Package ‘causaloptim’

October 31, 2023

Encoding UTF-8

Type Package

Title An Interface to Specify Causal Graphs and Compute Bounds on Causal Effects

Version 0.9.8

Date 2023-10-31

Maintainer Michael C Sachs <sachsmc@gmail.com>

Description When causal quantities are not identifiable from the observed data, it still may be possible to bound these quantities using the observed data. We outline a class of problems for which the derivation of tight bounds is always a linear programming problem and can therefore, at least theoretically, be solved using a symbolic linear optimizer. We extend and generalize the approach of Balke and Pearl (1994) <doi:10.1016/B978-1-55860-332-5.50011-0> and we provide a user friendly graphical interface for setting up such problems via directed acyclic graphs (DAG), which only allow for problems within this class to be depicted. The user can then define linear constraints to further refine their assumptions to meet their specific problem, and then specify a causal query using a text interface. The program converts this user defined DAG, query, and constraints, and returns tight bounds. The bounds can be converted to R functions to evaluate them for specific datasets, and to latex code for publication. The methods and proofs of tightness and validity of the bounds are described in a paper by Sachs, Jonzon, Gabriel, and Sjölander (2022) <doi:10.1080/10618600.2022.2071905>.

License MIT + file LICENSE

Imports methods, Rcpp (>= 1.0.1), shiny, rcdd

Depends R (>= 3.5.0), igraph

LinkingTo Rcpp

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/sachsmc/causaloptim

BugReports https://github.com/sachsmc/causaloptim/issues

Config/testthat/edition 3

NeedsCompilation yes

Author Michael C Sachs [aut, cre],
Erin E Gabriel [aut],
Arvid Sjölander [aut],
Gustav Jonzon [aut],
Alexander A Balke [ctb] ((C++ code)),
Colorado Reed [ctb] ((graph-creator.js))

Repository CRAN

Date/Publication 2023-10-31 14:30:02 UTC

R topics documented:

causaloptim-package *An Interface to Specify Causal Graphs and Compute Bounds on Causal Effects*

Description

Specify causal graphs using a visual interactive interface and then analyze them and compute symbolic bounds for the causal effects in terms of the observable parameters.

Details

Run the shiny app by `results <- specify_graph()`. See detailed instructions in the vignette `browseVignettes("causaloptim")`.

Author(s)

Michael C Sachs, Arvid Sjölander, Gustav Jonzon, Alexander Balke, Colorado Reed, and Erin Gabriel Maintainer: Michael C Sachs <sachsmc at gmail.com>

References

Sachs, M. C., Jonzon, G., Sjölander, A., & Gabriel, E. E. 2022, A general method for deriving tight symbolic bounds on causal effects. Journal of Computational and Graphical Statistics, In press. https://www.tandfonline.com/doi/full/10.1080/10618600.2022.2071905 .

See Also

`browseVignettes('causaloptim')` specify_graph

analyze_graph *Analyze the causal graph and effect to determine constraints and objective*

Description

The graph must contain certain edge and vertex attributes which are documented in the Details below. The shiny app run by specify_graph will return a graph in this format.

Usage

```
analyze_graph(graph, constraints, effectt)
```

Arguments

`graph`	An aaa-igraph-package object that represents a directed acyclic graph with certain attributes. See Details.
`constraints`	A vector of character strings that represent the constraints on counterfactual quantities
`effectt`	A character string that represents the causal effect of interest

Details

The graph object must contain the following named vertex attributes:

name The name of each vertex must be a valid R object name starting with a letter and no special characters. Good candidate names are for example, Z1, Z2, W2, X3, etc.

leftside An indicator of whether the vertex is on the left side of the graph, 1 if yes, 0 if no.

latent An indicator of whether the variable is latent (unobserved). There should always be a variable Ul on the left side that is latent and a parent of all variables on the left side, and another latent variable Ur on the right side that is a parent of all variables on the right side.

nvals The number of possible values that the variable can take on, the default and minimum is 2 for 2 categories (0,1). In general, a variable with nvals of K can take on values 0, 1, ..., (K-1).

In addition, there must be the following edge attributes:

rlconnect An indicator of whether the edge goes from the right side to the left side. Should be 0 for all edges.

edge.monotone An indicator of whether the effect of the edge is monotone, meaning that if V1 -> V2 and the edge is monotone, then a > b implies V2(V1 = a) >= V2(V1 = b). Only available for binary variables (nvals = 2).

