732A96 Lab 1

Emil K Svensson

2017-09-11

Question 1

```
library(bnlearn)
```

```
## Warning: package 'bnlearn' was built under R version 3.3.2
```

```
##
## Attaching package: 'bnlearn'
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
library(gRain)
```

```
## Loading required package: gRbase
```

```
## Warning: package 'gRbase' was built under R version 3.3.2
```

```
##
## Attaching package: 'gRbase'
```

```
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, parents
```

```
myasia <- asia
myalarm <- alarm

sapply(1:5, FUN = function(X) {

  set.seed(X)
  myFirstHc<- hc(myalarm, score = "bde", restart = 15)

  set.seed(X + 10)
  myFirstHc2<- hc(myalarm, score = "bde",restart = 15)

  cp1 <- cpdag(myFirstHc)
  cp2 <- cpdag(myFirstHc2)
  all.equal(cp1,cp2)
})
```

```
## [1] "Different number of directed/undirected arcs"
## [2] "Different number of directed/undirected arcs"
## [3] "Different number of directed/undirected arcs"
## [4] "Different number of directed/undirected arcs"
## [5] "Different number of directed/undirected arcs"
```

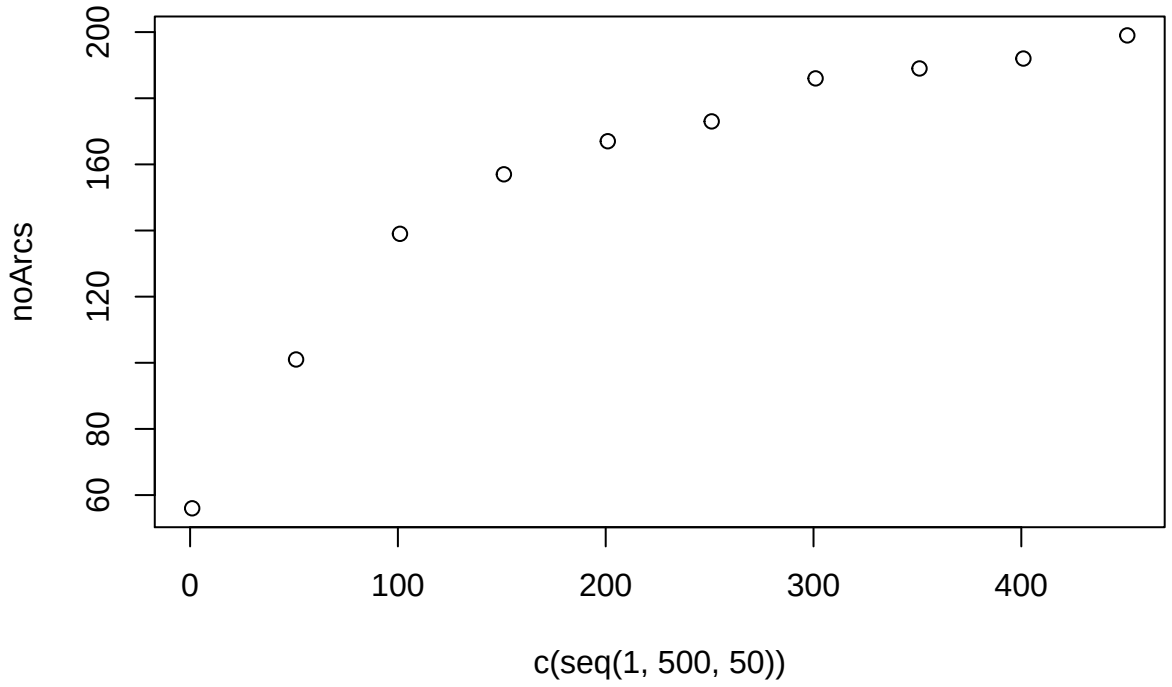
As we see from the output at some of these we reach different essential graphs even though the Hill-Climbing algorithm is determenistic. This is because we specify that we want to randomly start at different graphs

at different times which will make the HC-algorithm (which is a greedy algorithm) end in different end solutions. If we wouldn't do these random restarts we would end up with the same result since the algorithm is deterministic.

Assignment 2

```
noArcs<- sapply(c(seq(1,500,50)), FUN = function(X){
  nrow(hc(myalarm, score = "bde",iss = X)$arcs)
  })

plot(y=noArcs,x = c(seq(1,500,50)))
```



In the plot above we can see that the number of arcs are increasing with the increased imaginary sample size. This is because we put more weight on the prior probabilities in the BDeu score which are uniform i.e. a equal chance of having an edge comparing to not having an edge. Stressing this prior forces the data to be the decider between adding an edge or not and in these cases data tends to infer adding an edge rather than not.

Assignment 3

```
HCasia<- hc(myasia,score = "bde")
bnMFH<-bn.fit(HCasia,myasia)
gMFH<- as.grain(bnMFH)
cMFH<- (gMFH)


# Exact
querygrain(setEvidence(cMFH, c("E"), list(c("yes"))))$X
```

```
## X
```

```
##          no         yes
## 0.005405405 0.994594595
```

```
# Approx.
cpquery(bnMFH,event = (X == "yes"), evidence = (E == "yes"), n = 5000)
```

```
## [1] 0.997319
```

```
# Approx.
cpquery(bnMFH,event = (X == "yes"), evidence = (E == "yes"), n = 5000)
```

```
## [1] 0.9896641
```

Here we can see the probability that a X-ray has been conducted given that the patient has observed yes on the variable E (**tuberculosis versus lung cancer/bronchitis**) . The first output is exact and the two other are approximate. The approximated probabilities are just sampled from the full joint probability distribution and not fully calculated like in the exact and are therefor not the same every time.

```
numsamp <- 500000

noevent <- cpdist(bnMFH,nodes = "X",
        evidence = TRUE, n = numsamp)

cat(paste("With no nodes:",nrow(noevent)/numsamp))
```

```
## With no nodes: 1
```

```
twoevents <- cpdist(bnMFH,nodes = "X",
                    evidence = (E == "yes") & (B == "no"),
                    n = numsamp)

cat(paste("With two nodes:",nrow(twoevents)/numsamp))
```

```
## With two nodes: 0.026176
```

In the print out above you see the share of number of requested samples compared to what was given. This indicates that the functions sample from the full joint posterior and then for the cases we have discards those not associated with the conditional posterior we are trying to sample from when we add evidence.

Assignment 4

```
nograph <- 5000
nodes <- letters[1:5]
burn <- c(100,1000,100000)
evr  <- c(50,150,300)


result <- apply(cbind(burn,evr), MARGIN = 1, FUN = function(X) {

setofgraphs <-  random.graph(nodes, num = nograph, method ="ic-dag",
                             burn.in = X[1], every = X[2])

EssGraphs <- lapply(setofgraphs,FUN = cpdag)

length(unique(EssGraphs))
```

```
}
)


result/nograph
```

```
## [1] 0.6198 0.6188 0.6112
```

So around 60 % of the 5000 DAGs generated were unique so we aproximatly reduce the searchspace with up to 40 % (since we don't know how the edge cuts we can't say exactly how much of the searchpace we reduce.) which would could me considerd a good improvment. This could of course be even more since it is just a sample, in the group report we found only 30 % of the DAGs were unique.

The downside of using structure learning in the space of essential graphs would be that your forcing yourself to some extent to work with undirected edges, but in a setting where we have large amounts of data it would only be minor setback compared to the speed up.