Estimation of probability associated with collective counterfactual outcomes

Youjin Lee

Approximation of Directed Acyclic Graph (DAG)

In social network, treatments may spill over from the treated individual to his or her social contacts and outcomes may be contagious continuously. Therefore, causal inference using observational data from a single social network requires observing longitudinal data on treatments and outcomes as they evolve in real-time, so that each spillover or contagious event appears in the data. In these settings, directed acyclic graph (DAG) has been used to represent interference and contagion, but DAGs can quickly be cumbersome. Especially in most cases it is impossible to collect the kind of real-time data required. If it is, the resulting model will be generally high-dimensional and often too big to fit the available data.

To illustrate, using `simcausal R` package, we can define `DAG` object where outcome variables of (Y_1^t, Y_2^t, Y_3^t) change over times $t = 0, 1, 2, \ldots, 20$ while unit-specific treatment variable (A_1, A_2, A_3) has a causal effect on each of them. Figure 1 illustrates data generating mechanism or causal relationship which a `DAG` object of `lDAG` stands for.

```
devtools::install_github('osofr/simcausal', build_vignettes = FALSE)
#> Skipping install of 'simcausal' from a github remote, the SHA1 (43d70ccc) has not changed since last
#>   Use `force = TRUE` to force installation
library(simcausal)
library(igraph)
plotCG <- function(DAG, vertex_attrs = list(), edge_attrs = list()){
  if (!requireNamespace("igraph", quietly = TRUE)) {
    stop("igraph R package is required.", call. = FALSE)
  }

  ### variable without no parents
  parents <- c()
  for(i in 1:length(names(DAG))){
    if (length(attr(DAG, "parent")[[i]]) == 0) parents <- c(parents, names(DAG)[[i]])
  }

  max.time <- max(as.integer(unlist(lapply( strsplit(names(DAG), "_"), '[[', 2))))

  varnames <- unlist(lapply( strsplit(names(DAG), "_"), '[[', 1))
  time <- unlist(lapply( strsplit(names(DAG), "_"), '[[', 2))
  chainset <- unique(varnames[as.integer(time) >= 2])
  if (length(chainset) == 0) return("No chain component detected.")
  parentset = friendset = list()

  for (i in 1:length(chainset)) {

    t <- 1
    tmp <- eval(parse(text = as.character(paste0("attr(DAG, \"parent\")", "$", chainset[i], "_" ,as.cha
    parentset[[i]] <- parents[parents %in% tmp]
    friendset[[i]] <-  unlist(lapply(strsplit(tmp, "_"), '[[', 1 ))[which(as.integer(unlist(lapply(strsp

    for (t in 2:max.time) {
```

```
    tmp <- eval(parse(text = as.character(paste0("attr(DAG, \"parent\")", "$", chainset[i], "_" ,as.c

    newparentset <- parents[parents %in% tmp]
    parentset[[i]] <- newparentset[newparentset %in% parentset[[i]]]

    newfriendset <- unlist(lapply(strsplit(tmp, "_"), '[[', 1 ))[which(as.integer(unlist(lapply(strsp
    friendset[[i]] <- newfriendset[newfriendset %in% friendset[[i]]]
  }
}
parentnode <- unique(unlist(parentset))
friendnode <- unique(unlist(friendset))

edgelist <- c()
for (i in 1:length(chainset)) {
  for (r in 1:length(parentset[[i]])) {
    edgelist <- rbind(edgelist, c(chainset[i], parentset[[i]][[r]]))
  }
  for (r in 1:length(friendset[[i]])) {
    edgelist <- rbind(edgelist, c(chainset[i], friendset[[i]][[r]]))
  }
}
## check validity of chain graph
deleterow <- c()
for (i in 1:length(chainset)) {
  if (sum(rowSums(edgelist == chainset[i]) == 2) == 0) return("Invalid chain graph approximation.")
  deleterow <- c(deleterow, which(rowSums(edgelist == chainset[i]) == 2))
}
if (!is.null(deleterow)) edgelist = edgelist[-deleterow,]

deleterow <- c()
for (i in 1:nrow(edgelist)) {
  if (sum(edgelist[i,] %in% chainset) == 2) {
    if (sum(rowSums(matrix(edgelist %in% edgelist[i,], ncol = 2)) == 2 ) < 2) deleterow <- c(deleterou
  }
}
if (!is.null(deleterow)) edgelist <- edgelist[-deleterow,]

## draw graph
attnames_ver <- names(vertex_attrs)
attnames_edge <- names(edge_attrs)
vertex_attrs_default <- list(color = NA, label.color = "seagreen", shape = "none", size = 15, label.c
edge_attrs_default <- list(color = "black", width = 1, lty = 1, arrow.width = 1, arrow.size = 1)
vertex_attrs <- append(vertex_attrs, vertex_attrs_default[!(names(vertex_attrs_default)%in%attnames_v
edge_attrs <- append(edge_attrs, edge_attrs_default[!(names(edge_attrs_default)%in%attnames_edge)])
####

g <- igraph::graph.empty()
g <- igraph::add.vertices(g, nv = length(parentnode) + length(friendnode), attr = vertex_attrs)
igraph::V(g)$name <- c(parentnode, friendnode)
g <- igraph::add.edges(g, as.vector(t(edgelist)), attr = edge_attrs)


#g <- igraph::set.graph.attribute(g, 'layout')
```