The effectt parameter describes your causal effect of interest. The effectt parameter must be of the form

`p{V11(X=a)=a; V12(X=a)=b;...} op1 p{V21(X=b)=a; V22(X=c)=b;...} op2 ...`

where Vij are names of variables in the graph, a, b are numeric values from 0:(nvals - 1), and op are either - or +. You can specify a single probability statement (i.e., no operator). Note that the probability statements begin with little p, and use curly braces, and items inside the probability statements are separated by ;. The variables may be potential outcomes which are denoted by parentheses. Variables may also be nested inside potential outcomes. Pure observations such as `p{Y = 1}` are not allowed if the left side contains any variables. There are 2 important rules to follow: 1) Only variables on the right side can be in the probability events, and if the left side is not empty: 2) none of the variables in the left side that are intervened upon can have any children in the left side, and all paths from the left to the right must be blocked by the intervention set. Here the intervention set is anything that is inside the smooth brackets (i.e., variable set to values).

All of the following are valid effect statements:

`p{Y(X = 1) = 1} - p{Y(X = 0) = 1}`

`p{X(Z = 1) = 1; X(Z = 0) = 0}`

`p{Y(M(X = 0), X = 1) = 1} - p{Y(M(X = 0), X = 0) = 1}`

The constraints are specified in terms of potential outcomes to constrain by writing the potential outcomes, values of their parents, and operators that determine the constraint (equalities or inequalities). For example, `X(Z = 1) >= X(Z = 0)`

Value

A an object of class "linearcausalproblem", which is a list with the following components. This list can be passed to optimize_effect_2 which interfaces with the symbolic optimization program. Print and plot methods are also available.

variables Character vector of variable names of potential outcomes, these start with 'q' to match Balke's notation

parameters Character vector of parameter names of observed probabilities, these start with 'p' to match Balke's notation

constraints Character vector of parsed constraints

objective Character string defining the objective to be optimized in terms of the variables

p.vals Matrix of all possible values of the observed data vector, corresponding to the list of parameters.

q.vals Matrix of all possible values of the response function form of the potential outcomes, corresponding to the list of variables.

parsed.query A nested list containing information on the parsed causal query.

objective.nonreduced The objective in terms of the original variables, before algebraic variable reduction. The nonreduced variables can be obtained by concatenating the columns of q.vals.

response.functions List of response functions.

graph The graph as passed to the function.

R A matrix with coefficients relating the p.vals to the q.vals p = R * q

c0 A vector of coefficients relating the q.vals to the objective function theta = c0 * q

iqR A matrix with coefficients to represent the inequality constraints

Examples

```
### confounded exposure and outcome
b <- igraph::graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
```

causalproblemcheck *Check conditions on causal problem*

Description

Check that a given causal problem (a causal DAG together with a causal query) satisfies conditions that guarantee that the optimization problem is linear.

Usage

```
causalproblemcheck(digraph, query)
```

Arguments

`digraph`	An `igraph` object representing a digraph. Expected vertex attributes: `leftside`, `latent` and `nvals`. Optional vertex attributes: `exposure` and `outcome`. Expected edge attributes: `rlconnect` and `edge.monotone`.
`query`	A string representing a causal query / effect.

Value

`TRUE` if conditions are met; `FALSE` otherwise.

Examples

```
b <- graph_from_literal(X - +Y, Ur - +X, Ur - +Y)
V(b)$leftside <- c(0, 0, 0)
V(b)$latent <- c(0, 0, 1)
V(b)$nvals <- c(2, 2, 2)
V(b)$exposure <- c(1, 0, 0)
V(b)$outcome <- c(0, 1, 0)
E(b)$rlconnect <- c(0, 0, 0)
E(b)$edge.monotone <- c(0, 0, 0)
effectt <- "p{Y(X=1)=1}-p{Y(X=0)=1}"
causalproblemcheck(digraph = b, query = effectt)
```

constraintscheck *Check constraints*

Description

Check that a user-provided constraint is parsable, has valid variables and relations.

Usage

```
constraintscheck(constrainttext, graphres)
```

Arguments

constrainttext	A string representing a constraint.
graphres	An igraph object representing a DAG.

Value

TRUE if all check pass; else FALSE.

