Using R6causal

Juha Karvanen

2021-08-04

Overview

The R package `R6causal` implements an R6 class called `SCM`. The class aims to simplify working with structural causal models. The missing data mechanism can be defined as a part of the structural model.

The class contains methods for

- defining a structural causal model via functions, text or conditional probability tables
- printing basic information on the model
- plotting the graph for the model using packages `igraph` or `qgraph`
- simulating data from the model
- applying an intervention
- checking the identifiability of a query using the R packages `causaleffect` and `dosearch`
- defining the missing data mechanism
- simulating incomplete data from the model according to the specified missing data mechanism
- checking the identifiability in a missing data problem using the R package `dosearch`

In addition, there are functions for

- running experiments
- counterfactual inference using simulation

Setup

```r
library(R6causal)
library(data.table)
library(stats)
```

Defining the model

Structural causal model (SCM) for a backdoor situation can be defined as follows

```r
backdoor <- SCM$new("backdoor",
  uflist = list(
    uz = function(n) {return(runif(n))},
    ux = function(n) {return(runif(n))},
    uy = function(n) {return(runif(n))}
  ),
  vflist = list(
    z = function(uz) {
      return(as.numeric(uz < 0.4))},
    x = function(ux, z) {
      return(as.numeric(ux < 0.2 + 0.5*z))},
    y = function(uy, z, x) {
      return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
  )
)
```

A shortcut notation for this is

```
backdoor_text <- SCM$new("backdoor",
  uflist = list(
    uz = "n : runif(n)",
    ux = "n : runif(n)",
    uy = "n : runif(n)"
  ),
  vflist = list(
    z = "uz : as.numeric(uz < 0.4)",
    x = "ux, z : as.numeric(ux < 0.2 + 0.5*z)",
    y = "uy, z, x : as.numeric(uy < 0.1 + 0.4*z + 0.4*x)"
  )
)
```

Alternatively the functions of SCM can be specified via conditional probability tables

```
backdoor_condprob <- SCM$new("backdoor",
  uflist = list(
    uz = function(n) {return(runif(n))},
    ux = function(n) {return(runif(n))},
    uy = function(n) {return(runif(n))}
  ),
  vflist = list(
    z = function(uz) {
      return( generate_condprob( ycondx = data.table(z = c(0,1),
                                                     prob = c(0.6,0.4)),
                                 x = data.table(uz = uz),
                                 Umerge_expr = "uz"))},
    x = function(ux, z) {
      return( generate_condprob( ycondx = data.table(x = c(0,1,0,1),
                                                     z = c(0,0,1,1),
                                                     prob = c(0.8,0.2,0.3,0.7)),
                                              x = data.table(z = z, ux = ux),
                                              Umerge_expr = "ux"))},
    y = function(uy, z, x) {
      return( generate_condprob( ycondx = data.table(y= rep(c(0,1), 4),
                                                     z = c(0,0,1,1,0,0,1,1),
                                                     x = c(0,0,0,0,1,1,1,1),
                                                     prob = c(0.9,0.1,0.5,0.5,
                                                              0.5,0.5,0.1,0.9)),
                                              x = data.table(z = z, x = x, uy = uy),
                                              Umerge_expr = "uy"))}
  )
)
```

It is possible to mix the styles and define some elements of a function list as functions, some as text and some as conditional probability tables.

Printing the model

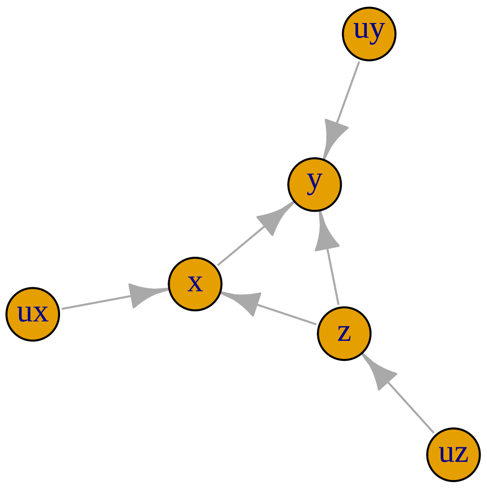
The print method presents the basic information on the model

```
backdoor
#> Name of the model:  backdoor
#>
#> Graph:
#>  z -> x
#>  z -> y
#>  x -> y
#>
#> Functions of background (exogenous) variables:
#>
#> $uz
#> function(n) {return(runif(n))}
#>
#> $ux
#> function(n) {return(runif(n))}
#>
#> $uy
#> function(n) {return(runif(n))}
#>
#> Functions of endogenous variables:
#>
#> $z
#> function(uz) {
#>       return(as.numeric(uz < 0.4))}
#>
#> $x
#> function(ux, z) {
#>       return(as.numeric(ux < 0.2 + 0.5*z))}
#>
#> $y
#> function(uy, z, x) {
#>       return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
#>
#> Topological order of endogenous variables:
#> [1] "z" "x" "y"
#>
#> No missing data mechanism
```

