Advanced Machine Learning - Lab 1 - individual report

Nicolas Taba (nicta839)

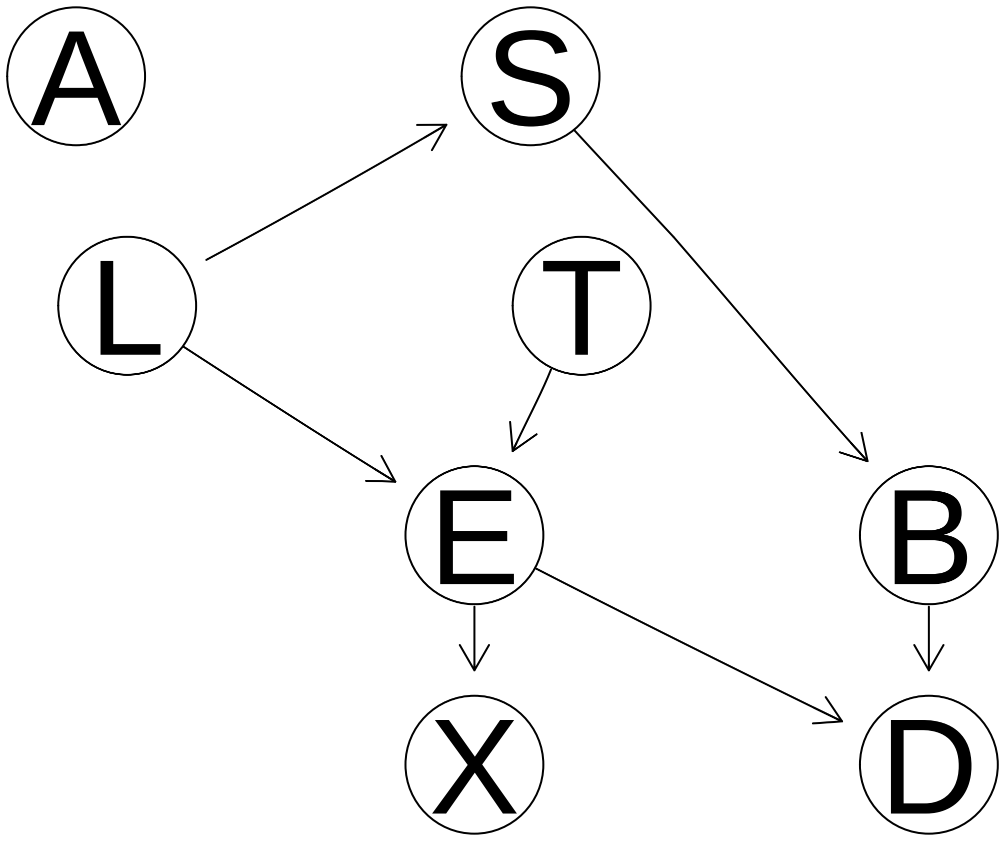
31/08/2021

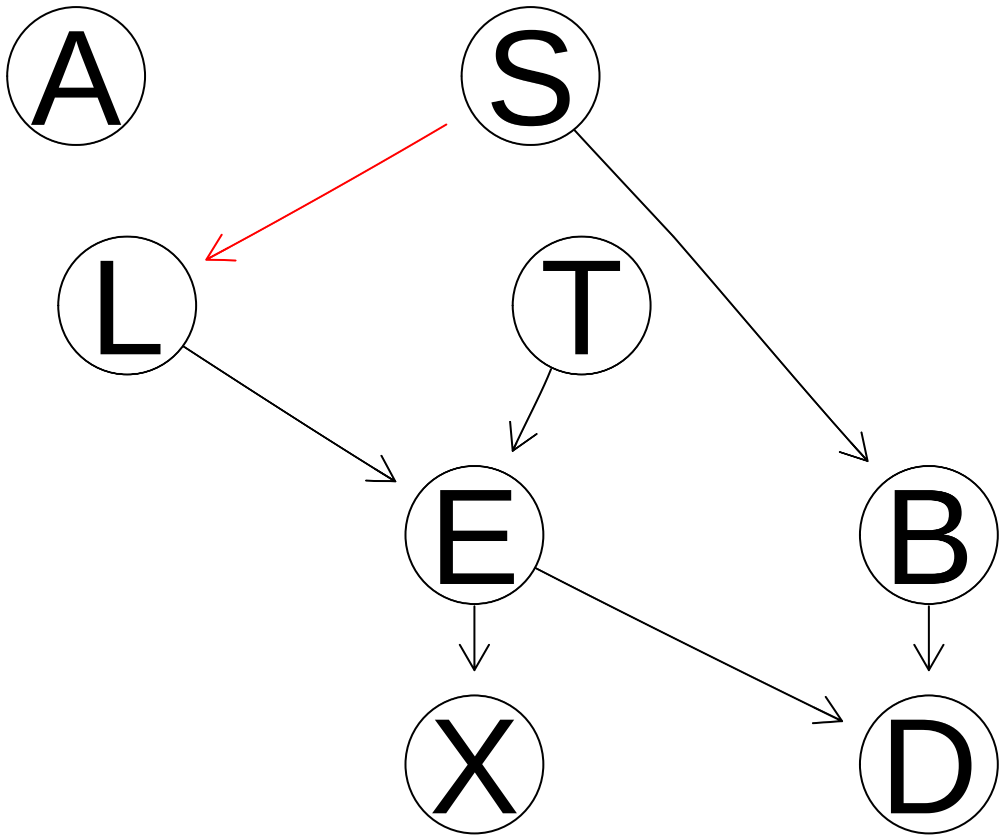
Question 1

```r
data("asia")
# set.seed(12345)

#generate bayesian networks
bn1_bic <- hc(asia, restart = 100, score = 'bic')
bn2_bic <- hc(asia, restart = 100, score = 'bic')

all.equal(bn1_bic, bn2_bic)
```