Examples

```
graphres <- graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y)
V(graphres)$leftside <- c(1, 0, 0, 1, 0)
V(graphres)$latent <- c(0, 0, 0, 1, 1)
V(graphres)$nvals <- c(3, 2, 2, 2, 2)
V(graphres)$exposure <- c(0, 1, 0, 0, 0)
V(graphres)$outcome <- c(0, 0, 1, 0, 0)
E(graphres)$rlconnect <- c(0, 0, 0, 0, 0)
E(graphres)$edge.monotone <- c(0, 0, 0, 0, 0)
constrainttext <- "X(Z = 1) >= X(Z = 0)"
constraintscheck(constrainttext = constrainttext, graphres = graphres) # TRUE
```

create_effect_vector *Translate target effect to vector of response variables*

Description

Translate target effect to vector of response variables

Usage

```
create_effect_vector(effect, graph, obsvars, respvars, q.list, variables)
```

Arguments

`effect`	Effect list, as returned by parse_effect
`graph`	The graph
`obsvars`	Vector of observed variable vertices from the graph
`respvars`	Response function, as returned by create_response_function
`q.list`	List with q matrices, as returned by create_q_matrix
`variables`	Vector of qs names

Value

A list with the target effect in terms of qs

Examples

```
graph <- graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y)
V(graph)$leftside <- c(1, 0, 0, 1, 0)
V(graph)$latent <- c(0, 0, 0, 1, 1)
V(graph)$nvals <- c(3, 2, 2, 2, 2)
V(graph)$exposure <- c(0, 1, 0, 0, 0)
V(graph)$outcome <- c(0, 0, 1, 0, 0)
E(graph)$rlconnect <- c(0, 0, 0, 0, 0)
E(graph)$edge.monotone <- c(0, 0, 0, 0, 0)
constraints <- "X(Z = 1) >= X(Z = 0)"
effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
leftind <- vertex_attr(graph)$leftside
cond.vars <- V(graph)[leftind == 1 & names(V(graph)) != "Ul"]
right.vars <- V(graph)[leftind == 0 & names(V(graph)) != "Ur"]
obsvars <- c(right.vars, cond.vars)
observed.variables <- V(graph)[V(graph)$latent == 0]
var.values <- lapply(names(observed.variables),
function(varname) seq(from = 0, to = causaloptim:::numberOfValues(graph, varname) - 1))
names(var.values) <- names(observed.variables)
p.vals <- do.call(expand.grid, var.values)
jd <- do.call(paste0, p.vals[, names(right.vars[right.vars$latent == 0]), drop = FALSE])
cond <- do.call(paste0, p.vals[, names(cond.vars[cond.vars$latent == 0]), drop = FALSE])
parameters <- paste0("p", paste(jd, cond, sep = "_"))
parameters.key <- paste(paste(names(right.vars[right.vars$latent == 0]), collapse = ""),
paste(names(cond.vars[cond.vars$latent == 0]), collapse = ""), sep = "_")
respvars <- create_response_function(graph, right.vars, cond.vars)
q.list <- create_q_matrix(respvars, right.vars, cond.vars, constraints)
variables <- as.character(unique(q.list$q.vals.all.lookup$vars))
linconstr.list <- create_R_matrix(graph, obsvars, respvars, p.vals, parameters, q.list, variables)
parameters <- linconstr.list$newparams
p.vals <- linconstr.list$newpvals
effect <- parse_effect(effectt)
var.eff <- create_effect_vector(effect, graph, obsvars, respvars, q.list, variables)
```

create_q_matrix — *Translate response functions into matrix of counterfactuals*

Description

Translate response functions into matrix of counterfactuals

Usage

```
create_q_matrix(respvars, right.vars, cond.vars, constraints)
```

Arguments

respvars	A list of functions as returned by create_response_function
right.vars	Vertices of graph on the right side
cond.vars	Vertices of graph on the left side
constraints	A vector of character strings that represent the constraints

