Package 'causaloptim'

October 17, 2024

Encoding UTF-8

Type Package

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

Version 1.0.0

Date 2024-10-16

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 shiny, rcdd

Depends R (>= 3.5.0), igraph

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

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

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

Config/testthat/edition 3

NeedsCompilation no

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 2024-10-17 09:00:02 UTC

Contents

`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. (2023). A general method for deriving tight symbolic bounds on causal effects. Journal of Computational and Graphical Statistics, 32(2), 567-576. 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 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 <- initialize_graph(igraph::graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y))
analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
```

---

btm_var *Recursive function to get the last name in a list*

Description

Recursive function to get the last name in a list

Usage

```
btm_var(x, name = NULL)
```

Arguments

x	a list
name	name of the top element of the list

Value

The name of the deepest nested list element

Examples

```
btm_var(list(X = list(Y = list(K = 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)
```

`check_constraints_violated` *Check whether any of the observable constraints implied by the causal model are violated for a given distribution of observables*

Description

Check whether any of the observable constraints implied by the causal model are violated for a given distribution of observables

Usage

```
check_constraints_violated(obj, probs, tol = 1e-12)
```

Arguments

`obj`	An object of class "causalmodel"
`probs`	A named vector of observable probabilities, in the same order as obj$data$parameters
`tol`	Tolerance for checking (in)equalities

Value

Either TRUE (violated) or FALSE (not violated) with messages indicating what constraints are violated if any.

Examples

```
graph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ur -+ X, Ur -+ Y))

iv_model <- create_causalmodel(graph, prob.form = list(out = c("X", "Y"), cond = "Z"))
check_constraints_violated(iv_model, probs = sample_distribution(iv_model))
```

check_linear_objective *Check linearity of objective function implied by a causal model and effect*

Description

Check linearity of objective function implied by a causal model and effect

Usage

```
check_linear_objective(causal_model, effectt)
```

Arguments

causal_model	An object of class "causalmodel" as produce by create_causalmodel
effectt	A character string that represents the causal effect of interest

Details

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.

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 effect must be fully specified, that is, all parents of a variable that is intervened upon need to be specified. The function cannot infer missing values or marginalize over some parents but not others.

Value

A logical value that is TRUE if the objective function is linear

Examples

```
 ## regular IV case

ivgraph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ur -+ X, Ur -+ Y))
prob.form <- list(out = c("Y", "X"), cond = "Z")

iv_model <- create_causalmodel(graph = ivgraph,
            prob.form = prob.form)
check_linear_objective(iv_model, effectt = "p{Y(X = 1) = 1}")

#'  ## contaminated IV case

civgraph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Z-+ Y, Ur -+ X, Ur -+ Y))

cont_iv <- create_causalmodel(graph = civgraph, prob.form = prob.form)

check_linear_objective(cont_iv, effectt = "p{Y(X = 1) = 1}")
```

---

check_parents *Check for paths from from to to*

Description

Check for paths from from to to

Usage

```
check_parents(parent_lookup, from, to, prev = NULL)
```

Arguments

parent_lookup	A list of vectors
from	character
to	character
prev	Should always be null when first called

Value

A list of paths or null if no path is found

Examples

```
parent_lookup <- list(M = "Am", Y = c("M", "Ay"), A = NULL, Am = "A", Ay = "A")
check_parents(parent_lookup, "A", "Y")
```

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_causalmodel *Create a structural causal model from a graph or a set of response functions*

Description

Given either a graph or a set of response functions (i.e., either graph or respvars may be provided), and a specification of what conditional probabilities are observed, produce a causal model.

Usage