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

```r
graphviz.compare(bn1_bic, bn2_bic)
```





```
cat("network 1 bde score: ", bnlearn::score(bn1_bic, asia, type = 'bde'))
```

```
## network 1 bde score:  -11095.79
```

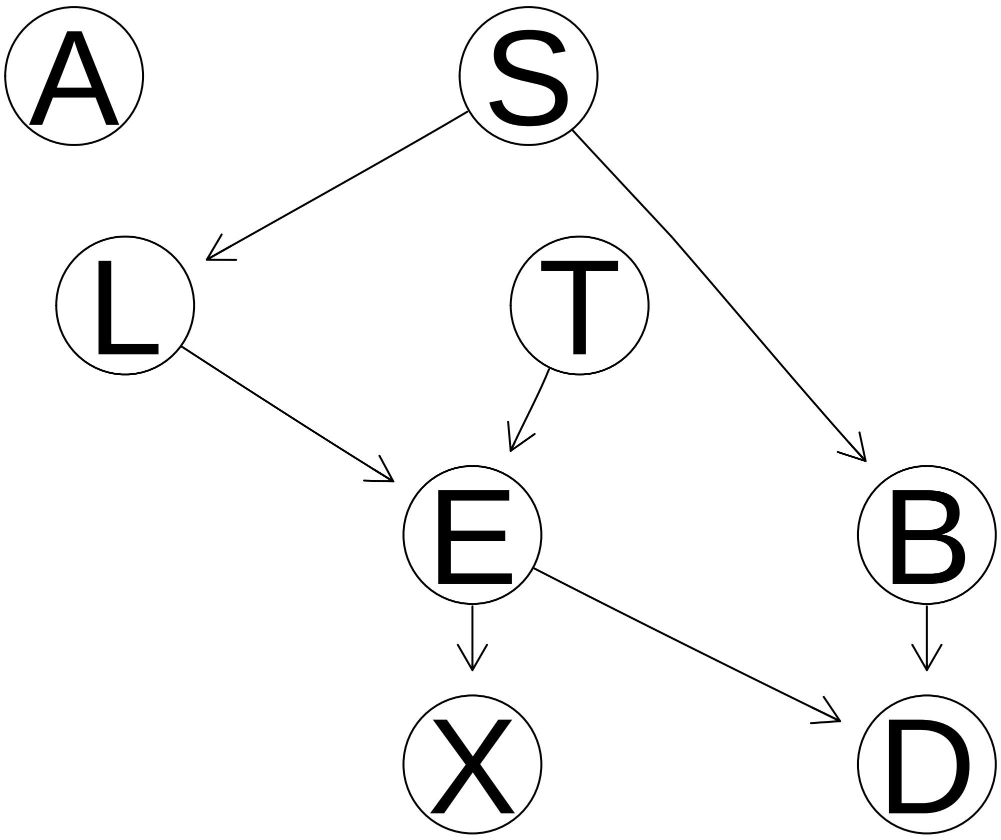
```
cat("\nnetwork 2 bde score; ", bnlearn::score(bn2_bic, asia, type = 'bde'))
```

```
##
## network 2 bde score;  -11095.79
```

```
# set.seed(12345)
bn1_bde <- hc(asia, restart = 100, score = 'bde')
bn2_bde <- hc(asia, restart = 100, score = 'bde')

all.equal(bn1_bde, bn2_bde)
```

```
## [1] TRUE
```

```
graphviz.compare(bn1_bde, bn2_bde)
```



```r
cat("network 1 bde score: ", bnlearn::score(bn1_bde, asia, type = 'bde'))
```

```
## network 1 bde score:  -11095.79
```

