Using R6causal

Juha Karvanen

2022-11-03

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

```
library(R6causal)
#library(R6)
#library(igraph)
library(data.table)
library(stats)
#source("../../R/R6causal.R")
```

Defining the model

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

```
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

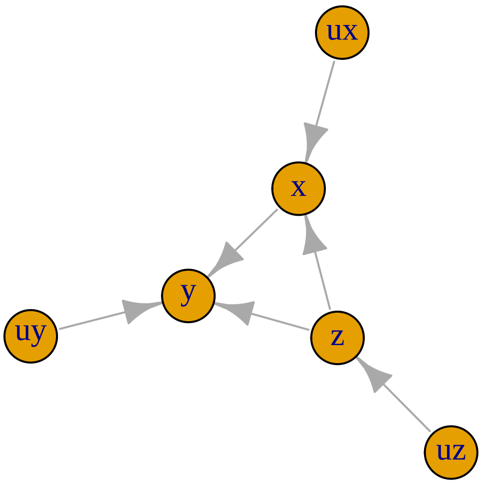
The print method presents the basic information on the model

```
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
```

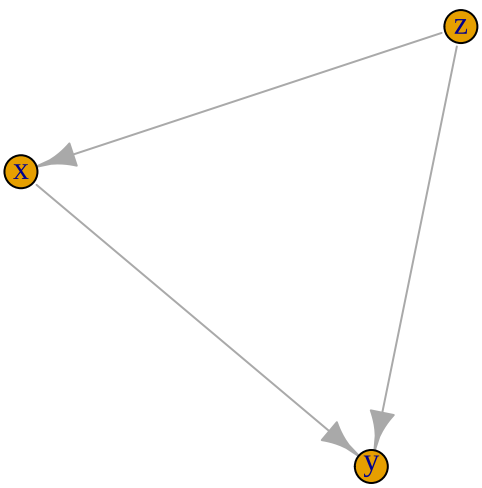
Plotting the graph

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.

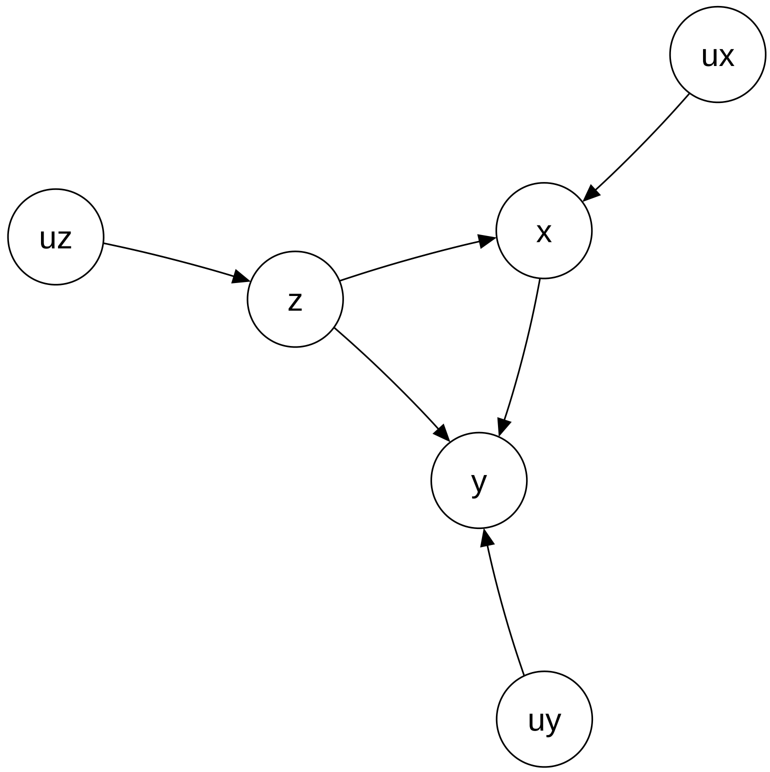
```
backdoor$plot(vertex.size = 25) # with package 'igraph'
```




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




```
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`.