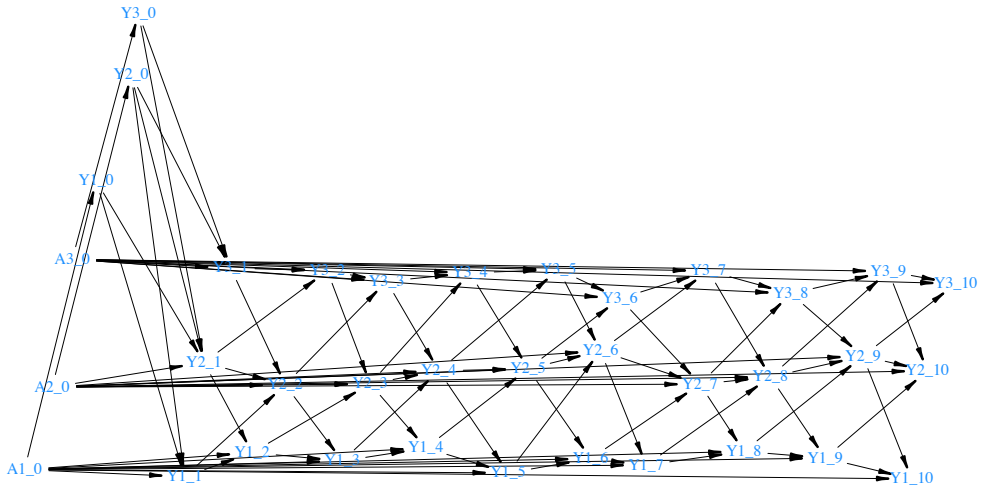

Figure 1: plotting an example of evolving outcome variables (Y_1, Y_2, Y_3) and treatment variables (A_1, A_2, A_3) following directed cyclic graph of `lDAG`.

```
  edgemode <- ifelse(rowSums(matrix(as_edgelist(g) %in% parentnode, ncol = 2) > 0), 1, 0)
  igraph::plot.igraph(g, edge.arrow.mode = edgemode)

}
D <- DAG.empty()
D <- D +
  node("A1", t = 0, distr = "rbern", prob = 0.3) +
  node("A2", t = 0, distr = "rbern", prob = 0.5) +
  node("A3", t = 0, distr = "rbern", prob = 0.7) +
  node("Y1", t = 0, distr = "rbern", prob = plogis(-1 + A1[0])) +
  node("Y2", t = 0, distr = "rbern", prob = plogis(0 - 0.5*A2[0])) +
  node("Y3", t = 0, distr = "rbern", prob = plogis(1 - 0.5*A3[0]))
t.end <- 20
D <- D +
  node("Y1", t = 1:t.end, distr = "rbern",
       prob = plogis(3*Y1[t-1] - 1.5 + A1[0] + Y2[t-1])) +
  node("Y2", t = 1:t.end, distr = "rbern",
       prob = plogis(3*Y2[t-1] - 2 - 0.5*A2[0] + Y1[t-1] -Y3[t-1])) +
  node("Y3", t = 1:t.end, distr = "rbern",
       prob = plogis(3*Y3[t-1] - 1 - 0.5*A3[0] -Y2[t-1]))

lDAG <- set.DAG(D, verbose = FALSE)
```

```
# draw Figure 1
par(mar = c(4, 4, 1, 1))
plotDAG(lDAG, tmax = 10, xjitter = 0.3, yjitter = 0.1,
     edge_attrs = list(width = 0.2, arrow.width = 0.5, arrow.size = 0.3),
     vertex_attrs = list(size = 10, label.cex = 0.5, label.color = "dodgerblue"),
     verbose = FALSE)
```

Note that for causal inference, DAGs are only useful when if we observe all of the Y_i^t variables. However we often observe outcomes at certain time point after some unknown process of interference or contagion: in the above case, $(Y_1^{t=20}, Y_2^{t=20}, Y_3^{t=20})$.