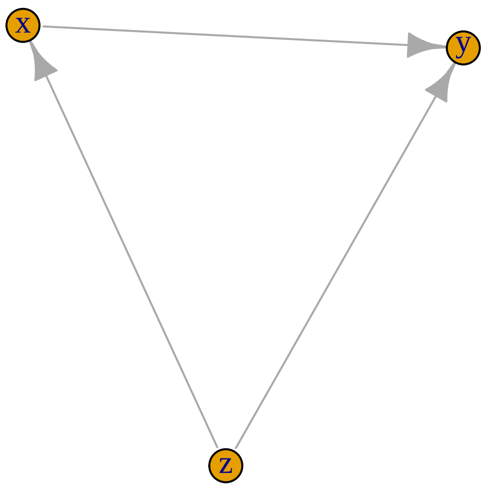
Plotting the graph

The plotting method of the package `igraph` is used by default. If `qgraph` is available, its plotting method can be used as well. The argument `subset` controls which variables are plotted. Plotting parameters are passed to the plotting method.

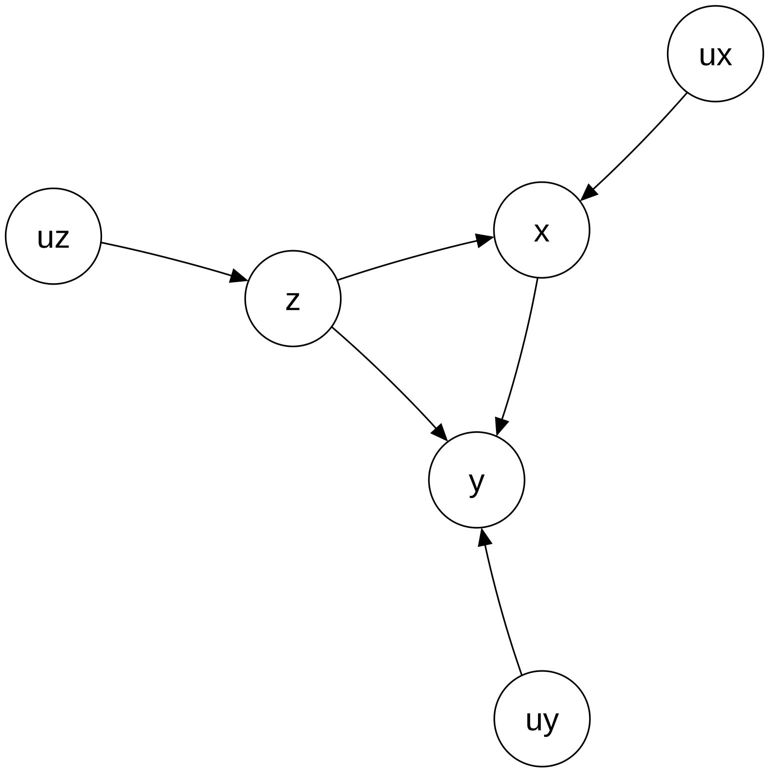
```
backdoor$plot(vertex.size = 25) # with package 'igraph'
```



```
backdoor$plot(subset = "v") # only observed variables
```



```r
if (requireNamespace("qgraph", quietly = TRUE)) backdoor$plot(method = "qgraph")
```



```
# alternative look with package 'qgraph'
```