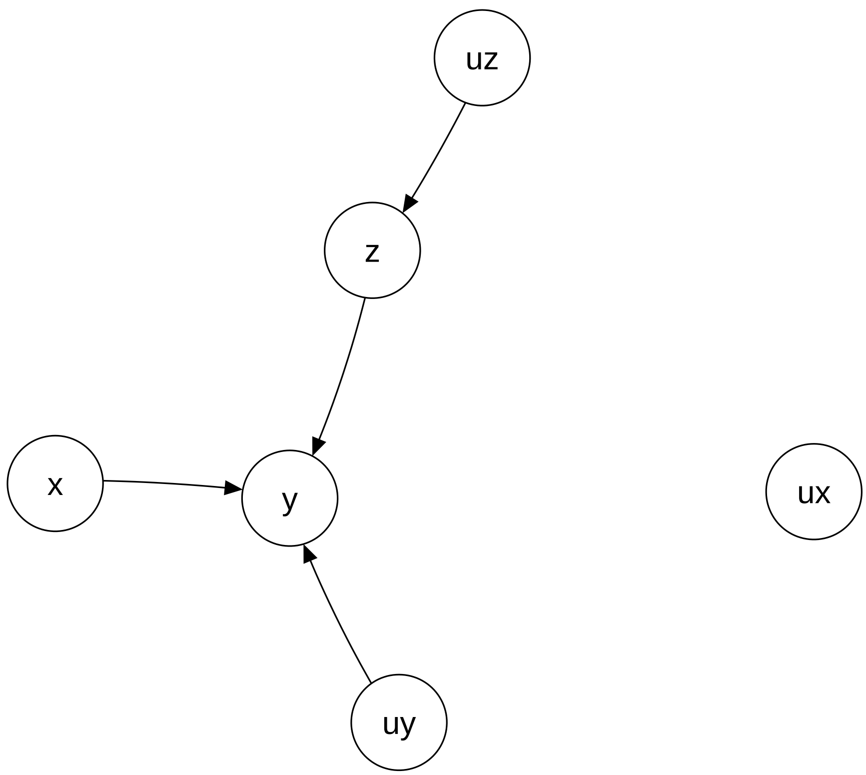
```
backdoor$simulate(10)
backdoor$simdata
#>            uz         ux         uy z x y
#>  1: 0.4091081 0.34619517 0.97233637 0 0 0
#>  2: 0.5390893 0.48763510 0.80148696 0 0 0
#>  3: 0.6125373 0.15654176 0.38739747 0 1 1
#>  4: 0.7774449 0.93048585 0.30228000 0 0 0
#>  5: 0.3501896 0.20500551 0.02204707 1 1 1
#>  6: 0.7942969 0.70065460 0.10267438 0 0 0
#>  7: 0.1883968 0.41287762 0.01575604 1 1 1
#>  8: 0.3635700 0.06318681 0.29469611 1 1 1
#>  9: 0.1685302 0.04877572 0.38833880 1 1 1
#> 10: 0.2077788 0.73245013 0.30549863 1 0 1
backdoor$simulate(8)
backdoor$simdata
#>           uz         ux        uy z x y
#> 1: 0.2955121 0.88016339 0.1973005 1 0 1
#> 2: 0.2821119 0.10627018 0.7361427 1 1 1
#> 3: 0.3012310 0.21539950 0.5498522 1 1 1
#> 4: 0.3649917 0.84037874 0.3807322 1 0 1
#> 5: 0.4277324 0.74141715 0.2367334 0 0 0
#> 6: 0.4124502 0.13979462 0.8838967 0 1 0
#> 7: 0.4362766 0.66678418 0.3588845 0 0 0
```

```
#> 8: 0.1899799 0.03925491 0.9931340 1 1 0
backdoor_text$simulate(20)
backdoor_condprob$simulate(30)
```