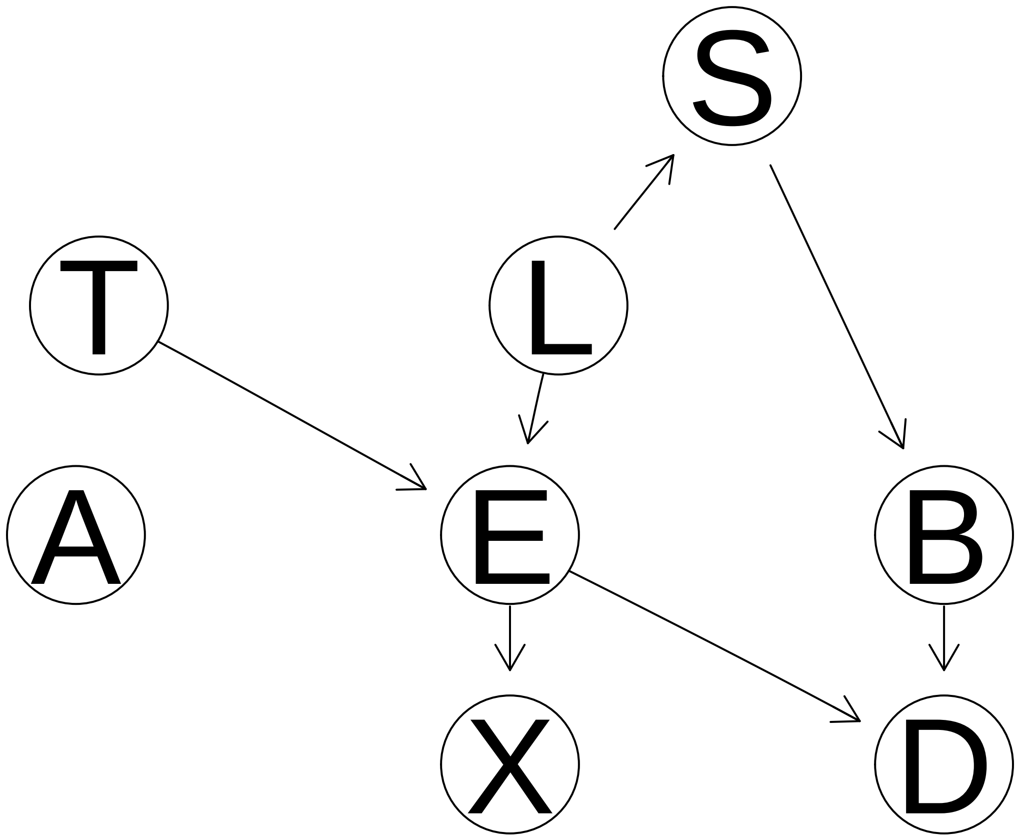
```r
cat("\nnetwork 2 bde score; ", bnlearn::score(bn2_bde, asia, type = 'bde'))
```

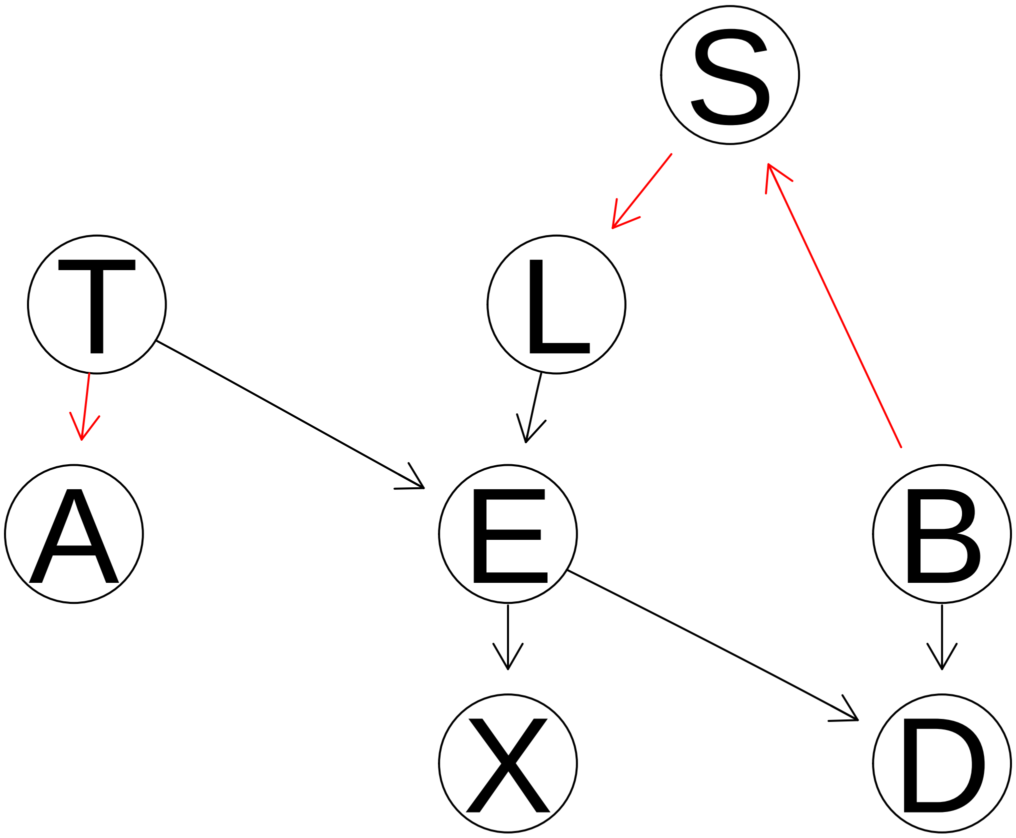
```
##
## network 2 bde score;  -11095.79
```

```r
bn1_iss <- hc(asia, restart = 100, iss = 1, score = 'bde')
bn2_iss <- hc(asia, restart = 100, iss = 2, score = 'bde')

all.equal(bn1_iss, bn2_iss)
```

```
## [1] "Different number of directed/undirected arcs"
```

```r
graphviz.compare(bn1_iss, bn2_iss)
```