Value

A list of 3 data frames of counterfactuals and their associated labels

Examples

```
graphres <- graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y)
V(graphres)$leftside <- c(1, 0, 0, 1, 0)
V(graphres)$latent <- c(0, 0, 0, 1, 1)
V(graphres)$nvals <- c(3, 2, 2, 2, 2)
V(graphres)$exposure <- c(0, 1, 0, 0, 0)
V(graphres)$outcome <- c(0, 0, 1, 0, 0)
E(graphres)$rlconnect <- c(0, 0, 0, 0, 0)
E(graphres)$edge.monotone <- c(0, 0, 0, 0, 0)
constraints <- "X(Z = 1) >= X(Z = 0)"
cond.vars <- V(graphres)[V(graphres)$leftside == 1 & names(V(graphres)) != "Ul"]
right.vars <- V(graphres)[V(graphres)$leftside == 0 & names(V(graphres)) != "Ur"]
respvars <- create_response_function(graphres, right.vars, cond.vars)
create_q_matrix(respvars, right.vars, cond.vars, constraints)
```

`create_response_function` *Translate regular DAG to response functions*

Description

Translate regular DAG to response functions

Usage

```
create_response_function(graph, right.vars, cond.vars)
```

Arguments

`graph`	An aaa-igraph-package object that represents a directed acyclic graph that contains certain edge attributes. The shiny app returns a graph in this format and see examples.
`right.vars`	Vertices of graph on the right side
`cond.vars`	Vertices of graph on the left side

Value

A list of functions representing the response functions

Examples

```
### confounded exposure and outcome
b <- igraph::graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
cond.vars <- V(b)[V(b)$leftside == 1 & names(V(b)) != "Ul"]
right.vars <- V(b)[V(b)$leftside == 0 & names(V(b)) != "Ur"]
create_response_function(b, right.vars, cond.vars)
```

`create_R_matrix` *Create constraint matrix*

Description

Matrix and text representation of constraints on observed probabilities

Usage

```
create_R_matrix(
  graph,
  obsvars,
  respvars,
  p.vals,
  parameters,
  q.list,
  variables
)
```

Arguments

`graph`	The graph
`obsvars`	Vector of observed variable vertices from the graph
`respvars`	Response function, as returned by create_response_function
`p.vals`	Observed probability matrix
`parameters`	Vector of ps names
`q.list`	List with q matrices, as returned by create_q_matrix
`variables`	Vector of qs names

Value

A list with the R matrix and the string representation

Examples

```
graph <- graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y)
V(graph)$leftside <- c(1, 0, 0, 1, 0)
V(graph)$latent <- c(0, 0, 0, 1, 1)
V(graph)$nvals <- c(3, 2, 2, 2, 2)
V(graph)$exposure <- c(0, 1, 0, 0, 0)
V(graph)$outcome <- c(0, 0, 1, 0, 0)
E(graph)$rlconnect <- c(0, 0, 0, 0, 0)
E(graph)$edge.monotone <- c(0, 0, 0, 0, 0)
constraints <- "X(Z = 1) >= X(Z = 0)"
effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
leftind <- vertex_attr(graph)$leftside
cond.vars <- V(graph)[leftind == 1 & names(V(graph)) != "Ul"]
right.vars <- V(graph)[leftind == 0 & names(V(graph)) != "Ur"]
obsvars <- c(right.vars, cond.vars)
observed.variables <- V(graph)[V(graph)$latent == 0]
var.values <- lapply(names(observed.variables),
function(varname) seq(from = 0, to = causaloptim:::numberOfValues(graph, varname) - 1))
names(var.values) <- names(observed.variables)
p.vals <- do.call(expand.grid, var.values)
jd <- do.call(paste0, p.vals[, names(right.vars[right.vars$latent == 0]), drop = FALSE])
cond <- do.call(paste0, p.vals[, names(cond.vars[cond.vars$latent == 0]), drop = FALSE])
parameters <- paste0("p", paste(jd, cond, sep = "_"))
```

```
parameters.key <- paste(paste(names(right.vars[right.vars$latent == 0]), collapse = ""),
paste(names(cond.vars[cond.vars$latent == 0]), collapse = ""), sep = "_")
respvars <- create_response_function(graph, right.vars, cond.vars)
q.list <- create_q_matrix(respvars, right.vars, cond.vars, constraints)
variables <- as.character(unique(q.list$q.vals.all.lookup$vars))
linconstr.list <- create_R_matrix(graph, obsvars, respvars, p.vals, parameters, q.list, variables)
```

get_default_effect *Define default effect for a given graph*

Description

Define default effect for a given graph

Usage

```
get_default_effect(graphres)
```

Arguments

graphres	The graph object, should have vertex attributes "outcome" and "exposure"

