# Lab 1 - Advanced Machine Learning

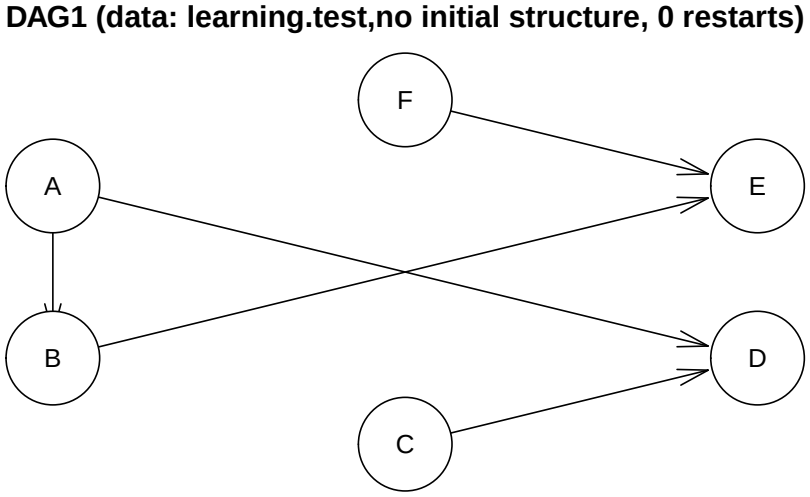
*Gustav Sternelöv*

*September 2, 2016*

## 1. Show that multiple runs hill-climbing algorithm can return non-equivalent DAGs. Can we check if two DAGs are equivalent by checking if they receive the same score ?

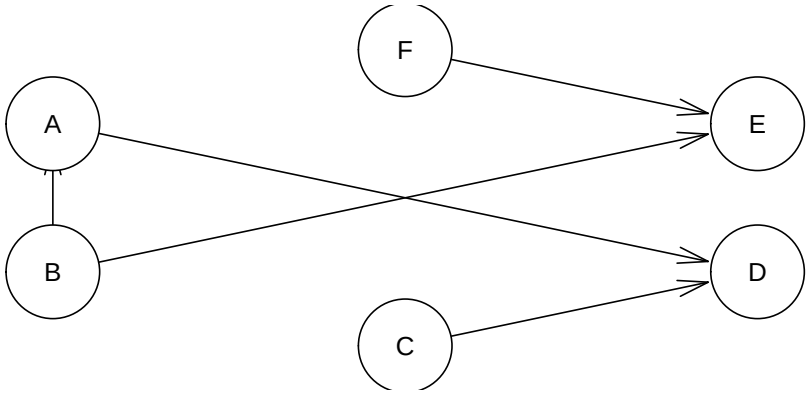
In order to investigate the results given by the hill-climbing algorithm are the data sets *learning.test* and *asia* used. Moreover, the results with and without an specified initial structure is compared and the number of restarts parameter is also configured.

First, a structure is created from the *learning.test* data set with the *hc* algorithm and with the restart parameter set to 0. The graphical representation of the model is shown below.




Then, a new structure is created on the same data set but with the earlier structure as initial structure and the number of restarts set to 100.

**DAG2 (data: learning.test, DAG1 as initial structure, 100 restarts)**




It can be seen that the DAGs received for the *learning.test* data almost is the same for both cases. It is the arrow for the link between node **A** and **B** that has changed direction for the second DAG. This can also be confirmed by using the function *all.equal*.