Applying an intervention

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

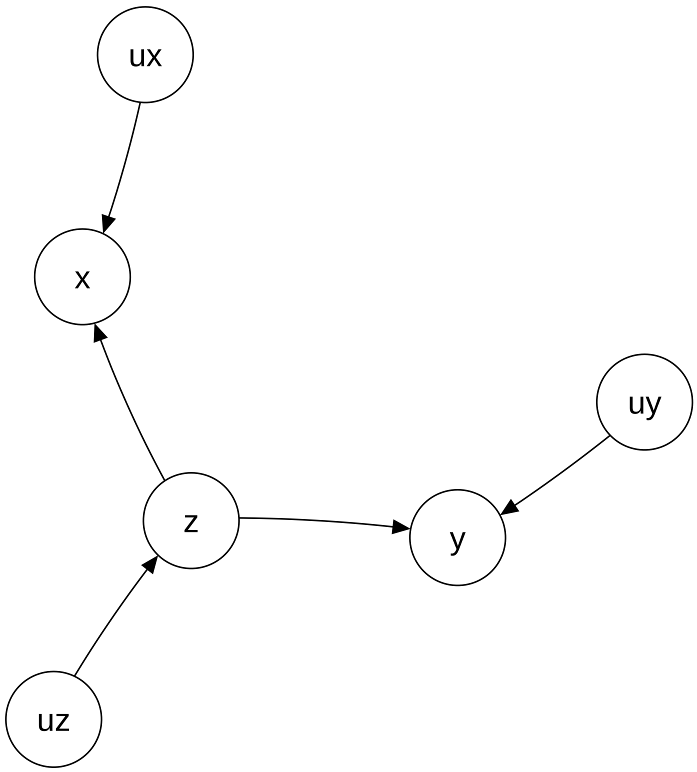
```
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot(method = "qgraph") # to see that arrows incoming to x are cut
```




```
backdoor_x1$simulate(10) # simulating from the intervened model
backdoor_x1$simdata
#>             uz         ux         uy z x y
#>  1: 0.41861761 0.68672435 0.11187979 0 1 1
#>  2: 0.49636772 0.79814902 0.17176071 0 1 1
#>  3: 0.77179848 0.09161171 0.08292911 0 1 1
#>  4: 0.03267088 0.70431273 0.42218246 1 1 1
#>  5: 0.21609534 0.63153036 0.48479283 1 1 1
#>  6: 0.58926917 0.36204053 0.00715665 0 1 1
#>  7: 0.05274911 0.51120479 0.09639364 1 1 1
#>  8: 0.05755443 0.71059840 0.17977485 1 1 1
#>  9: 0.80513274 0.08637929 0.25507561 0 1 1
#> 10: 0.18856871 0.80590205 0.87986503 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] 0 0 1 1 1 0 0 0 0 0 ...
#>   .. ..$ V2: num [1:10000] 1 1 1 1 1 1 1 0 1 1 ...
#>   .. ..- attr(*, ".internal.selfref")=<externalptr>
colMeans(backdoor_experiment$response_list$y)
#>     V1     V2
#> 0.2676 0.6551
```

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 (a simple case)

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] 0.53982
```

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