If Y_i^t is approximated equal to Y_i^{t-1}, the conditional independence implied by the chain graph, which includes

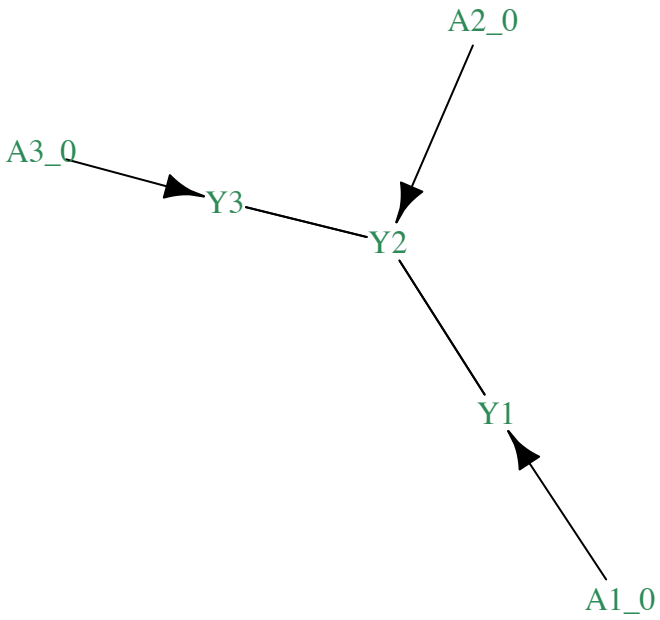

Figure 2: Plotting chain graph component from `lDAG`.

directed edges and undirected edges with no directed cycles, holds approximately in the DAG model. In this case, conditional (in)dependence structure in DAG from Figure 1 (approximately) holds for those of chain graph in Figure 2.

```
# draw chain graph components from lDAG (Figure 2)
library(netchain)
par(mar = c(4, 4, 1, 1))
plotCG(lDAG)
```

Input observations

Chain graphs can be used to define hybrid graphical models combining features of both log-linear models on undirected graphs and DAG models used to represent causal relationships. Chain graphs provide justification of parsimonious conditional log-linear models to identify causal effects on the collective outcomes.

To estimate the parameters in conditional log-linear model, we require enough independent replicates of observations. Observations must take a form of a $m \times n$ matrix for binary Y and A respectively, where m is the number of units and n is the number of independent observations. Depending on the contexts of your scenario, we can enter no confounders (Defaults to `listC = NULL`), a $m \times n$ matrix for `listC`, or a list of $m \times n$ matrices for `listC` for which of each matrix stands for observations for one confounding factors (we later call them `C1`, `C2`, ..., `Cq`).

We provide a simple function `simGibbs` to generate binary $(\mathbf{Y}, \mathbf{A}, \mathbf{C})$ from chain graph model under simple scenario, which requires additional structural information (`weight.matrix`, `treat.matrix`, and `cov.matrix`). Consider the following conditional log-linear model following chain graph model defined up-to two-way interaction effects:

$$p\big(\mathbf{Y} = (y_1, y_2, \ldots, y_m) \mid \mathbf{A} = \mathbf{a}, \mathbf{C} = \mathbf{c}\big) = \frac{1}{Z} \exp\left\{ \sum_{i,j=1}^{m} w_{ij} y_i y_j + \sum_{i,j=1}^{m} k_{ij} a_i y_j + \sum_{i,j=1}^{m} h_{ij} c_i y_j \right\}$$

A weight.matrix$_{ij}$ (w_{ij}) indicates two-way interaction effects between Y_i and Y_j when $i \neq j$ and main effect

of each outcomes when $i = j$. Note that by the construction of chain graph model, `weight.matrix` should be symmetric. A treat.matrix$_{ij}$ (k_{ij}) indicates a direct causal effect of A_i on Y_j; a cov.matrix$_{ij}$ (h_{ij}) indicates a direct causal effect of C_i on Y_j. By the definition, `treat.matrix` and `cov.matrix` are directional.