S
T
L
A
E
B
X
D



```
cat("network 1 bde score: ", score(bn1_iss, asia, type = 'bde'))
```

```
## network 1 bde score:  -11095.79
```

```
cat("\nnetwork 2 bde score; ", score(bn2_iss, asia, type = 'bde'))
```

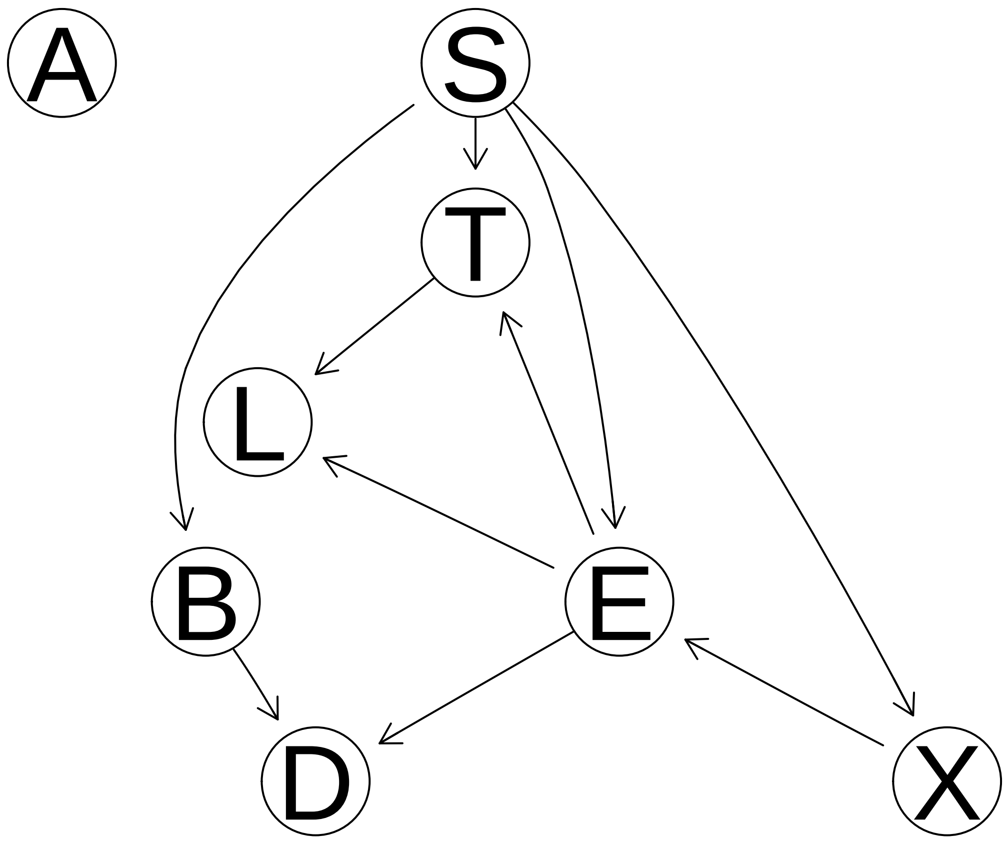
```
##
## network 2 bde score;  -11096.64
```

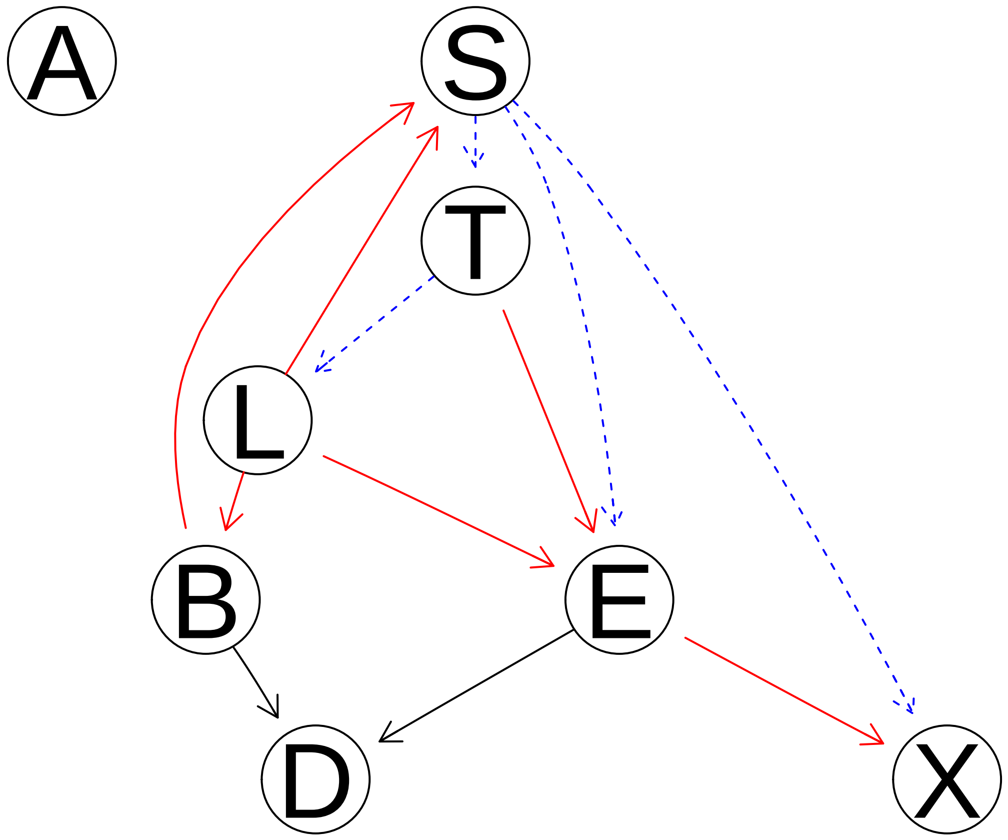
When using the same metrics to compare the hill-climbing algorithm, we are returned with equivalent graphs where the causal direction between some nodes is different, but the graph presents the same unshielded colliders. The graphs are equivalent as their BDE scores are the same. In the case where we use different imaginary sample sizes, we also see that the hill-climbing algorithm performs differently when sample sizes are different and creates new graphs with new causal relationships. This implies that for different sizes of datasets, we obtain different results.