Value

A string that can be passed to parse_effect

Examples

```
graphres <- graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y)
V(graphres)$leftside <- c(1, 0, 0, 1, 0)
V(graphres)$latent <- c(0, 0, 0, 1, 1)
V(graphres)$nvals <- c(3, 2, 2, 2, 2)
V(graphres)$exposure <- c(0, 1, 0, 0, 0)
V(graphres)$outcome <- c(0, 0, 1, 0, 0)
E(graphres)$rlconnect <- c(0, 0, 0, 0, 0)
E(graphres)$edge.monotone <- c(0, 0, 0, 0, 0)
get_default_effect(graphres = graphres) == "p{Y(X = 1)=1} - p{Y(X = 0)=1}" # TRUE
```

graphrescheck — *Check conditions on digraph*

Description

Check that a given digraph satisfies the conditions of 'no left to right edges', 'no cycles', 'valid number of categories' and 'valid variable names'. Optionally returns the digraph if all checks are passed.

Usage

```
graphrescheck(graphres, ret = FALSE)
```

Arguments

graphres	An igraph object representing a digraph.
ret	A logical value. Default is FALSE. Set to TRUE to also return graphres if all checks are passed.

Value

If ret=FALSE (default): TRUE if all checks pass; else FALSE. If ret=TRUE: graphres if all checks pass; else FALSE.

Examples

```
graphres <- graph_from_literal(X -+ Y, X -+ M, M -+ Y, Ul -+ X, Ur -+ M, Ur -+ Y)
V(graphres)$leftside <- c(1, 0, 0, 1, 0)
V(graphres)$latent <- c(0, 0, 0, 1, 1)
V(graphres)$nvals <- c(2, 2, 2, 2, 2)
V(graphres)$exposure <- c(0, 0, 0, 0, 0)
V(graphres)$outcome <- c(0, 0, 0, 0, 0)
E(graphres)$rlconnect <- c(0, 0, 0, 0, 0, 0)
E(graphres)$edge.monotone <- c(0, 0, 0, 0, 0, 0)
graphrescheck(graphres = graphres) # TRUE
```

interpret_bounds — *Convert bounds string to a function*

Description

Convert bounds string to a function

Usage

```
interpret_bounds(bounds, parameters)
```

Arguments

bounds	The bounds element as returned by optimize_effect
parameters	Character vector defining parameters, as returned by analyze_graph

Value

A function that takes arguments for the parameters, i.e., the observed probabilities and returns a vector of length 2: the lower bound and the upper bound.

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect(obj)
bounds_func <- interpret_bounds(bounds$bounds, obj$parameters)
bounds_func(.1, .1, .4, .3)
# vectorized
do.call(bounds_func, lapply(1:4, function(i) runif(5)))
```

---

latex_bounds *Latex bounds equations*

Description

Latex bounds equations

Usage

```
latex_bounds(bounds, parameters, prob.sym = "P", brackets = c("(", ")"))
```

Arguments

bounds	Vector of bounds as returned by optimize_effect
parameters	The parameters object as returned by analyze_graph
prob.sym	Symbol to use for probability statements in latex, usually "P" or "pr"
brackets	Length 2 vector with opening and closing bracket, usually `c("(", ")")`, or `c("\{", "\}")`

Value

A character string with latex code for the bounds

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect(obj)
latex_bounds(bounds$bounds, obj$parameters)
latex_bounds(bounds$bounds, obj$parameters, "Pr")
```

---

optimize_effect *Run the Balke optimizer*

Description

Given a object with the linear programming problem set up, compute the bounds using the c++ code developed by Alex Balke. Bounds are returned as text but can be converted to R functions using interpret_bounds, or latex code using latex_bounds. This legacy function has been replaced as default with the more efficient optimize_effect_2.

Usage

```
optimize_effect(obj)
```

Arguments

obj	Object as returned by analyze_graph

Value

An object of class "balkebound" that contains the bounds and logs as character strings

See Also

optimize_effect_2 which is the current default

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
optimize_effect(obj)
```

optimize_effect_2 *Run the optimizer*

Description

Given an object with the linear programming problem set up, compute the bounds using rcdd. Bounds are returned as text but can be converted to R functions using interpret_bounds, or latex code using latex_bounds.

Usage

```
optimize_effect_2(obj)
```

Arguments

obj	Object as returned by analyze_graph

Value

An object of class "balkebound" that contains the bounds and logs as character strings

See Also

optimize_effect which is a legacy version

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
optimize_effect_2(obj)
```

opt_effect *Compute a bound on the average causal effect*

Description

This helper function does the heavy lifting for `optimize_effect_2`. For a given casual query, it computes either a lower or an upper bound on the corresponding causal effect.