```
backdoor_x1$simulate(100000)
mean(backdoor_x1$simdata$y)
#> [1] 0.66197
```

Counterfactual inference (parallel worlds)

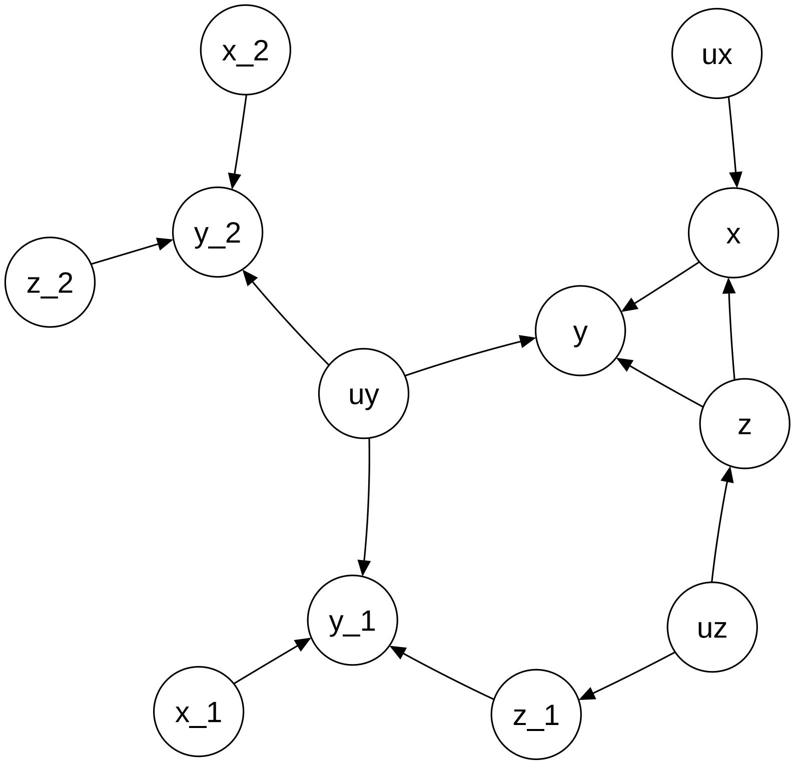
Parallel world graphs (a generalization of a twin graph) are used for counterfactual inference with several counterfactual interventions . The package implements class `ParallelWorld` which heritates class `SCM`. A `ParallelWorld` object is created from an `SCM` object by specifying the interventions for each world. By default the variables of the parallel worlds are named with suffixes "_1", "_2", ...

In the example below, we have the original world (variables `x`, `z`, `y`) and its two variants. In the variant 1 (variables `x_1`, `z_1`, `y_1`), the value of `x` (variable `x_1` in the object) is set to be 0. In the variant 2 (variables `x_2`, `z_2`, `y_2`), the value of `x` (variable `x_2` in the object) is set to be 0 and the value of `z` (variable `z_2` in the object) is set to be 1.

```
backdoor_parallel <- ParallelWorld$new(
                         backdoor,
                        dolist=list(
                             list(target = "x",
                                 ifunction = 0),
                             list(target = list("z","x"),
                                  ifunction = list(1,0))
                         )
 )
backdoor_parallel
#> Name of the model:  backdoor
#>
#> Graph:
#>  uz -> z
#>  z -> x
```

```
#>  uy -> y
#>  z -> y
#>  x -> y
#>  uz -> z_1
#>  uy -> y_1
#>  z_1 -> y_1
#>  x_1 -> y_1
#>  uy -> y_2
#>  z_2 -> y_2
#>  x_2 -> y_2
#>
#> Functions of background (exogenous) variables:
#>
#> $uz
#> function(n) {return(runif(n))}
#> <bytecode: 0x000001dff58df010>
#>
#> $ux
#> function(n) {return(runif(n))}
#> <bytecode: 0x000001dff578b3d8>
#>
#> $uy
#> function(n) {return(runif(n))}
#> <bytecode: 0x000001dff5632770>
#>
#> Functions of endogenous variables:
#>
#> $z
#> function(uz) {
#>       return(as.numeric(uz < 0.4))}
#> <bytecode: 0x000001dff5517890>
#>
#> $x
#> function(ux, z) {
#>       return(as.numeric(ux < 0.2 + 0.5*z))}
#> <bytecode: 0x000001dff5274400>
#>
#> $y
#> function(uy, z, x) {
#>       return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
#> <bytecode: 0x000001dff4cdfd68>
#>
#> $z_1
#> function (uz)
#> {
#>     return(as.numeric(uz < 0.4))
#> }
#>
#> $x_1
#> function (...)
#> {
#>     return(constant)
#> }
```