We could also try to have different initial graphs and check if the graphs are the same.

```
set.seed(12345)
g1 = random.graph(colnames(asia))
# set.seed(12345)
g2 = random.graph(colnames(asia))

set.seed(12345)
hc1 = hc(asia, g1)
set.seed(12345)
hc2 = hc(asia, g2)
```

```
graphviz.compare(hc1, hc2)
```





```
all.equal(hc1, hc2)
```

```
## [1] "Different number of directed/undirected arcs"
```

```
cat("network 1 bde score: ", score(hc1, asia, type = 'bde'))
```

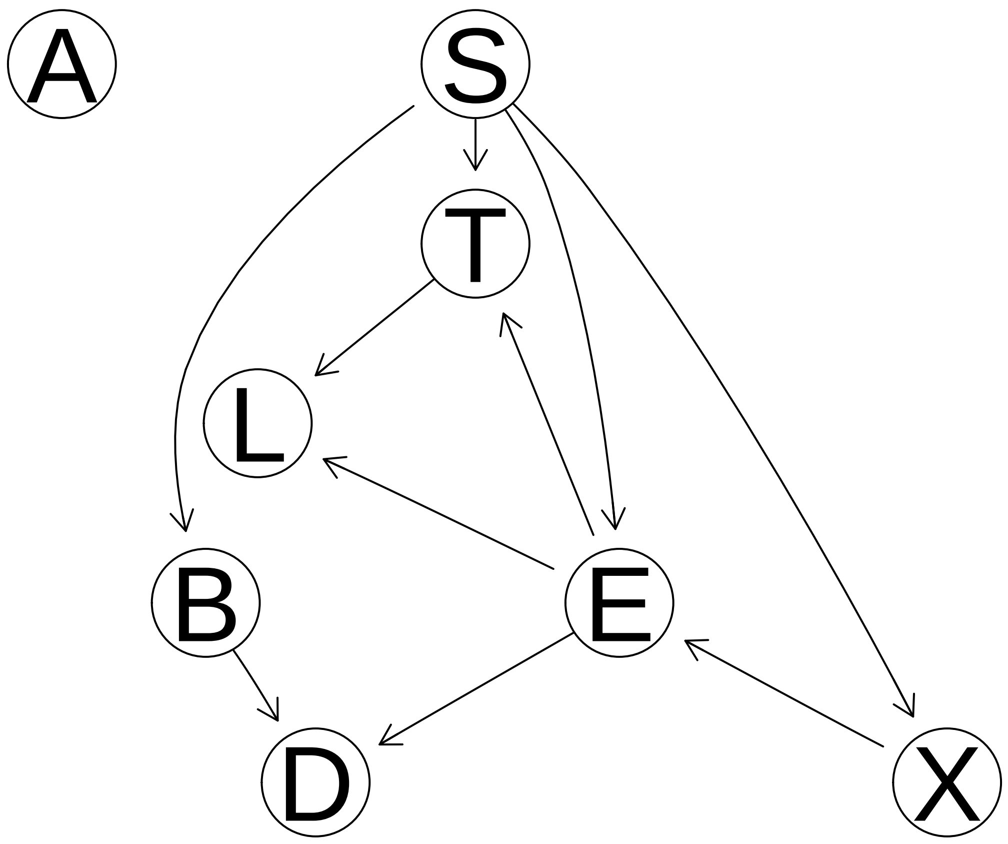
```
## network 1 bde score:  -11107.79
```