Simulating data

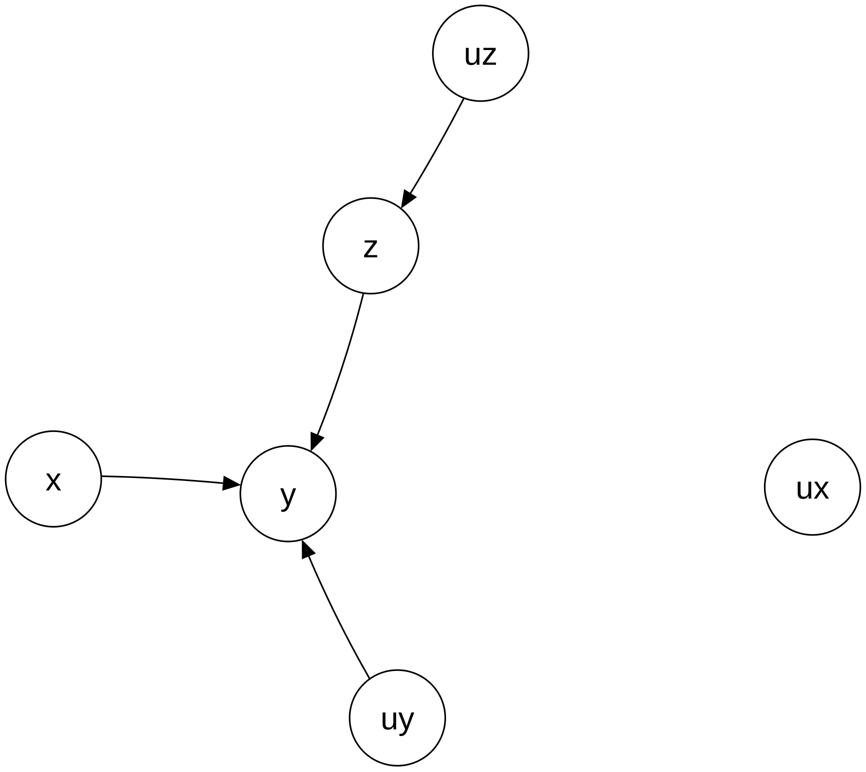
Calling method `simulate()` creates or updates data table `simdata`.

```
backdoor$simulate(10)
backdoor$simdata
#>              uz        ux        uy z x y
#>  1: 0.92761356 0.7176438 0.4150709 0 0 0
#>  2: 0.57147885 0.1760676 0.1452800 0 1 1
#>  3: 0.16298318 0.8849381 0.1799324 1 0 1
#>  4: 0.54141327 0.3424119 0.6899829 0 0 0
#>  5: 0.44568308 0.3633493 0.5155971 0 0 0
#>  6: 0.95376329 0.8965434 0.9721235 0 0 0
#>  7: 0.82920237 0.6932220 0.7148293 0 0 0
#>  8: 0.69452836 0.3106642 0.2805279 0 0 0
#>  9: 0.07441669 0.7214312 0.3507570 1 0 1
#> 10: 0.93447222 0.4552273 0.6246044 0 0 0
backdoor$simulate(8)
backdoor$simdata
#>           uz        ux         uy z x y
#> 1: 0.1417510 0.9124533 0.37549492 1 0 1
#> 2: 0.9253028 0.8371725 0.16993355 0 0 0
#> 3: 0.1899357 0.4874400 0.71402648 1 1 1
#> 4: 0.1450190 0.4010997 0.22326233 1 1 1
#> 5: 0.5945450 0.3248856 0.46045935 0 0 0
#> 6: 0.4502809 0.6612345 0.89066512 0 0 0
#> 7: 0.9843953 0.7876970 0.13702972 0 0 0
#> 8: 0.6895589 0.2498981 0.04366408 0 0 1
backdoor_text$simulate(20)
backdoor_condprob$simulate(30)
```

Applying an intervention

In an intervention, the structural equation of the target variable is changed.