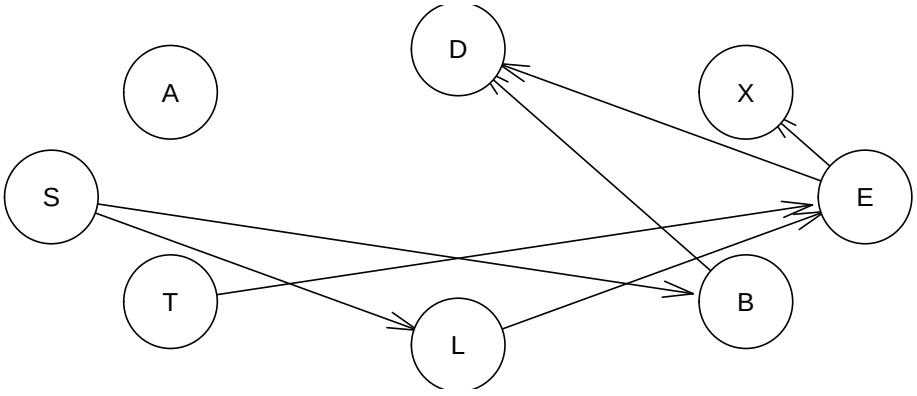
```
all.equal(lt1, lt2)
```

```
## [1] "Different arc sets"
```

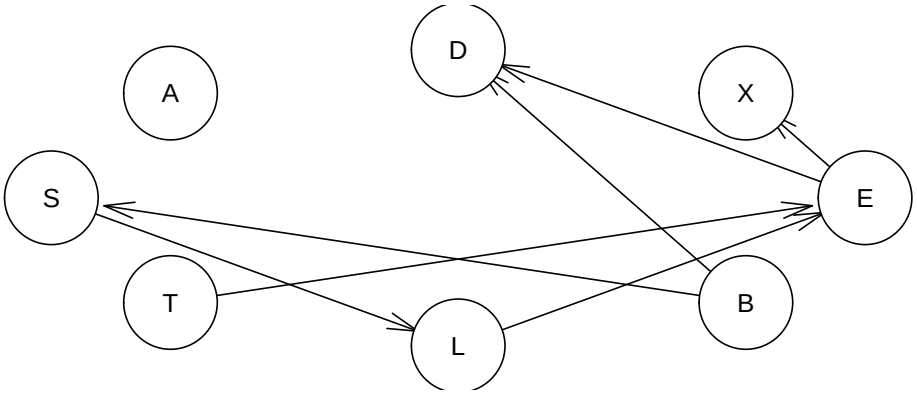
The score for the first DAG is $-23967.65$ and for the second DAG the score is the same, $-23967.65$. Hence, the score is equivalent for both DAGs even though the DAGs not are equal.

Then, the same procedure is repeated for the *Asia* data set. The DAG given for the data set by the hc algorithm with no initial structure and 0 restarts is shown below.

**DAG3 (data: Asia, no initial structure, 0 restarts)**




With the earlier structure as initial structure and number of restarts equal to 100, the following structure is obtained.




The difference between DAG3 and DAG4 is the arrow for the link between node **S** and node **B**. This inequality is tested and it can be confirmed that the DAGs not are equivalent.

```
all.equal(as1, as2)
```