```
#> <environment: 0x000001dffd94ec20>
#>
#> $y_1
#> function (uy, z_1, x_1)
#> {
#>     return(as.numeric(uy < 0.1 + 0.4 * z_1 + 0.4 * x_1))
#> }
#>
#> $z_2
#> function (...)
#> {
#>     return(constant)
#> }
#> <environment: 0x000001dffbea9738>
#>
#> $x_2
#> function (...)
#> {
#>     return(constant)
#> }
#> <environment: 0x000001dffbeb0438>
#>
#> $y_2
#> function (uy, z_2, x_2)
#> {
#>     return(as.numeric(uy < 0.1 + 0.4 * z_2 + 0.4 * x_2))
#> }
#>
#> Topological order of endogenous variables:
#> [1] "x_1" "z_2" "x_2" "z"   "z_1" "y_2" "x"   "y_1" "y"
#>
#> No missing data mechanism
if (requireNamespace("qgraph", quietly = TRUE)) backdoor_parallel$plot(method = "qgraph")
```



Counterfactual data can be simulated with function `counterfactual`. In the example below, we know that variable `y` obtained value 0 in the original world as well as variants 1 and 2. We are interested in the counterfactual distribution of `y` if `x` had been set to 1.

```
cfdata <- counterfactual(backdoor_parallel,
                         situation = list(
                            do = NULL,
                            condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                         target = "x",
                         ifunction = 1,
                         n = 100000)
mean(cfdata$y)
#> [1] 0.12149
```

The printed value is a simulation based estimate for the counterfactual probability $P(Y = 1)$.

An alternative way for answering the same question defines the case of interest as one of the parallel worlds (here variant 3).

```
backdoor_parallel2 <- ParallelWorld$new(
                         backdoor,
                        dolist=list(
                             list(target = "x",
                                 ifunction = 0),
                             list(target = list("z","x"),
                                  ifunction = list(1,0)),
                              list(target = "x",
                                 ifunction = 1)
                         )
 )
cfdata <- counterfactual(backdoor_parallel2,
```

```
                          situation = list(
                             do = NULL,
                             condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                          n = 100000)
mean(cfdata$y_3)
#> [1] 0.12534
```

The printed value is a simulation based estimate for the counterfactual probability $P(Y = 1)$.

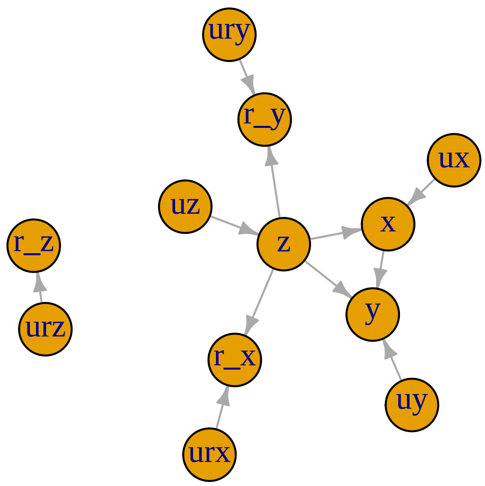
A model with a missing data mechanism

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

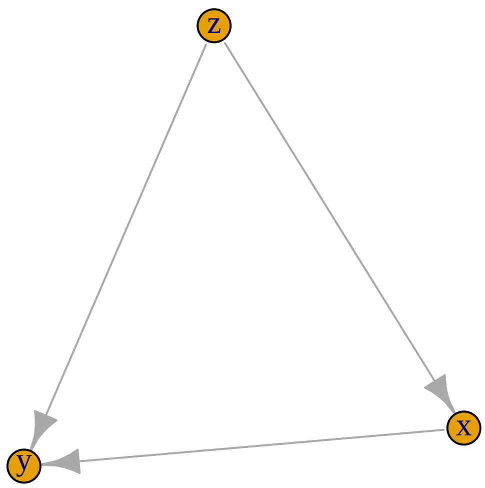
```
backdoor_md <- SCM$new("backdoor_md",
                       uflist = list(
                         uz = "n : runif(n)",
                         ux = "n : runif(n)",
                         uy = "n : runif(n)",
                         urz = "n : runif(n)",
                         urx = "n : 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.   :0.06381   Min.   :0.003863   Min.   :0.008065   Min.   :0.01034
#>  1st Qu.:0.29191   1st Qu.:0.238020   1st Qu.:0.238162   1st Qu.:0.23061
#>  Median :0.55226   Median :0.426365   Median :0.508321   Median :0.48659
#>  Mean   :0.54417   Mean   :0.464233   Mean   :0.506965   Mean   :0.48106
#>  3rd Qu.:0.77181   3rd Qu.:0.686373   3rd Qu.:0.746092   3rd Qu.:0.73163
#>  Max.   :0.98037   Max.   :0.998504   Max.   :0.998480   Max.   :0.97202
#>       urx               ury                  z              x
#>  Min.   :0.01922   Min.   :0.001737   Min.   :0.00   Min.   :0.00
#>  1st Qu.:0.27450   1st Qu.:0.256149   1st Qu.:0.00   1st Qu.:0.00
#>  Median :0.52858   Median :0.495024   Median :0.00   Median :0.00
```

