Graphical Simulator Generator

Bob and Associates

2023-04-30

Here we attempt to implement Judea Pearl’s “[Multistage Simpson’s Paradox machine](http://ftp.cs.ucla.edu/pub/stat_ser/r414-reprint.pdf)” (Figure 3).

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.2

library(bnlearn)

## Warning: package 'bnlearn' was built under R version 4.2.2

library(Rgraphviz)

## Loading required package: graph

## Loading required package: BiocGenerics

##   
## Attaching package: 'BiocGenerics'

## The following object is masked from 'package:bnlearn':  
##   
## score

## The following objects are masked from 'package:dplyr':  
##   
## combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, append, as.data.frame, basename, cbind, colnames,  
## dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,  
## grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,  
## order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,  
## rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,  
## union, unique, unsplit, which.max, which.min

##   
## Attaching package: 'graph'

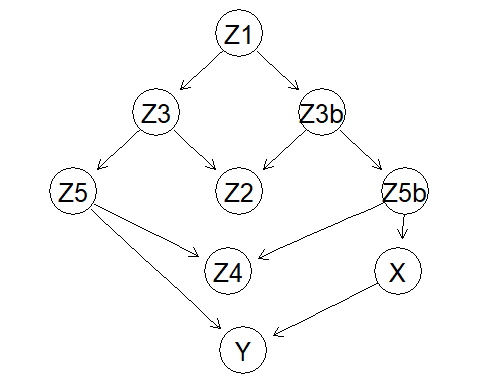
## The following objects are masked from 'package:bnlearn':  
##   
## degree, nodes, nodes<-

## Loading required package: grid

(The plot of the graph is the only thing that depends on Rgraphviz, which needs to be installed from Bioconductor, or bnlearn which also has some complicated dependencies as I recall. You don’t need these packages to run the simulation.)

Define the causal graph and display it:

dependency\_str <- "[Z1]  
 [Z3 | Z1 ]  
 [Z3b| Z1 ]  
 [Z2 | Z3 : Z3b]  
 [Z5 | Z3 ]  
 [Z5b| Z3b ]  
 [Z4 | Z5 : Z5b]  
 [X | Z5b ]  
 [Y | X : Z5 ]" %>% gsub(' ', '', .)  
  
node\_descriptions <- dependency\_str %>% strsplit('\n') %>% '[['(1)  
  
tree = model2network(node\_descriptions %>% paste0(collapse=''))  
graphviz.plot(tree)



Parse the graph to list the parents of each node.

parse\_node\_str <- function(node\_str){  
 # node <- node\_str %>% gsub("\\[([^:]+).\*\\]", "\\1", .)  
 parts <- node\_str %>%  
 gsub('[', '', ., fixed=TRUE) %>%   
 gsub(']', '', ., fixed=TRUE) %>%   
 strsplit('\\| ?') %>% '[['(1) %>%   
 strsplit('\\: ?')  
   
 node <- list(id = parts[[1]])  
   
 if (length(parts) > 1){  
 node['parents'] <- parts[2]  
 }  
   
 node  
}  
  
parsed\_nodes <- node\_descriptions %>% lapply(parse\_node\_str)  
  
node\_list <- parsed\_nodes %>% lapply(function(node) node$parents)  
names(node\_list) <- parsed\_nodes %>% lapply(function(node) node$id)

Simulate data according to the dependency relationships in the graph. First find the nodes with no parents and give them random values. Then find the nodes whose parents have already been simulated, and give them values using a function that depends on the parents (here we use simple linear functions, but other functions should be possible). Repeat until all nodes have been added as columns in the dataframe of simulated data.

gen\_fun <- function(data\_cols, noise\_sd=0.01){  
 M <- data\_cols %>% as.data.frame %>% as.matrix  
 num\_data\_cols <- length(data\_cols)  
 beta <- runif(num\_data\_cols)  
 noise <- rnorm(nrow(M), sd=noise\_sd)  
 M %\*% beta + noise  
}  
  
N <- 100  
  
columns <- list()  
  
remaining\_nodes <- node\_list  
  
while (length(remaining\_nodes) > 0){  
 for (node\_name in names(remaining\_nodes)){  
 node <- remaining\_nodes[[node\_name]]  
 if (length(node) == 0){  
 # no dependencies  
 print(sprintf("Adding root node %s", node\_name))  
 columns[[node\_name]] = runif(N)  
 remaining\_nodes[[node\_name]] <- NULL  
 } else if ( length(setdiff(names(node), names(remaining\_nodes))) == 0 ) {  
 # all dependencies are already in the data  
 print(sprintf("Adding dependent node %s", node\_name))  
 columns[[node\_name]] = gen\_fun( columns[unlist(node)] )  
 remaining\_nodes[node\_name] <- NULL  
 } else {  
 print(sprintf("Starting over for node %s", node\_name))  
 }  
 # keep going  
 }  
}

## [1] "Adding root node Z1"  
## [1] "Adding dependent node Z3"  
## [1] "Adding dependent node Z3b"  
## [1] "Adding dependent node Z2"  
## [1] "Adding dependent node Z5"  
## [1] "Adding dependent node Z5b"  
## [1] "Adding dependent node Z4"  
## [1] "Adding dependent node X"  
## [1] "Adding dependent node Y"

df <- as.data.frame(columns)

This is what the resulting dataframe looks like:

df %>% head

## Z1 Z3 Z3b Z2 Z5 Z5b Z4  
## 1 0.6288137 0.14361662 0.4658809 0.3426260 0.04725460 0.23604332 0.07115357  
## 2 0.6987388 0.15491006 0.5254247 0.3700805 0.05028079 0.25480540 0.09359162  
## 3 0.2996405 0.07606951 0.1974289 0.1706045 0.01632231 0.08820002 0.03945484  
## 4 0.4433550 0.08263462 0.3169188 0.2133872 0.02068772 0.15235379 0.03396327  
## 5 0.5291328 0.10757537 0.3805036 0.2791004 0.02525266 0.20832392 0.06888168  
## 6 0.6804690 0.16697903 0.4881396 0.3731873 0.03469232 0.25492302 0.07899340  
## X Y  
## 1 0.10517014 0.038656370  
## 2 0.11331280 0.050226613  
## 3 0.03108469 0.015978467  
## 4 0.06958815 0.023702614  
## 5 0.06996019 0.006715332  
## 6 0.11717086 0.041491854

Now we can test whether the reversals we are looking for in the coefficient of ‘X’ actually happen. If the confounding is correctly implemented, we expect to see the sign of the coefficient of ‘X’ flip back and forth between negative and positive as we add the covariates to the analysis in order.

coef( lm(Y ~ X, df) )['X']

## X   
## 0.3376978

coef( lm(Y ~ X + Z1, df) )['X']

## X   
## 0.1249766

coef( lm(Y ~ X + Z1 + Z2, df) )['X']

## X   
## 0.1248999

coef( lm(Y ~ X + Z1 + Z2 + Z3, df) )['X']

## X   
## 0.139609

coef( lm(Y ~ X + Z1 + Z2 + Z3 + Z4, df) )['X']

## X   
## 0.1439648

coef( lm(Y ~ X + Z1 + Z2 + Z3 + Z4 + Z5, df) )['X']

## X   
## 0.1758282

So far we have good news and bad news: We can autogenerate data from a given causal graph by walking down the causal DAG and applying a node-wise data generation function (or functions). Now if we can only get the simulated data to actually capture the demonstrable Simpson’s paradox relationships we are looking for we’d be golden!