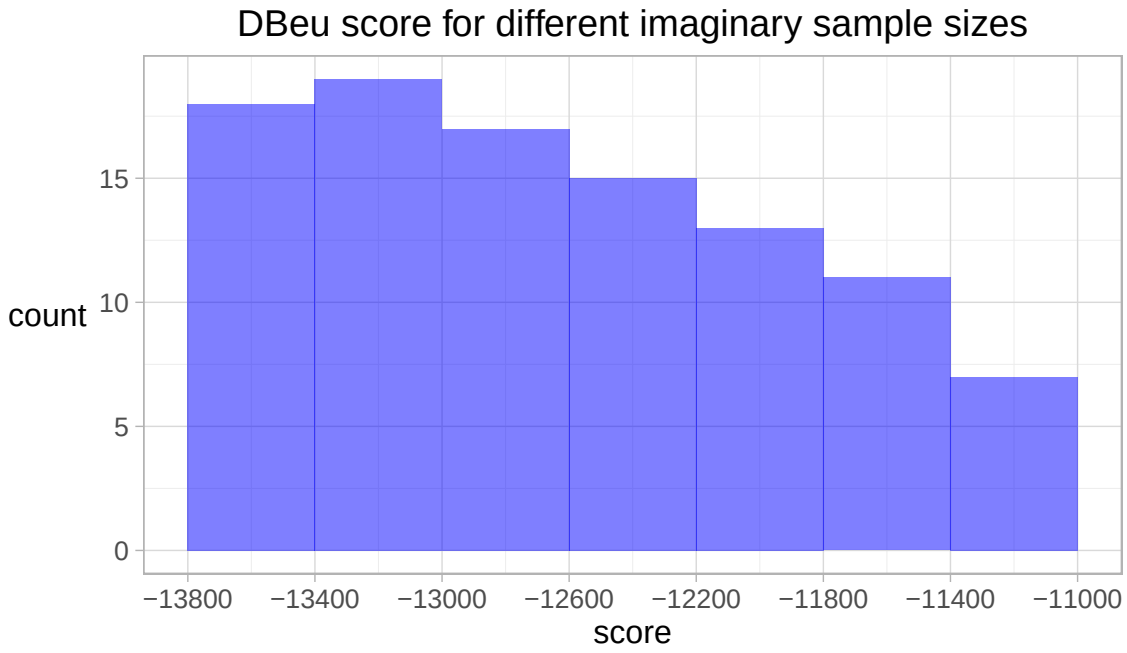
```
## [1] "Different arc sets"
```

The score for DAG3 is $-11147.65$ and for DAG4 the score is $-11147.65$. Again, the case is that the DAGs not are equivalent but they have the same score.

Since the score depends on the independencies in the model and not is influenced by how the arrows are directed it cannot be used for answering if two DAGs are equivalent or not. Two different DAGs can have the same score if they have the same independencies. This was the case for the DAGs given for both the *learning.test* and the *asia* data set. Therefore, the score cannot be used for checking the equivalence of DAGs.

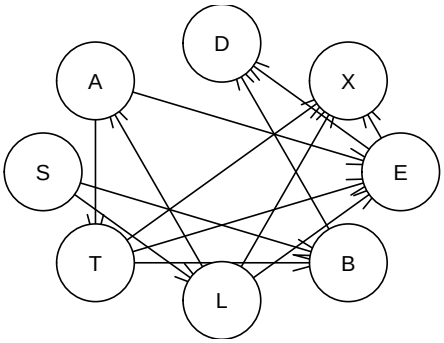
## 2. Show that increasing the equivalent sample size (a.k.a imaginary sample size) in the BDeu score decreases regularization. Explain why this happens.

For the data set *asia* is the *BDeu* score computed for the structure given by the hc algorithm with the number of restarts set to 10. The score is calculated for all imaginary sample sizes in the interval 10 to 1000 by steps of 10. All of these scores are presented in the histogram below.

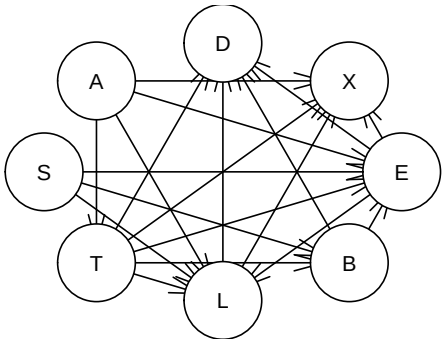



For higher values of the imaginary sample size are the scores more and more similar. So, as the imaginary sample size increases the resulting DAGs becomes more and more similar. This can also be proven by plotting some DAGs with different imaginary sample sizes. In the plot on the following page are four different DAGs shown. The first is a DAG given when the imaginary sample size is set to 10, the second when it is set to 100 and the third when the sample size is set to 500. It can easily be seen that as the imaginary sample size increases, the regularization decreases. The result of this is that the number of links in the DAG, the complexity of the model, increases.
The fourth plot is the true DAG and it is notable that this DAG has much fewer links compared to those with a high imaginary sample size. It is most similar to the first DAG while the third DAG, with ISS=500, is very far of the true DAG.