Usage

```
opt_effect(opt, obj)
```

Arguments

`opt`	A string. Either `"min"` or `"max"` for a lower or an upper bound, respectively.
`obj`	An object as returned by the function `analyze_graph`. Contains the casual query to be estimated.

Value

An object of class `optbound`; a list with the following named components:

- `expr` is the *main* output; an expression of the bound as a print-friendly string,
- `type` is either `"lower"` or `"upper"` according to the type of the bound,
- `dual_vertices` is a numeric matrix whose rows are the vertices of the convex polytope of the dual LP,
- `dual_vrep` is a V-representation of the dual convex polytope, including some extra data.

`parse_constraints` *Parse text that defines a the constraints*

Description

Parse text that defines a the constraints

Usage

```
parse_constraints(constraints, obsnames)
```

Arguments

`constraints`	A list of character strings
`obsnames`	Vector of names of the observed variables in the graph

Value

A data frame with columns indicating the variables being constrained, what the values of their parents are for the constraints, and the operator defining the constraint (equality or inequalities).

Examples

```
constrainttext <- "X(Z = 1) >= X(Z = 0)"
obsnames <- c("Z", "X", "Y")
parse_constraints(constraints = constrainttext, obsnames = obsnames)
```

parse_effect — *Parse text that defines a causal effect*

Description

Parse text that defines a causal effect

Usage

```
parse_effect(text)
```

Arguments

`text`	Character string

Value

A nested list that contains the following components:

- **vars** For each element of the causal query, this indicates potential outcomes as names of the list elements, the variables that they depend on, and the values that any variables are being fixed to.
- **oper** The vector of operators (addition or subtraction) that combine the terms of the causal query.
- **values** The values that the potential outcomes are set to in the query.
- **pcheck** List of logicals for each element of the query that are TRUE if the element is a potential outcome and FALSE if it is an observational quantity.

Examples

```
effectt <- "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
parse_effect(text = effectt)
```

plot.linearcausalproblem — *Plot the graph from the causal problem with a legend describing attributes*

Description

Plot the graph from the causal problem with a legend describing attributes

Usage

```
## S3 method for class 'linearcausalproblem'
plot(x, ...)
```

Arguments

`x`	object of class "linearcausalproblem"
`...`	Not used

Value

Nothing

See Also

plot_graphres which plots the graph only

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
V(b)$exposure <- c(1,0,0)
V(b)$outcome <- c(0,1,0)
E(b)$rlconnect <- c(0,0,0)
E(b)$edge.monotone <- c(0,0,0)
q <- "p{Y(X=1)=1}-p{Y(X=0)=1}"
obj <- analyze_graph(graph = b, constraints = NULL, effectt <- q)
plot(obj)
```

`plot_graphres` *Plot the analyzed graph object*

Description

Special plotting method for igraphs of this type

Usage

```
plot_graphres(graphres)
```

Arguments

`graphres`	an igraph object

Value

None

See Also

plot.linearcausalproblem which plots a graph with attributes

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
V(b)$exposure <- c(1,0,0)
V(b)$outcome <- c(0,1,0)
E(b)$rlconnect <- c(0,0,0)
E(b)$edge.monotone <- c(0,0,0)
plot(b)
```

`print.linearcausalproblem` *Print the causal problem*

Description

Print the causal problem

Usage

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

Arguments

`x`	object of class "linearcausaloptim"
`...`	Not used

Value

x, invisibly

`querycheck` *Check conditions on query*

Description

Given an admissible causal DAG, check that given a causal query satisfies conditions that guarantee the corresponding causal problem to be a linear program. Throws error messages detailing any conditions violated.

Usage

```
querycheck(effecttext, graphres)
```

Arguments

effecttext	A string representing a causal query.
graphres	An igraph object representing a digraph.

Value

TRUE if effecttext is parsable, contains only variables in V(graphres) and satisfies conditions for linearity; else FALSE.