```
#>  Mean   :0.52855   Mean   :0.502900   Mean   :0.36   Mean   :0.39
#>  3rd Qu.:0.78149   3rd Qu.:0.734692   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :0.99570   Max.   :0.997536   Max.   :1.00   Max.   :1.00
#>        y
#>  Min.   :0.00
#>  1st Qu.:0.00
#>  Median :0.00
#>  Mean   :0.39
#>  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   :0.3656   Mean   :0.3684   Mean   :0.3684   Mean   :0.93
#>  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   :7        NA's   :5        NA's   :5
#>       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   :0.95   Mean   :0.95
#>  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.   :0.06381   Min.   :0.003863   Min.   :0.008065   Min.   :0.002976
#>  1st Qu.:0.29191   1st Qu.:0.238020   1st Qu.:0.238162   1st Qu.:0.234996
#>  Median :0.55226   Median :0.426365   Median :0.508321   Median :0.553774
#>  Mean   :0.54417   Mean   :0.464233   Mean   :0.506965   Mean   :0.505304
#>  3rd Qu.:0.77181   3rd Qu.:0.686373   3rd Qu.:0.746092   3rd Qu.:0.772887
#>  Max.   :0.98037   Max.   :0.998504   Max.   :0.998480   Max.   :0.996145
#>       urx               ury                 z               x
#>  Min.   :0.02581   Min.   :0.002119   Min.   :0.00   Min.   :0.00
#>  1st Qu.:0.25041   1st Qu.:0.297996   1st Qu.:0.00   1st Qu.:0.00
#>  Median :0.49194   Median :0.481074   Median :0.00   Median :0.00
#>  Mean   :0.50464   Mean   :0.486273   Mean   :0.36   Mean   :0.39
#>  3rd Qu.:0.73475   3rd Qu.:0.701777   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :0.99805   Max.   :0.987774   Max.   :1.00   Max.   :1.00
#>        y
#>  Min.   :0.00
#>  1st Qu.:0.00
#>  Median :0.00
#>  Mean   :0.39
#>  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   :0.3656   Mean   :0.3587   Mean   :0.3407   Mean   :0.93  
#>  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   :7        NA's   :8        NA's   :9                      
#>       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   :0.92   Mean   :0.91  
#>  3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :1.00   Max.   :1.00  
#> 
```

Applying Do-search to 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`.