```
create_causalmodel(
  graph = NULL,
  respvars = NULL,
  prob.form,
  p.vals,
  constraints = NULL,
  right.vars = NULL
)
```

Arguments

graph	A graph with special edge and vertex attributes, as produced by initialize_graph
respvars	List of response functions as produced by create_response_function
prob.form	A list with two named elements "out", "cond" where each element is a character vector of variable names that appear in p.vals
p.vals	Data frame defining which probabilities are observable. The variable names of p.vals must all appear in prob.form. If missing, will assume that all combinations of the variables values are observed.
constraints	A vector of character strings that represent the constraints on counterfactual quantities
right.vars	A vector of character strings indicating which variables are on the right side of the graph. Only required when graph is NULL. See examples.

Details

It is assumed that probabilities of the form p(out | cond) are observed, for each combination of values in p.vals. cond may be NULL in which case nothing is conditioned on.

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

An object of class "causalmodel"

Examples

```
 ## regular IV case

graph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ur -+ X, Ur -+ Y))

iv_model <- create_causalmodel(graph, prob.form = list(out = c("X", "Y"), cond = "Z"))
# with monotonicity
iv_model_mono <- create_causalmodel(graph, prob.form = list(out = c("X", "Y"), cond = "Z"),
                 constraints = list("X(Z = 1) >= X(Z = 0)"))

#showing the use of right.vars
```

```
b <- initialize_graph(graph_from_literal(Ul -+ X -+ Y -+ Y2, Ur -+ Y, Ur -+ Y2))
V(b)$latent <- c(1, 0, 1, 0, 1)
respvars <- create_response_function(b)
create_causalmodel(graph = b, constraints = "Y2(Y = 1) >= Y2(Y = 0)",
                   p.vals = expand.grid(X = 0:1, Y2 = 0:1),
                   prob.form = list(out = "Y2", cond = "X"))
 ## need to specify right.vars because it cannot be inferred from the response functions alone
create_causalmodel(graph = NULL, respvars = respvars,
                   constraints = "Y2(Y = 1) >= Y2(Y = 0)",
                   p.vals = expand.grid(X = 0:1, Y2 = 0:1),
                   prob.form = list(out = "Y2", cond = "X"),
                   right.vars = c("Y", "Y2"))
```

`create_effect_vector` *Translate target effect to vector of response variables*

Description

Translate target effect to vector of response variables

Usage

```
create_effect_vector(causal_model, effect)
```

Arguments

`causal_model`	An object of class "causalmodel" as produced by create_causalmodel
`effect`	Effect list, as returned by parse_effect

Value

A list with the target effect in terms of qs

Examples

```
graph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y))
constraints <- "X(Z = 1) >= X(Z = 0)"
effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
p.vals <- expand.grid(Z = 0:1, X = 0:1, Y = 0:1)
prob.form <- list(out = c("X", "Y"), cond = "Z")
effect <- parse_effect(effectt)
ivmod <- create_causalmodel(graph, respvars = NULL, p.vals = p.vals, prob.form = prob.form,
         constraints = constraints)
var.eff <- create_effect_vector(ivmod, effect)
```

`create_linearcausalproblem` *Create linear causal problem from causal model and effect*

Description

A more flexible alternative to analyze_graph that takes as inputs the causal model and effect.

Usage

```
create_linearcausalproblem(causal_model, effectt)
```

Arguments

`causal_model`	An object of class "causalmodel" as produce by create_causalmodel
`effectt`	A character string that represents the causal effect of interest

Details

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}`

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 <- initialize_graph(igraph::graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y))
confmod <- create_causalmodel(graph = b, prob.form =  list(out = c("X", "Y"), cond = NULL))
create_linearcausalproblem(confmod, effectt = "p{Y(X = 1) = 1}")
```

`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 <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ul -+ Z, Ur -+ X, Ur -+ Y))
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)
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)
```

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 initialize_graph will add them to a regular igraph object with sensible defaults.

Value

A list of functions representing the response functions

Examples

```
### confounded exposure and outcome
b <- initialize_graph(igraph::graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y))
create_response_function(b)
```

find_all_paths *Find all paths in a causal model*

Description

Given a set of response functions, find all directed paths from from to to

Usage

```
find_all_paths(respvars, from, to)
```

Arguments

respvars	A set of response functions as created by create_response_function
from	A character string indicating the start of the path
to	A character string indicating the end of the path

Value

A list with all the paths or a list with NULL if there are none

Examples