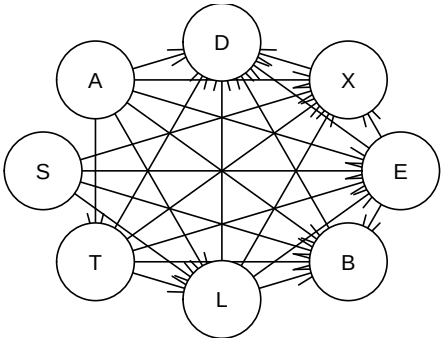
**ISS=10**



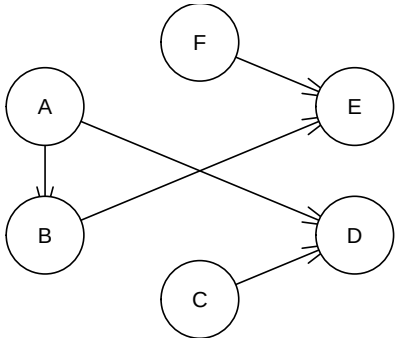

**ISS=100**




**ISS=500**




**True DAG**



## 3. You already know the LS algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally. Compare the answers given to some queries by exact and approximate inference algorithms. When running the approximate algorithm several times, why may we obtain different answers ? Is the approximate algorithm equally accurate when the query includes no observed nodes and when it includes many observerd nodes ?

In this assignment I have chosen to use the data set *learning.test* and the structure in DAG2 from the first section in the report. ## i) To start with, I look at the probabilities for node **A** when there are no evidence. The output below contains the results for the approximate method first and then also for the exact method. There is no big difference between the results given by the respective method.

```
## evi_cb
##      a      b      c
## 0.3344 0.3392 0.3264

## $A
## A
##      a      b      c
## 0.3336 0.3340 0.3324
```

### ii)

Next, the results of the methods are compared for the node $A$ given the evidence that the node $D$ is equal to $c$. By looking at the tables below it can again be seen that the difference is small between the results for the methods.

```
## evi_cb
##         a         b         c
## 0.2384363 0.1065354 0.6550283

## $A
## A
##         a         b         c
## 0.2413185 0.1023344 0.6563470
```

### iii)

To test whether the approximate method returns different results if the same query is repeated several times, the case investigated in *i)* is repeated four more times. The results are presented below and the obtained results do differ. This is because the function that uses the approximate method, *cpdist*, is based on Monte Carlo particle filters. The effect of this is that the results returned by the function may be slightly different when the same query is executed several times.

```
## evi_cb
##      a      b      c
## 0.3329 0.3324 0.3347
```

```
## evi_cb
##      a      b      c
## 0.3384 0.3283 0.3333
```

```
## evi_cb
##      a      b      c
## 0.3329 0.3329 0.3342
```

```
## evi_cb
##      a      b      c
## 0.3377 0.3362 0.3261
```

## iiii)

The last comparsion between the approximate method and the exact method is for the case when many nodes are observed. Presented by the tables below is the probabilities for the node **B** given that the node **A**$==b$, **C**$==a$, **D**$==c$, **F**$==a$, **E**$==a$. Compared to the earlier results is the difference between the approximate and the exact method clear. It seems like the approximate method is less accurate when the query includes a higher number of observed nodes.

```
## evi_cb
##          a          b          c
## 0.79166667 0.15277778 0.05555556
```

```
## $B
## B
##          a          b          c
## 0.80755207 0.10254077 0.08990715
```

## 4. There are 29281 DAGs with five nodes. Compute approximately the fraction of the 29281 DAGs that represent different independence models. How does this affect structure learning ?

This is accomplished by using the *random.graph* function. Two of the parameters for the function, *burn.in* and *every*, are configured in order to achieve a good result. In each test are 30 000 random graphs produced, the default burn-in period is 150 and the default value of *every* is 1.

In the first test are no changes made to the default settings and the number of unique essential graphs returned is 6081. When the burn-in period is set to 1000 and *every* still at the default value, 6036 EGs are returned. In the third test is a low value, 0.1, of *every* tested together with a burn-in of 1000 and it results in only 1369 essential graphs being returned. The fourth and final test examines the effect of setting *every* to a higher value, 10. The result of this is that 7480 are returned.