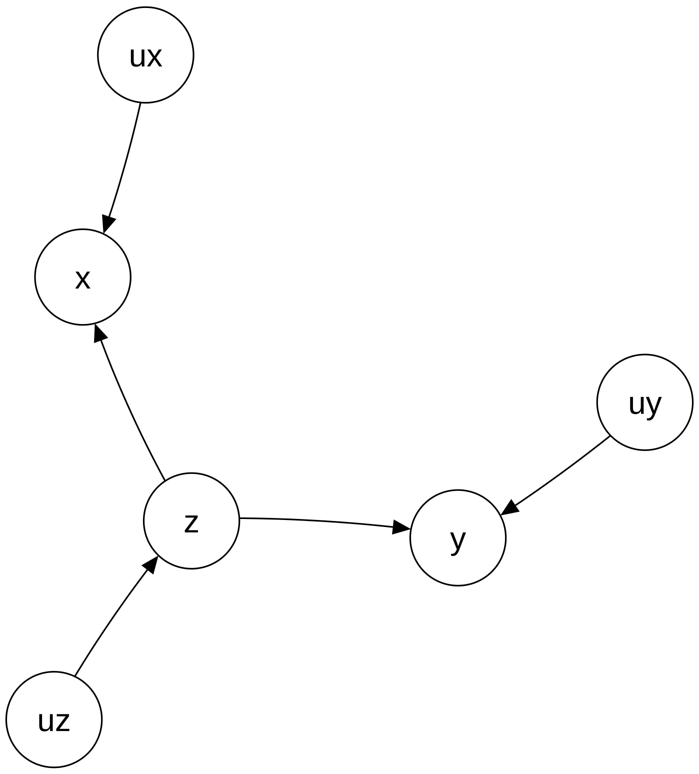
```
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot(method = "qgraph") # to see that arrows incoming to x are cut
```



```
backdoor_x1$simulate(10) # simulating from the intervened model
backdoor_x1$simdata
#>              uz         ux        uy z x y
#>  1: 0.008998798 0.09139074 0.2849736 1 1 1
#>  2: 0.018103428 0.85485724 0.8428110 1 1 1
#>  3: 0.999682913 0.81319096 0.1801496 0 1 1
#>  4: 0.123383418 0.21296194 0.4864987 1 1 1
#>  5: 0.121181384 0.03025285 0.1090193 1 1 1
#>  6: 0.145479019 0.86174771 0.8145651 1 1 1
#>  7: 0.783874152 0.29548418 0.9072963 0 1 0
#>  8: 0.324165836 0.47013422 0.7425031 1 1 1
#>  9: 0.425305847 0.69256935 0.9925019 0 1 0
#> 10: 0.090275605 0.55409570 0.4821261 1 1 1
```

An intervention can redefine a structural equation

```
backdoor_yz <- backdoor$clone()  # making a copy
backdoor_yz$intervene("y",
  function(uy, z) {return(as.numeric(uy < 0.1 + 0.8*z ))}) # making y a function of z only
backdoor_yz$plot(method = "qgraph") # to see that arrow x -> y is cut
```



Running an experiment (set of interventions)

The function `run_experiment` applies a set of interventions, simulates data and collects the results.

```
backdoor_experiment <- run_experiment(backdoor,
                                     intervene = list(x = c(0,1)),
                                     response = "y",
                                     n = 10000)
str(backdoor_experiment)
#> List of 2
#>  $ interventions:Classes 'data.table' and 'data.frame':  2 obs. of  1 variable:
#>   ..$ x: num [1:2] 0 1
#>   ..- attr(*, ".internal.selfref")=<externalptr>
#>   ..- attr(*, "sorted")= chr "x"
#>  $ response_list:List of 1
#>   ..$ y:Classes 'data.table' and 'data.frame':   10000 obs. of  2 variables:
#>   .. ..$ V1: num [1:10000] 1 1 0 0 0 0 0 0 1 0 ...
#>   .. ..$ V2: num [1:10000] 0 0 1 1 1 1 1 0 0 0 ...
#>   .. ..- attr(*, ".internal.selfref")=<externalptr>
colMeans(backdoor_experiment$response_list$y)
#>     V1     V2
#> 0.2649 0.6581
```

Applying the ID algorithm and Do-search

There are direct plugins to R packages `causaleffect` and `dosearch` that can be used to solve identifiability problems.

```
backdoor$causal.effect(y = "y", x = "x")
#> [1] "\\sum_{z}P(y|z,x)P(z)"
backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")
#> \sum_{z}\left(p(z)p(y|x,z)\right)
```

