable.

Examples

```
graph <- make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
is.p.fix(graph, 'A')
```

---

make.graph *Create graph object.*

Description

This function create a graph object that can be used in other functions in this package.

Usage

```
make.graph(vertices, bi_edges, di_edges, multivariate.variables = NULL)
```

Arguments

vertices	A character vector of vertices in the graph.
bi_edges	A list of vectors that record the bidirectional edges in the graph. For example, bi_edges=list(c('A','B')) means there is a bidirectional edge between vertex A and B.
di_edges	A list of vectors that record the directed edges in the graph. For example, di_edges=list(c('A','B')) means there is a directed edge from vertex A to B.
multivariate.variables	A list of variables that are multivariate in the causal graph. For example, multivariate.variables=list means M is bivariate and the corresponding columns in the dataframe are M1 and M2. Default is NULL.

Value

A graph object with the following components:

vertices Equivalent to the input argument vertices.

fixed A data frame with a column fixed that indicates whether the vertex is fixed or not. The vertices is not fixed initially.

bi_edges Equivalent to the input argument bi_edges.

di_edges Equivalent to the input argument di_edges.

multivariate.variables A list of variables that are multivariate in the causal graph. For example, multivariate.variables=list(M=c('M1,'M2')) means M is bivariate and the corresponding columns in the dataframe are M1 and M2. Default is NULL.

Examples

```
make.graph(vertices=c('A','M','L','Y','X'),
bi_edges=list(c('A','Y')),
di_edges=list(c('X','A'), c('X','M'), c('X','L'),
c('X','Y'), c('M','Y'), c('A','M'), c('A','L'), c('M','L'), c('L','Y')))
```

Index