To summarize these tests it can be conluded that it does not seem to be any clear effect of changing the burn-in period. At least not for this set of tests. Changing the value of *every* do on the other hand have a strong effect where a low value returns a substantially lower amount of essential graphs.

Depending on what settings that are chosen the number of unique essential graphs differ. The lowest number of different EGs was 1357 and the highest 7483. However, in each of the tests the case is that the same EGs represent several different DAGs which makes the structure learning part a bit problematic.

When you choose what seems to be the most appropriate EG, there is a risk that what actually is the best DAG is missed or that since all these potential DAGs have the same score, the wrong one may be picked. On the other hand, the number of potential DAGs is very high so it is not an realistic option to generate and compare all different DAGs. Hence, one problem with bayesian networks and the causal links between the nodes is that they are hard to find since several different DAGs have the same independencies, the same EG.

## R code

```
## ---- echo=FALSE, warning=FALSE, message=FALSE--------------------------
library(bnlearn)
library(gRain)
library(ggplot2)

data(learning.test)
data(asia)
options(scipen = 10)

## ---- echo=FALSE, fig.height=3.5, fig.width=8, fig.align='center'-------- 1
## learning.test data
set.seed(1897)
lt1 <- hc(learning.test, restart = 0)
par(mar = c(1.1, 4.1, 2.1, 2.1))
plot(lt1, main = "DAG1 (data: learning.test,no initial structure, 0 restarts)")
ltS1 <- round(score(lt1, data = learning.test, type = "bde"), 2)
# vstructs(lt1) cpdag(lt1)

## ---- echo=FALSE, fig.height=3.5, fig.width=8, fig.align='center'--------
set.seed(1897)
lt2 <- hc(learning.test, start = lt1, restart = 100)
par(mar = c(1.1, 4.1, 4.1, 2.1))
plot(lt2, main = "DAG2 (data: learning.test, DAG1 as initial structure, 100 restarts)")
ltS2 <- round(score(lt2, data = learning.test, type = "bde"), 2)
# vstructs(lt2) cpdag(lt2)

## ---- echo=TRUE---------------------------------------------------------
all.equal(lt1, lt2)

## ---- echo=FALSE, fig.height=3.5, fig.width=8, fig.align='center'--------
## asia data set
set.seed(3110)
as1 <- hc(asia, restart = 0)
par(mar = c(1.1, 4.1, 4.1, 2.1))
plot(as1, main = "DAG3 (data: Asia, no initial structure, 0 restarts)")
aS1 <- round(score(as1, data = asia, type = "bde"), 2)
# vstructs(as1) cpdag(as1)

## ---- echo=FALSE, fig.height=3.5, fig.width=8, fig.align='center'--------
set.seed(3110)
as2 <- hc(asia, start = as1, restart = 100)
par(mar = c(1.1, 4.1, 4.1, 2.1))
plot(as2, main = "DAG4 (data: Asia, DAG3 as initial structure, 100 restarts)")
aS2 <- round(score(as2, data = asia, type = "bde"), 2)
# vstructs(as2) cpdag(as2)

## ----echo=TRUE----------------------------------------------------------
all.equal(as1, as2)

## ---- echo=FALSE, fig.height=3.5, fig.width=6, fig.align='center'-------- 2
## asia data set
```

```r
as1_2 <- hc(asia, score = "bde", restart = 10)

sSize <- seq(10, 1000, 10)
ScoreF <- data.frame(iss = seq(10, 1000, 10), score = NA)
for (i in 1:100) {
    ScoreF[i, 2] <- score(as1_2, data = asia, type = "bde", iss = ScoreF[i,
        1])
}
ggplot(ScoreF, aes(x = score)) + geom_histogram(binwidth = 400, alpha = 0.5,
    fill = "blue") + theme_light() + scale_x_continuous(breaks = c(seq(-11000,
    -14000, -400))) + theme(axis.title.y = element_text(angle = 0)) + ggtitle("DBeu score for different