Counterfactual inference

Let us assume that intervention do(X=0) was applied and the response Y = 0 was recorded. What is the probability that in this situation the intervention do(X=1) would have led to the response Y = 1? We estimate this probability by means of simulation.

```
cfdata <- counterfactual(backdoor, situation = list(do = list(target = "x", ifunction = 0),
                                                    condition = data.table( x = 0, y = 0)),
                           target = "x", ifunction = 1, n = 100000)
mean(cfdata$y)
#> [1] 0.53906
```

The result differs from P(Y = 1 | do(X = 1))

```
backdoor_x1$simulate(100000)
mean(backdoor_x1$simdata$y)
#> [1] 0.6621
```

A model with a missing data mechanism

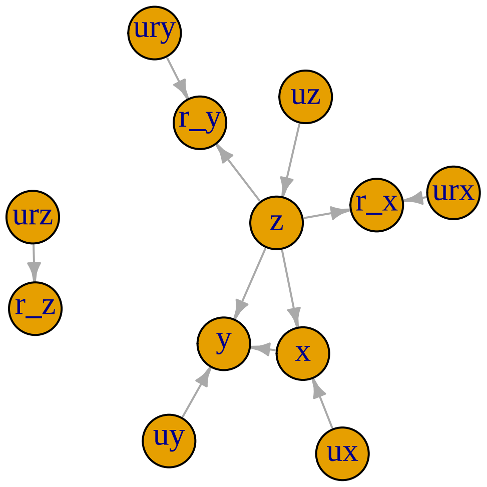
The missing data mechanism is defined in similar manner as the other variables.

```
backdoor_md <- SCM$new("backdoor_md",
                       uflist = list(
                         uz = "n : runif(n)",
                         ux = "n : runif(n)",
                         uy = "n : runif(n)",
                         urz = "n : runif(n)",
                         urx = "n : runif(n)",
                         ury = "n : runif(n)"
                       ),
                       vflist = list(
                         z = "uz : as.numeric(uz < 0.4)",
                         x = "ux, z : as.numeric(ux < 0.2 + 0.5*z)",
                         y = "uy, z, x : as.numeric(uy < 0.1 + 0.4*z + 0.4*x)"
```

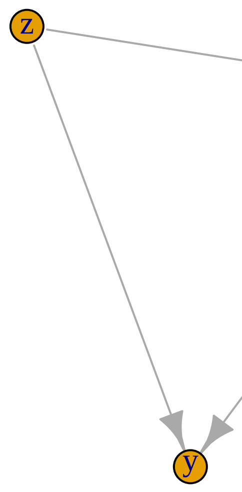
```
                              ),
                              rflist = list(
                                z = "urz : as.numeric( urz < 0.9)",
                                x = "urx, z : as.numeric( (urx + z)/2 < 0.9)",
                                y = "ury, z : as.numeric( (ury + z)/2 < 0.9)"
                              ),
                              rprefix = "r_"
)
```

Plotting the graph for a model with missing data mechanism

```
backdoor_md$plot(vertex.size = 25, edge.arrow.size=0.5) # with package 'igraph'
```



```
backdoor_md$plot(subset = "v") # only observed variables a
```



```
if (!requireNamespace("qgraph", quietly = TRUE)) backdoor_md$plot(method = "qgraph")
# alternative look with package 'qgraph'
```