```
b <- initialize_graph(igraph::graph_from_literal(X -+ Z, Z -+ Y, X -+ Y, Ur -+ Z, Ur -+ Y))
medmod <- create_response_function(b)
find_all_paths(medmod, "X", "Y")
igraph::all_simple_paths(b, "X", "Y", mode = "out")
```

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
```

initialize_graph *Initialize an igraph object for use with causaloptim*

Description

Checks for required attributes and adds defaults if missing

Usage

```
initialize_graph(graph)
```

Arguments

graph	An object of class igraph

Value

An igraph with the vertex attributes leftside, latent, and nvals, and edge attributes rlconnect and edge.monotone

Examples

```
b <- igraph::graph_from_literal(X -+ Y)
b2 <- initialize_graph(b)
V(b2)$nvals
```

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)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect_2(obj)
bounds_func <- interpret_bounds(bounds$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_2
`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)$edge.monotone <- c(0, 0, 0)
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect_2(obj)
latex_bounds(bounds$bounds, obj$parameters)
latex_bounds(bounds$bounds, obj$parameters, "Pr")
```

list_to_path *Recursive function to translate an effect list to a path sequence*

Description

Recursive function to translate an effect list to a path sequence

Usage

```
list_to_path(x, name = NULL)
```

Arguments

x	A list of vars as returned by parse_effect
name	The name of the outcome variable

Value

a list of characters describing the path sequence

Examples

```
nofill <- "p{Y(X = 1, M1 = 1, M2(X = 1, M1 = 1)) = 1}"
eff2 <- parse_effect(nofill)$vars[[1]][[1]]
list_to_path(eff2, "Y")
```

optimize_effect_2 *Run the optimizer to obtain symbolic bounds*

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 or create_linearcausalproblem

Value

An object of class "balkebound" that is a list that contains the bounds and logs as character strings, and a function to compute the bounds

Examples

```
b <- initialize_graph(graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y))
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.causalmodel *Print relevant information about the causal model*

Description

Print relevant information about the causal model

Usage

```
## S3 method for class 'causalmodel'
print(x, omit_cf_constraints = FALSE, omit_obs_constraints = FALSE, ...)
```

Arguments

x	object of class "causalmodel"
omit_cf_constraints	Do not print the counterfactual constraints
omit_obs_constraints	Do not print the observable constraints
...	Not used

Value

x, invisibly

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}"
querycheck(effecttext = effecttext, graphres = graphres) # TRUE
```

rdirichlet *Sample from a Dirichlet distribution*

Description

Generate a random vector from the k-dimensional symmetric Dirichlet distribution with concentration parameter alpha

Usage

```
rdirichlet(k, alpha = 1)
```

Arguments

k	Length of the vector
alpha	Concentration parameters

Value

a numeric vector

Examples

```
qvals <- rdirichlet(16, 1)
sum(qvals)
```

---

sample_distribution *Sample a distribution of observable probabilities that satisfy the causal model*

Description

Sample a distribution of observable probabilities that satisfy the causal model

Usage

```
sample_distribution(
  obj,
  simplex_sampler = function(k) {
     rdirichlet(k, alpha = 1)
 }
)
```

Arguments

obj	An object of class "causalmodel"
simplex_sampler	A function to generate a random sample from the simplex in k dimensions, where k is the number of variables (q parameters, obj$data$variables). By default this is uniform (symmetric dirichlet with parameter 1).

Value

A vector of observable probabilities that satisfy the causal model

Examples

```
graph <- initialize_graph(graph_from_literal(Z -+ X, X -+ Y, Ur -+ X, Ur -+ Y))
prob.form <- list(out = c("X", "Y"), cond = "Z")

iv_model <- create_causalmodel(graph, prob.form = prob.form)
sample_distribution(iv_model)
```

`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_2
`nsim`	Number of simulation replicates

Value

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

Examples

```
b <- initialize_graph(graph_from_literal(X -+ Y, Ur -+ X, Ur -+ Y))
obj <- analyze_graph(b, constraints = NULL, effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
bounds <- optimize_effect_2(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