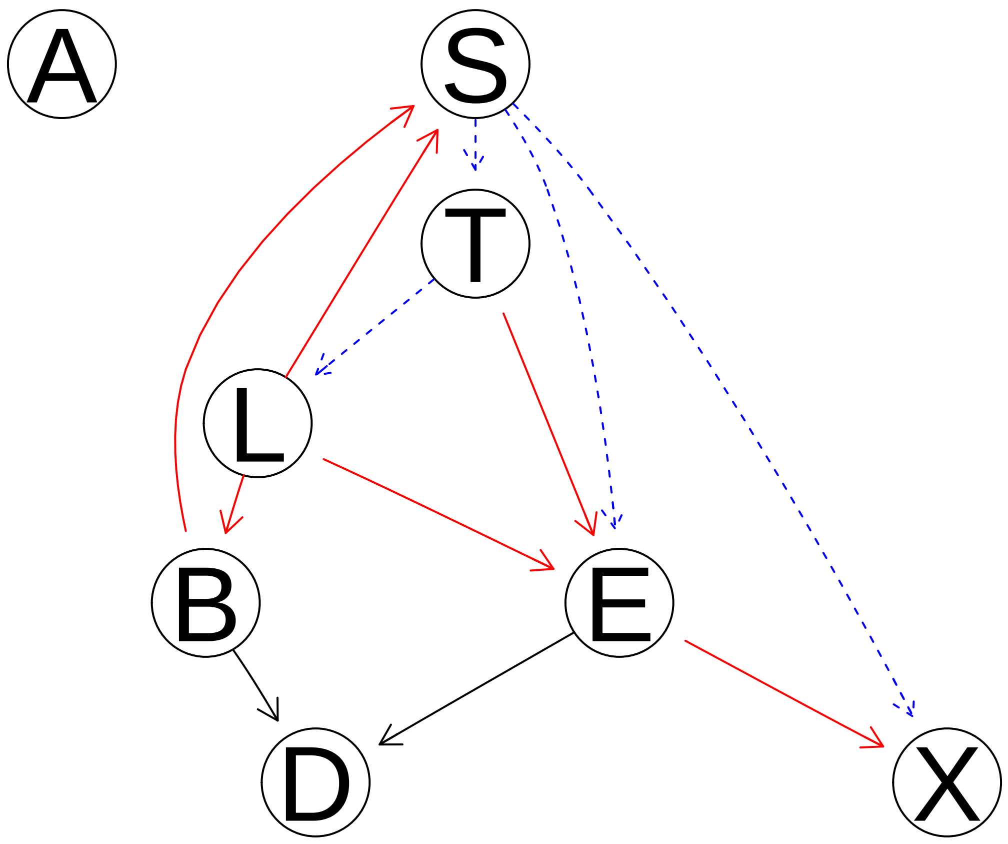
```
cat("\nnetwork 2 bde score; ", score(hc2, asia, type = 'bde'))
```

```
##
## network 2 bde score;  -11101.59
```

We now produce different graphs when we initialize the nodes differently for the same hill-climbing process. This is because the hill-climbing algorithm is greedy and reaches only local optimas. Initializing the algorithm (original graph) changes what local optima we reach even if we initialize the hillclimbing algorithm in the same way.

```
set.seed(12345)
g1 = random.graph(colnames(asia))
# set.seed(12345)
g2 = random.graph(colnames(asia))

set.seed(12345)
hc1 = hc(asia, g1, score = "bde")
set.seed(12345)
hc2 = hc(asia, g2, score = "bde")

graphviz.compare(hc1, hc2)
```


A
S
T
L
B
E
D
X



```r
all.equal(hc1, hc2)
```

```
## [1] "Different number of directed/undirected arcs"
```

```r
cat("network 1 bde score: ", score(hc1, asia, type = 'bde'))
```

```
## network 1 bde score:  -11107.79
```

```r
cat("\nnetwork 2 bde score; ", score(hc2, asia, type = 'bde'))
```

```
##
## network 2 bde score;  -11101.59
```

Question 2

```r
# split the data
set.seed(12345)
n <- dim(asia)[1]
id <- sample(1:n, floor(n*0.8))
train <- asia[id, ]
test <- asia[-id, ]

# incremental assosciation of training data for PC
IAMB <- iamb(x = train)
#fit the model
model_fit <- bn.fit(IAMB, train)
# Into grain
bn_grain <- as.grain(model_fit)
```

```
#compile
bn_fin <- compile(bn_grain)

## predictions
# need a func (reuse for true network)
bn_func <- function(data, bn){
  pred <- rep(0, nrow(data))

  for(i in 1:nrow(data)){
    evid <- setEvidence(bn, nodes = colnames(data)[-2], states = data[i, -2])
    prob <- as.numeric(unlist(querygrain(evid, nodes = "S")))

    if(prob[1]>0.5){
      pred[i] <- "no"
    }else{
      pred[i] <- "yes"
    }
  }
  return(pred)
}

# setEvidence wants characters!!
test_fix <- apply(test, 2, as.character)

predict_bn <- bn_func(data = test_fix, bn_fin)
confusion <- table(test[,"S"], predict_bn)
confusion
```

```
##      predict_bn
##        no yes
##   no  346 167
##   yes 135 352
```

```
missclasification <- 1 - (sum(diag(confusion))/ sum(confusion))
missclasification
```

```
## [1] 0.302
```

```
#true network
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#same operations as above
true_fit <- bn.fit(dag, train)
true_grain <- as.grain(true_fit)
true_bn <- compile(true_grain)

pred_true <- bn_func(data = test_fix, true_bn)


# confusion matrix and accuracy
confusion_true <- table(test[,"S"], pred_true)
confusion_true
```

```
##      pred_true
##        no yes
##   no  337 176
##   yes 121 366
```

```
missclasification_true <- 1 - (sum(diag(confusion_true))/ sum(confusion_true))
missclasification_true
```

```
## [1] 0.297
```