Examples

```
graphres <- graph_from_literal(X -+ Y, X -+ M, M -+ Y, Ul -+ X, Ur -+ M, Ur -+ Y)
V(graphres)$leftside <- c(1, 0, 0, 1, 0)
V(graphres)$latent <- c(0, 0, 0, 1, 1)
V(graphres)$nvals <- c(2, 2, 2, 2, 2)
V(graphres)$exposure <- c(0, 0, 0, 0, 0)
V(graphres)$outcome <- c(0, 0, 0, 0, 0)
E(graphres)$rlconnect <- c(0, 0, 0, 0, 0, 0)
E(graphres)$edge.monotone <- c(0, 0, 0, 0, 0, 0)
effecttext <- "p{Y(M(X = 0), X = 1) = 1} - p{Y(M(X = 0), X = 0) = 1}"
causaloptim:::querycheck(effecttext = effecttext, graphres = graphres) # TRUE
```

---

simulate_bounds *Simulate bounds*

Description

Run a simple simulation based on the bounds. For each simulation, sample the set of counterfactual probabilities from a uniform distribution, translate into a multinomial distribution, and then compute the objective and the bounds in terms of the observable variables.

Usage

```
simulate_bounds(obj, bounds, nsim = 1000)
```

Arguments

obj	Object as returned by analyze_graph
bounds	Object as returned by optimize_effect
nsim	Number of simulation replicates

Value

A data frame with columns: objective, bound.lower, bound.upper

Examples

```
b <- graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y)
V(b)$leftside <- c(0,0,0)
V(b)$latent <- c(0,0,1)
V(b)$nvals <- c(2,2,2)
E(b)$rlconnect <- E(b)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect(obj)
simulate_bounds(obj, bounds, nsim = 5)
```

---

specify_graph *Shiny interface to specify network structure and compute bounds*

Description

This launches the Shiny interface in the system's default web browser. The results of the computation will be displayed in the browser, but they can also be returned to the R session by assigning the result of the function call to an object. See below for information on what is returned.

Usage

```
specify_graph()
```

Value

If the button "Exit and return graph object" is clicked, then only the graph is returned as an aaa-igraph-package object.

If the bounds are computed and the button "Exit and return objects to R" is clicked, then a list is returned with the following elements:

graphres The graph as drawn and interpreted, an aaa-igraph-package object.

obj The objective and all necessary supporting information. This object is documented in analyze_graph. This can be passed directly to optimize_effect_2.

bounds.obs Object of class 'balkebound' as returned by optimize_effect_2.

constraints Character vector of the specified constraints. NULL if no constraints.

effect Text describing the causal effect of interest.

boundsFunction Function that takes parameters (observed probabilities) as arguments, and returns a vector of length 2 for the lower and upper bounds.

update_effect *Update the effect in a linearcausalproblem object*

Description

If you want to use the same graph and response function, but change the effect of interest, this can save some computation time.

Usage

```
update_effect(obj, effectt)
```

Arguments

obj	An object as returned by analyze_graph
effectt	A character string that represents the causal effect of interest

Value

A object of class linearcausalproblem, see analyze_graph for details

Examples

```
b <- igraph::graph_from_literal(X -+ Y, X -+ M, M -+ Y, Ul -+ X, Ur -+ Y, Ur -+ M)
V(b)$leftside <- c(1, 0, 0, 1, 0)
V(b)$latent <- c(0, 0, 0, 1, 1)
V(b)$nvals <- c(2, 2, 2, 2, 2)
E(b)$rlconnect <- c(0, 0, 0, 0, 0, 0)
E(b)$edge.monotone <- c(0, 0, 0, 0, 0, 0)
CDE0_query <- "p{Y(M = 0, X = 1) = 1} - p{Y(M = 0, X = 0) = 1}"
CDE0_obj <- analyze_graph(b, constraints = NULL, effectt = CDE0_query)
CDE0_bounds <- optimize_effect_2(CDE0_obj)
CDE0_boundsfunction <- interpret_bounds(bounds = CDE0_bounds$bounds,
parameters = CDE0_obj$parameters)
CDE1_query <- "p{Y(M = 1, X = 1) = 1} - p{Y(M = 1, X = 0) = 1}"
CDE1_obj <- update_effect(CDE0_obj, effectt = CDE1_query)
CDE1_bounds <- optimize_effect_2(CDE1_obj)
CDE1_boundsfunction <- interpret_bounds(bounds = CDE1_bounds$bounds,
parameters = CDE1_obj$parameters)
```

Index