The following codes generate $n = 100$ (`n.gibbs`×`n.sample` $= 10 \times 10$) observations of $m = 3$ (`n.unit` $= 3$) units assuming chain graph of $Y_1 - -Y_2 - -Y_3$ and direct effect from $A_i \to Y_i$ and $C_i \to Y_i$ for $i = 1, 2, 3$.

```
library(netchain)
weight.matrix = matrix(c(0.5, 1, 0, 1, 0.3, 0.5, 0, 0.5, -0.5), 3, 3)
simobs = simGibbs(n.unit = 3, n.gibbs = 10, n.sample = 10,
                  weight.matrix,
                  treat.matrix = 0.5*diag(3), cov.matrix= (-0.3)*diag(3) )
inputY = simobs$inputY
inputA = simobs$inputA
inputC = simobs$inputC
head(inputY)
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#> [2,]    1    1    0
#> [3,]    1    1    0
#> [4,]    1    1    1
#> [5,]    1    1    1
#> [6,]    1    1    1
head(inputA)
#>      [,1] [,2] [,3]
#> [1,]    0    1    0
#> [2,]    0    1    0
#> [3,]    0    1    0
#> [4,]    0    1    0
#> [5,]    0    1    0
#> [6,]    0    1    0
head(inputC)
#>      [,1] [,2] [,3]
#> [1,]    0    1    1
#> [2,]    0    1    1
#> [3,]    0    1    1
#> [4,]    0    1    1
#> [5,]    0    1    1
#> [6,]    0    1    1
```

Input edge information

With (outcomes, treatment, confounders) = (`inputY`, `inputA`, `listC`), we ultimately aim to calculate probability associated with counterfactual collective outcomes given a certain unit-specific treatment of length m, $\mathbf{a}$ (`treatment`):

$$P\big(\mathbf{Y}(\mathbf{a}) = \mathbf{y}\big).$$

Depending on the form of target outcomes of $\mathbf{y}$, `targetoutcome` may vary: in the above example of $m = 3$ units, `targetoucome = (1,0,0)` indicates we want to calculate $P\big(\mathbf{Y}(\mathbf{a}) = (1, 0, 0)\big)$. If we want to calculate the probability of multiple outcomes, e.g. homogeneous (unanimous) outcomes of (0,0,0) or (1,1,1), you can input a matrix `targetoutcome = rbind(c(0,0,0), (1,1,1))`. If you want to calculate the probability when only one unit has one, instead of listing all mutually exclustive, possible outcomes `targetoutcome = rbind(c(1,0,0), c(0,1,0), c(0,0,1))`, you can specify the number of one (or maximum value of `inputY`):

`targetoutcome = 1`. The default is `targetoutcome = "mean"`, which calculate $E[\bar{Y}] := E[(Y_1 + Y_2 + \ldots + Y_n)/n]$.

Most tricky but important part is edge information (`R.matrix`, `E.matrix`, and `edgeinfo`), which determines the factors put in the conditional log-linear model.

A `R.matrix` is $m \times m$ relational symmetric matrix where $\text{R.matrix}_{ij} = 1$ indicates Y_i and Y_j are adjacent, meaning the existence of interactions of feedback between the two. A `E.matrix` a $m \times m$ matrix where E.matrix_{ij} indicates A_i has a direct causal effect on Y_j; defaults to diagonal matrix, which indicates no interference. On the other hand, `edgeinfo` a list of matrix specifying additional directed edges (from confounders (specified in `listC`) or treatment (`inputA`) to the outcomes (`inputY`)) information. Assume that `edgeinfo = list(mat1, mat2)` where both `mat1` and `mat2` are matrices. For each matrix the first column specifies the types of variables that must involve outcome variable, `"Y"`; while `"A"` indicates treatment variable and `"C"` indicates confounder. If you input multiple confounders in `listC` as a list, `"C1"` indicates the first confounders, `"C2"` indicates the second, and so on. For each matrix the second column specifies an index for unit corresponding to the varaible in the first column. For example,

```
mat1 = rbind(c("Y", 1), c("A", 1), c("A", 2))
```

implies that A_1 and A_2 have a direct causal effect on Y_1 and these two have a interaction effect; this will lead to three-way interaction term of Y_1, A_1, and A_2 in the conditional log-linear model. (In this case a direct effect of A_1 on Y_1 and A_2 on Y_1 should be also (manually) specified.)

```
mat2 = rbind(c("Y", 1), c("C1", 1), c("C2", 1))
```

When you have multiple confounders and these confounders interactively have an effect on the outcomes, you can put edge information as `mat2`, which implies a direct effect of C_{11} and C_{21} on Y_1 but these effects are dependent each other. (In this case, you should include `rbind(c("Y", 1), c("C1",1))` and `rbind(c("Y", 1), c("C2", 1) )` in `edgeinfo` manually.)