Simulating incomplete data

By default both complete data and incomplete data are simulated. The incomplete dataset is named as `$simdata_md`.

```
backdoor_md$simulate(100)
summary(backdoor_md$simdata)
#>        uz                 ux                 uy                urz          
#>  Min.   :0.005884   Min.   :0.008417   Min.   :0.004449   Min.   :0.001893  
#>  1st Qu.:0.254081   1st Qu.:0.268156   1st Qu.:0.237812   1st Qu.:0.260992  
#>  Median :0.556790   Median :0.461585   Median :0.457962   Median :0.452310  
#>  Mean   :0.516753   Mean   :0.480535   Mean   :0.473301   Mean   :0.492822  
#>  3rd Qu.:0.771201   3rd Qu.:0.681275   3rd Qu.:0.700072   3rd Qu.:0.768516  
#>  Max.   :0.977778   Max.   :0.984977   Max.   :0.993655   Max.   :0.967244  
#>       urx                ury                  z              x       
#>  Min.   :0.003383   Min.   :0.002698   Min.   :0.00   Min.   :0.00  
#>  1st Qu.:0.221811   1st Qu.:0.250277   1st Qu.:0.00   1st Qu.:0.00  
#>  Median :0.496167   Median :0.492532   Median :0.00   Median :0.00  
#>  Mean   :0.486460   Mean   :0.490796   Mean   :0.37   Mean   :0.41  
#>  3rd Qu.:0.701128   3rd Qu.:0.716006   3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :0.996765   Max.   :0.989260   Max.   :1.00   Max.   :1.00  
#>        y       
#>  Min.   :0.00  
#>  1st Qu.:0.00  
#>  Median :0.00  
#>  Mean   :0.43  
#>  3rd Qu.:1.00  
#>  Max.   :1.00  
summary(backdoor_md$simdata_md)
#>       z_md             x_md             y_md             r_z      
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00  
#>  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.00  
#>  Median :0.0000   Median :0.0000   Median :0.0000   Median :1.00  
#>  Mean   :0.3529   Mean   :0.4149   Mean   :0.4167   Mean   :0.85  
#>  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.00  
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00  
#>  NA's   :15       NA's   :6        NA's   :4                      
#>       r_x            r_y      
#>  Min.   :0.00   Min.   :0.00  
#>  1st Qu.:1.00   1st Qu.:1.00  
#>  Median :1.00   Median :1.00  
#>  Mean   :0.94   Mean   :0.96  
#>  3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :1.00   Max.   :1.00  
#> 
```

By using the argument `fixedvars` one can keep the complete data unchanged and re-simulate the missing data mechanism.

```
backdoor_md$simulate(100, fixedvars = c("x","y","z","ux","uy","uz"))
summary(backdoor_md$simdata)
#>        uz                 ux                  uy                 urz         
#>  Min.   :0.005884   Min.   :0.008417   Min.   :0.004449   Min.   :0.01263  
#>  1st Qu.:0.254081   1st Qu.:0.268156   1st Qu.:0.237812   1st Qu.:0.21430  
#>  Median :0.556790   Median :0.461585   Median :0.457962   Median :0.47089  
#>  Mean   :0.516753   Mean   :0.480535   Mean   :0.473301   Mean   :0.48831  
#>  3rd Qu.:0.771201   3rd Qu.:0.681275   3rd Qu.:0.700072   3rd Qu.:0.76358  
#>  Max.   :0.977778   Max.   :0.984977   Max.   :0.993655   Max.   :0.99925  
#>       urx               ury                 z              x       
#>  Min.   :0.01947   Min.   :0.01725   Min.   :0.00   Min.   :0.00  
#>  1st Qu.:0.21390   1st Qu.:0.31877   1st Qu.:0.00   1st Qu.:0.00  
#>  Median :0.46413   Median :0.55850   Median :0.00   Median :0.00  
#>  Mean   :0.47486   Mean   :0.55431   Mean   :0.37   Mean   :0.41  
#>  3rd Qu.:0.68919   3rd Qu.:0.78426   3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :0.99629   Max.   :0.97410   Max.   :1.00   Max.   :1.00  
#>        y       
#>  Min.   :0.00  
#>  1st Qu.:0.00  
#>  Median :0.00  
#>  Mean   :0.43  
#>  3rd Qu.:1.00  
#>  Max.   :1.00  
summary(backdoor_md$simdata_md)
#>       z_md             x_md             y_md             r_z      
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00  
#>  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.00  
#>  Median :0.0000   Median :0.0000   Median :0.0000   Median :1.00  
#>  Mean   :0.3587   Mean   :0.4167   Mean   :0.3913   Mean   :0.92
#>  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.00
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00
#>  NA's   :8        NA's   :4        NA's   :8
#>       r_x            r_y
#>  Min.   :0.00   Min.   :0.00
#>  1st Qu.:1.00   1st Qu.:1.00
#>  Median :1.00   Median :1.00
#>  Mean   :0.96   Mean   :0.92
#>  3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :1.00   Max.   :1.00
#>
```

Applying Do-search for a missing data problem

```
backdoor_md$dosearch(data = "p(x*,y*,z*,r_x,r_y,r_z)", query = "p(y|do(x))")
#> \sum_{z}\left(\frac{p(z,r_z = 1)}{p(r_z = 1)}p(y|z,r_z = 1,x,r_x = 1,r_y = 1)\right)
```

It is automatically recognized that the problem is a missing data problem when `rflist != NULL`.