Question 3

```
# Markov blanket of S

mb <- mb(model_fit, "S")

mb_predict <- rep(0, nrow(test_fix))

for (i in 1:nrow(test_fix)){
  evidence <- setEvidence(bn_fin, nodes = mb, states = test_fix[i, mb])
  prob <- as.numeric(unlist(querygrain(evidence, nodes = "S")))

  if(prob[1]>0.5){
    mb_predict[i] <- "no"
  }else{
    mb_predict[i] <- "yes"
  }
}

confusion_mb <- table(test$S, mb_predict)
confusion_mb
```

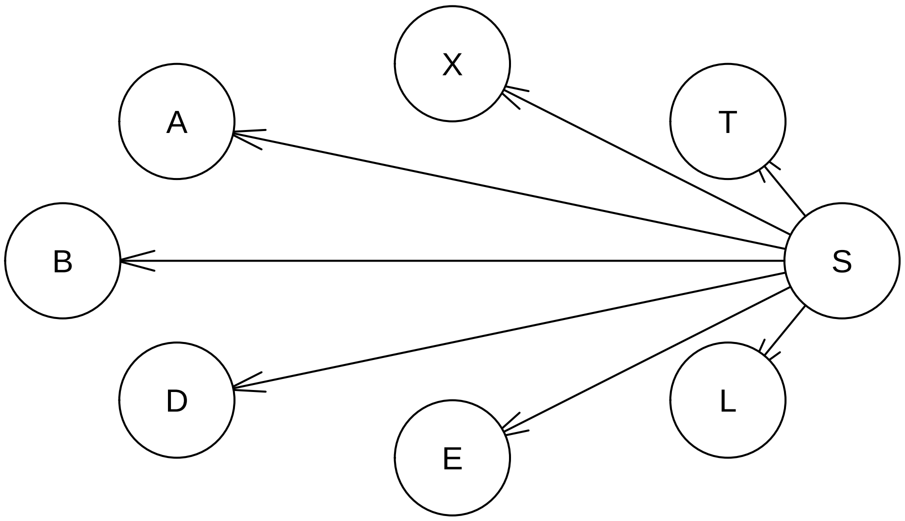
```
##      mb_predict
##         no yes
##   no   346 167
##   yes 135 352
```

```
missclass_mb <- 1 - (sum(diag(confusion_mb))/ sum(confusion_mb))
missclass_mb
```

```
## [1] 0.302
```

Question 4

```
# Naive bayes graph

naive_bayes = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(naive_bayes)
```



```
nbayes_fit <- bn.fit(naive_bayes, data = train)
nbayes_grain <- as.grain(nbayes_fit)
nbayes_compile <- compile(nbayes_grain)

nbayes_pred <- rep(0, nrow(test_fix))

for(i in 1:nrow(test_fix)){
  evidence <- setEvidence(nbayes_compile, nodes = c("A", "T", "L", "B", "E", "X", "D"), states = test_fi
  prob <- unlist(querygrain(object = evidence, node = "S"))

  if(prob[1]>0.5){
    nbayes_pred[i] <- "no"
  }else{
    nbayes_pred[i] <- "yes"
  }
}

confusion_nbayes <- table(test$S, nbayes_pred)
confusion_nbayes
```

```
##      nbayes_pred
##        no yes
##   no  359 154
##   yes 180 307
```

```
missclass_nbayes <- 1 - (sum(diag(confusion_nbayes))/ sum(confusion_nbayes))
missclass_nbayes
```

```
## [1] 0.334
```

Question 5

We obtain similar results for questions 2 and 3. It is reasonable to assume that to explain the variable S, the Markov Blanket assumption is enough to make inferences, i.e. the other variables (not parents and not children of S) do not influence in our final conditional probability on S. The Naive Bayes classifier in question 4 puts forth a strong assumption of independence between variables that is not likely to hold in this practical case. This strong assumption explains the reduced performance of this method observed in the missclassification rate.