Counterfactual outcomes

With those observations and edge information, the next step is to infer parameters in the conditional log-linear models, and with those parameters we generate counterfactual outcomes using Gibbs sampling. Gibbs sampling requires given `treatment` assignment, the number of independent Gibbs samplings (`n.gibbs`), the number of samples per each iteration (`n.sample`), and the number of burn-in samples (`n.burn`) per each iteration.

```
set.seed(1234)
library(netchain)
weight.matrix = matrix(c(0.5, 1, 0, 1, 0.3, 0.5, 0, 0.5, -0.5), 3, 3)
simobs = simGibbs(n.unit = 3, n.gibbs = 10, n.sample = 10,
                  weight.matrix,
                  treat.matrix = 0.5*diag(3), cov.matrix= (-0.3)*diag(3) )
inputY = simobs$inputY
inputA = simobs$inputA
inputC = simobs$inputC

R.matrix = ifelse(weight.matrix==0, 0, 1)
diag(R.matrix) = 0

result = chain.causal.multi(targetoutcome = "mean", treatment = c(1,0,0),
inputY, inputA, listC = inputC, R.matrix = R.matrix, E.matrix = diag(3),
edgeinfo = list(rbind(c("Y", 1), c("C", 1)), rbind(c("Y", 2), c("C", 2)),
rbind(c("Y", 3), c("C", 3))), n.obs = 1000, n.burn = 100)
```

```
print(result)
#> $causalprob
#> [1] 0.5854467
#>
#> $n.par
#> [1] 11
#>
#> $par.est
#>  [1]  2.5175370 -0.2081631 -0.4115525  1.4919613  0.5595840  0.1569165
#>  [7]  1.4209752  1.4663121 -3.1785419 -1.1046469 -0.8115134
```

From the results above, when $\mathbf{a} = (1, 0, 0)$, $E[\bar{Y}(\mathbf{a})]$ is around 0.59.

Approximate DAG process

Using `sim` function provided in `R` package `simcausal`, we can generate simulated data following specified DAG (e.g. `lDAG` from Figure 1). From the above example where we define `DAG` object of `lDAG`, we generate `sample.size` $= 1000$ observations to examine the performance of `chain.causal.multi` function to infer counterfactual outcomes ($\{Y_1(\mathbf{a}), Y_2(\mathbf{a}), Y_3(\mathbf{a})\}$) under specific treatment assignment of $\mathbf{a} = (1, 0, 1)$. Using `set.targetE` and `eval.target` functions from `simcausal`, we can derive (approximate) counterfactual outcomes under $\mathbf{a} = (1, 0, 1)$ by generating 10000 counterfactual outcomes.

```
sample.size <- 1000
simcausaldat <- sim(DAG = lDAG, n = sample.size, rndseed = 1234)
#> simulating observed dataset from the DAG object

treatments <- c(1, 0, 1) # treatment assignment
act_theta <- c(node("A1", t = 0, distr = "rbern", prob = alpha),
               node("A2", t = 0, distr = "rbern", prob = beta),
               node("A3", t = 0, distr = "rbern", prob = gamma))
lDAG <- lDAG + action("all_theta101", nodes = act_theta,
                      alpha = treatments[1], beta = treatments[2], gamma = treatments[3])
Y1_treat101 <- set.targetE(DAG = lDAG, outcome = "Y1", t = t.end,
                         param = "all_theta101")
Y2_treat101 <- set.targetE(DAG = lDAG, outcome = "Y2", t = t.end,
                           param = "all_theta101")
Y3_treat101 <- set.targetE(DAG = lDAG, outcome = "Y3", t = t.end,
                           param = "all_theta101")
results1 <- eval.target(Y1_treat101, n = 10000, verbose = FALSE)$res
results2 <- eval.target(Y2_treat101, n = 10000, verbose = FALSE)$res
results3 <- eval.target(Y3_treat101, n = 10000, verbose = FALSE)$res
print(c(results1, results2, results3))
#> Mean_Y1_20 Mean_Y2_20 Mean_Y3_20
#>     0.8988     0.5334     0.3254
```