## ---- echo=FALSE, fig.height=6, fig.width=8, fig.align='center'----------
par(mar = c(1.1, 4.1, 4.1, 2.1))
par(mfrow = c(2, 2))
as1_2 <- hc(asia, score = "bde", restart = 10, iss = 10)
as2_2 <- hc(asia, score = "bde", restart = 10, iss = 100)
as3_2 <- hc(asia, score = "bde", restart = 10, iss = 500)
plot(as1_2, main = "ISS=10")
plot(as2_2, main = "ISS=100")
plot(as3_2, main = "ISS=500")
# True DAG
res = empty.graph(names(learning.test))
modelstring(res) = "[A][C][F][B|A][D|A:C][E|B:F]"
plot(res, main = "True DAG")


## ---- echo=FALSE----------------------------------------------------------
ltFit <- bn.fit(lt2, learning.test)
# approximate method, no evidence
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = TRUE)
prop.table(table(evi_cb))

# Exact method, no evidence
ltFit_grain <- as.grain(ltFit)
ltFit_grain <- compile(ltFit_grain)

ltFit_grain1 <- setFinding(ltFit_grain, nodes = c("A"), states = TRUE)
querygrain(ltFit_grain1)[1]

## ---- echo=FALSE----------------------------------------------------------
## Approximate method, with evidence
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = (D == "c"))
prop.table(table(evi_cb))

# Exact method, with evidence
ltFit_grain2 <- setEvidence(ltFit_grain, nodes = c("D"), states = c("c"))
querygrain(ltFit_grain2)[1]

## ---- echo=FALSE----------------------------------------------------------
## Approximate method for several different seeds
set.seed(814)
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = TRUE)
```

```r
prop.table(table(evi_cb))

set.seed(311015)
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = TRUE)
prop.table(table(evi_cb))

set.seed(1991)
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = TRUE)
prop.table(table(evi_cb))

set.seed(1897)
evi_cb <- cpdist(ltFit, nodes = c("A"), evidence = TRUE)
prop.table(table(evi_cb))

## ---- echo=FALSE-----------------------------------------------------------
## Approximate versus Exact when many observed nodes approximate method, no
## evidence
set.seed(800014)
evi_cb <- cpdist(ltFit, nodes = c("B"), evidence = c(A == "b" & C == "a" & D ==
    "c" & F == "a" & E == "a"))
prop.table(table(evi_cb))

# Exact method, no evidence
ltFit_grain3 <- setFinding(ltFit_grain, nodes = c("A", "C", "D", "F", "E"),
    states = c("b", "a", "c", "a", "a"))
querygrain(ltFit_grain3)[2]

## ---- echo=FALSE-----------------------------------------------------------
gr_1 <- random.graph(nodes = c("A", "B", "C", "D", "E"), num = 30000, method = "melancon")
gr_list <- list()
for (i in 1:length(gr_1)) {
    gr_list[[i]] <- cpdag(gr_1[[i]])
}
Lgr_1 <- length(unique(gr_list))

gr_2 <- random.graph(nodes = c("A", "B", "C", "D", "E"), num = 30000, method = "melancon",
    burn.in = 1000)
gr_list <- list()
for (i in 1:length(gr_2)) {
    gr_list[[i]] <- cpdag(gr_2[[i]])
}
Lgr_2 <- length(unique(gr_list))

gr_3 <- random.graph(nodes = c("A", "B", "C", "D", "E"), num = 30000, method = "melancon",
    burn.in = 1000, every = 0.1)
gr_list <- list()
for (i in 1:length(gr_3)) {
    gr_list[[i]] <- cpdag(gr_3[[i]])
}
Lgr_3 <- length(unique(gr_list))

gr_4 <- random.graph(nodes = c("A", "B", "C", "D", "E"), num = 30000, method = "melancon",
    burn.in = 1000, every = 10)
```

```
gr_list <- list()
for (i in 1:length(gr_4)) {
    gr_list[[i]] <- cpdag(gr_4[[i]])
}
Lgr_4 <- length(unique(gr_list))

## ----code=readLines(knitr::purl('C:\\Users\\Gustav\\Documents\\AdvML\\Lab1\\Lab1_1.Rmd',documentation
## = 1)), eval = FALSE, tidy=TRUE----
```