Appendix: All code for this report

```r
knitr::opts_chunk$set(echo = TRUE)
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("RBGL")
BiocManager::install("Rgraphviz")
BiocManager::install("gRain")
library(bnlearn)
library(Rgraphviz)
library(gRain)
data("asia")
# set.seed(12345)

#generate bayesian networks
bn1_bic <- hc(asia, restart = 100, score = 'bic')
bn2_bic <- hc(asia, restart = 100, score = 'bic')

all.equal(bn1_bic, bn2_bic)
graphviz.compare(bn1_bic, bn2_bic)

cat("network 1 bde score: ", bnlearn::score(bn1_bic, asia, type = 'bde'))
cat("\nnetwork 2 bde score; ", bnlearn::score(bn2_bic, asia, type = 'bde'))
# set.seed(12345)
bn1_bde <- hc(asia, restart = 100, score = 'bde')
bn2_bde <- hc(asia, restart = 100, score = 'bde')

all.equal(bn1_bde, bn2_bde)
graphviz.compare(bn1_bde, bn2_bde)
cat("network 1 bde score: ", bnlearn::score(bn1_bde, asia, type = 'bde'))
cat("\nnetwork 2 bde score; ", bnlearn::score(bn2_bde, asia, type = 'bde'))
bn1_iss <- hc(asia, restart = 100, iss = 1, score = 'bde')
bn2_iss <- hc(asia, restart = 100, iss = 2, score = 'bde')

all.equal(bn1_iss, bn2_iss)
graphviz.compare(bn1_iss, bn2_iss)
cat("network 1 bde score: ", score(bn1_iss, asia, type = 'bde'))
cat("\nnetwork 2 bde score; ", score(bn2_iss, asia, type = 'bde'))
set.seed(12345)
g1 = random.graph(colnames(asia))
# set.seed(12345)
g2 = random.graph(colnames(asia))

set.seed(12345)
hc1 = hc(asia, g1)
set.seed(12345)
hc2 = hc(asia, g2)

graphviz.compare(hc1, hc2)

all.equal(hc1, hc2)

cat("network 1 bde score: ", score(hc1, asia, type = 'bde'))
cat("\nnetwork 2 bde score; ", score(hc2, asia, type = 'bde'))

set.seed(12345)
g1 = random.graph(colnames(asia))
# set.seed(12345)
g2 = random.graph(colnames(asia))

set.seed(12345)
hc1 = hc(asia, g1, score = "bde")
set.seed(12345)
hc2 = hc(asia, g2, score = "bde")

graphviz.compare(hc1, hc2)

all.equal(hc1, hc2)

cat("network 1 bde score: ", score(hc1, asia, type = 'bde'))
cat("\nnetwork 2 bde score; ", score(hc2, asia, type = 'bde'))
# split the data
set.seed(12345)
n <- dim(asia)[1]
id <- sample(1:n, floor(n*0.8))
train <- asia[id, ]
test <- asia[-id, ]

# incremental assosciation of training data for PC
IAMB <- iamb(x = train)
#fit the model
model_fit <- bn.fit(IAMB, train)
# Into grain
bn_grain <- as.grain(model_fit)
#compile
bn_fin <- compile(bn_grain)

## predictions
# need a func (reuse for true network)
bn_func <- function(data, bn){
  pred <- rep(0, nrow(data))

  for(i in 1:nrow(data)){
    evid <- setEvidence(bn, nodes = colnames(data)[-2], states = data[i, -2])
    prob <- as.numeric(unlist(querygrain(evid, nodes = "S")))

    if(prob[1]>0.5){
      pred[i] <- "no"
    }else{
      pred[i] <- "yes"
    }
  }
  return(pred)
}

# setEvidence wants characters!!
```

```r
test_fix <- apply(test, 2, as.character)

predict_bn <- bn_func(data = test_fix, bn_fin)
confusion <- table(test[,"S"], predict_bn)
confusion

missclasification <- 1 - (sum(diag(confusion))/ sum(confusion))
missclasification
#true network
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#same operations as above
true_fit <- bn.fit(dag, train)
true_grain <- as.grain(true_fit)
true_bn <- compile(true_grain)

pred_true <- bn_func(data = test_fix, true_bn)


# confusion matrix and accuracy
confusion_true <- table(test[,"S"], pred_true)
confusion_true

missclasification_true <- 1 - (sum(diag(confusion_true))/ sum(confusion_true))
missclasification_true
# Markov blanket of S

mb <- mb(model_fit, "S")

mb_predict <- rep(0, nrow(test_fix))

for (i in 1:nrow(test_fix)){
  evidence <- setEvidence(bn_fin, nodes = mb, states = test_fix[i, mb])
  prob <- as.numeric(unlist(querygrain(evidence, nodes = "S")))

  if(prob[1]>0.5){
    mb_predict[i] <- "no"
  }else{
    mb_predict[i] <- "yes"
  }
}

confusion_mb <- table(test$S, mb_predict)
confusion_mb

missclass_mb <- 1 - (sum(diag(confusion_mb))/ sum(confusion_mb))
missclass_mb


# Naive bayes graph

naive_bayes = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(naive_bayes)
```

```
nbayes_fit <- bn.fit(naive_bayes, data = train)
nbayes_grain <- as.grain(nbayes_fit)
nbayes_compile <- compile(nbayes_grain)

nbayes_pred <- rep(0, nrow(test_fix))

for(i in 1:nrow(test_fix)){
  evidence <- setEvidence(nbayes_compile, nodes = c("A", "T", "L", "B", "E", "X", "D"), states = test_f
  prob <- unlist(querygrain(object = evidence, node = "S"))

  if(prob[1]>0.5){
    nbayes_pred[i] <- "no"
  }else{
    nbayes_pred[i] <- "yes"
  }
}

confusion_nbayes <- table(test$S, nbayes_pred)
confusion_nbayes


missclass_nbayes <- 1 - (sum(diag(confusion_nbayes))/ sum(confusion_nbayes))
missclass_nbayes
```