Those numbers above illustrate the (approximate) true counterfactual outcomes $\{Y_1(\mathbf{a}), Y_2(\mathbf{a}), Y_3(\mathbf{a})\}$ under $\mathbf{a} = (1, 0, 1)$. From the below, we illustrate how `chain.causal.multi` esimates individual counterfactual outcomes as well as collective counterfactual outcomes when outcomes at the final stage ($\{Y_1^{t=20}, Y_2^{t=20}, Y_3^{t=20}\}$) are only available among time-evolving outcomes. Inference on single counterfactual outcome can be easily done by specifying `targetoutcome` as a matrix of which rows contain all possible combinations where $Y_1 = 1$.

```
library(gtools)
dat <- cbind(simcausaldat$A1_0, simcausaldat$A2_0, simcausaldat$A3_0,
             simcausaldat$Y1_20, simcausaldat$Y2_20, simcausaldat$Y3_20)
```

```
colnames(dat) <- c("A1", "A2", "A3", "Y1", "Y2", "Y3")
dat <- as.data.frame(dat)
R.matrix <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
permat <- permutations(2, 3, c(0,1), repeats.allow = T)
marginalY1 <- permat[which(permat[,1] == 1),]
print(marginalY1) ## note that values in the first column (Y1) are all 1.
#>      [,1] [,2] [,3]
#> [1,]    1    0    0
#> [2,]    1    0    1
#> [3,]    1    1    0
#> [4,]    1    1    1
result.chain1 <- chain.causal.multi(targetoutcome = marginalY1,
                                    treatment = treatments,
                                    inputY = cbind(dat$Y1, dat$Y2, dat$Y3),
                                    inputA = cbind(dat$A1, dat$A2, dat$A3),
                                    listC = NULL, R.matrix = R.matrix,
                                    E.matrix = diag(3), edgeinfo = NULL,
                                    n.obs = 500, n.burn = 100)$causalprob
print(c(true = results1, estimate = result.chain1))
#> true.Mean_Y1_20        estimate
#>        0.898800        0.908206
```

Likewise we also estimate $Y_2(\mathbf{a})$ and $Y_3(\mathbf{a})$ by replacing `targetoutcome` matrix.

```
# Print out the results of Y_{2}(a) and Y_{3}(a)
print(c(true = results2, estimate = result.chain2))
#> true.Mean_Y2_20        estimate
#>        0.533400        0.518136
print(c(true = results3, estimate = result.chain3))
#> true.Mean_Y3_20        estimate
#>        0.325400        0.348916
```

We have observed that bias decreases and coverage rate increases as the number of simulated data (`sample.size`) increases.

Identifying influential units

A function `causal.influence` takes the same input as `chain.causal.multi` except that a specific `treatment` is replaced by `Avalues` that will eventually generate a set of treatment assignments that treat (`Avalues` $= 1$) one unit while controlling (`Avalues` $= 0$) all the others. In this way, `causal.influence` evaluates the average of collective outcomes under each treatment assignment $E\big[\mathbf{Y}(\mathbf{a}_j)\big]$ as a measure of influence of unit j, where $\mathbf{a}_j$ indicates the sole intervention of unit j, but `targetoutcome` can be a vector, matrix or an integer as the case of `chain.causal.multi`.

```
set.seed(1234)
edgeinfo = list(rbind(c("Y", 1), c("C", 1)), rbind(c("Y", 2), c("C", 2)),
                rbind(c("Y", 3), c("C", 3)))
influence = causal.influence(targetoutcome = "mean", Avalues = c(1,0),
                             inputY, inputA, listC = inputC, R.matrix, E.matrix = diag(3),
                             edgeinfo = edgeinfo, n.obs = 1000, n.burn = 100)
print(influence)
#> $influence
#> [1] 0.5924533 0.6999467 0.7095200
#>
```

```
#> $n.par
#> [1] 11
#>
#> $par.est
#>  [1]  2.5175370 -0.2081631 -0.4115525  1.4919613  0.5595840  0.1569165
#>  [7]  1.4209752  1.4663121 -3.1785419 -1.1046469 -0.8115134
```

The above result says that $E\big[\bar{\mathbf{Y}}(\mathbf{a}_3)\big] \approx 0.71$ achieves the highest influence, indicating that among three, treating unit $i = 3$ would result highest average potential outcomes.