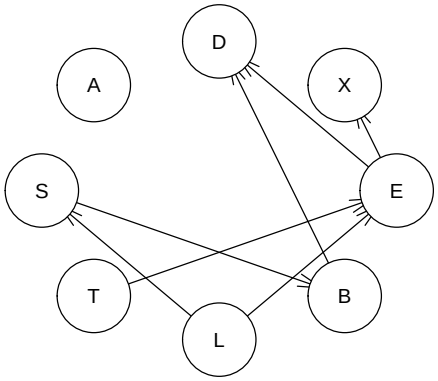
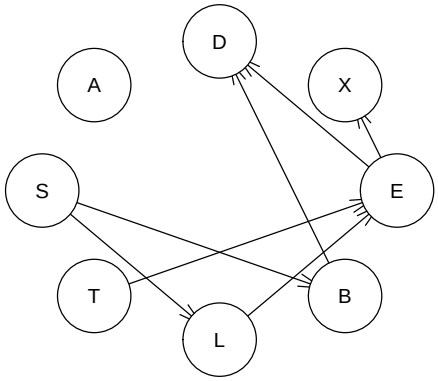
TDDE15 - Lab1 - Group

Task 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is included in the bnlearn package. To load the data, run data("asia").

```
library(bnlearn)

data('asia')

#First bayes network
structure1bayes=hc(asia)

#Iterating through bayes networks until I het two equal
for (i in 1:100){
  structurebayes=hc(asia, restart = 1)

  #Comparison of bayes networks
  boolean=all.equal(structure1bayes,structurebayes)

  if(boolean!=TRUE){
    break
  }
  structure1bayes=structurebayes

}

#Showing the difference between the networks
plot(structurebayes)
plot(structure1bayes)
```





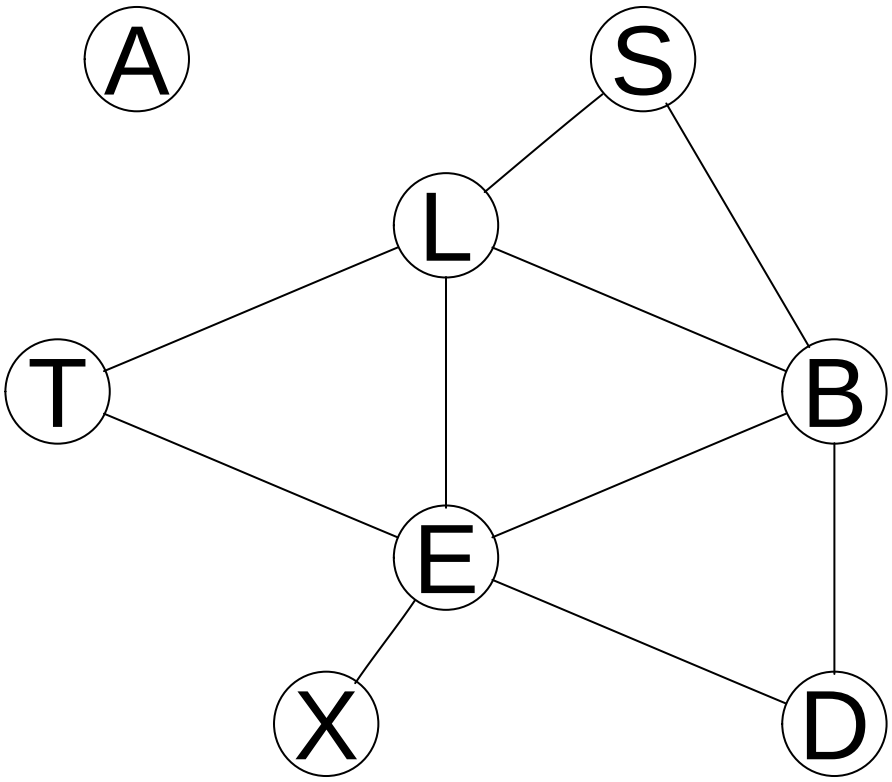
When doing the hill climbing algorithm the log score of the graph given the data is calculated for each possible added/removed or flipped arc. The "move" that improves the score the most is added but this doesn't necessarily lead to the global optima. The probability of reaching a global optima can be increased by adding more random restarts.

Task 2

Learn a BN from 80 % of the Asia dataset. The dataset is included in the bnlearn package. To load the data, run data("asia"). Learn both the structure and the parameters. Use any learning algorithm and settings that you consider appropriate. Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no. In other words, compute the posterior probability distribution of S for each case and classify it in the most likely class. To do so, you have to use exact or approximate inference with the help of the bnlearn and gRain packages, i.e. you are not allowed to use functions such as predict. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asia BN, which can be obtained by running dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]").

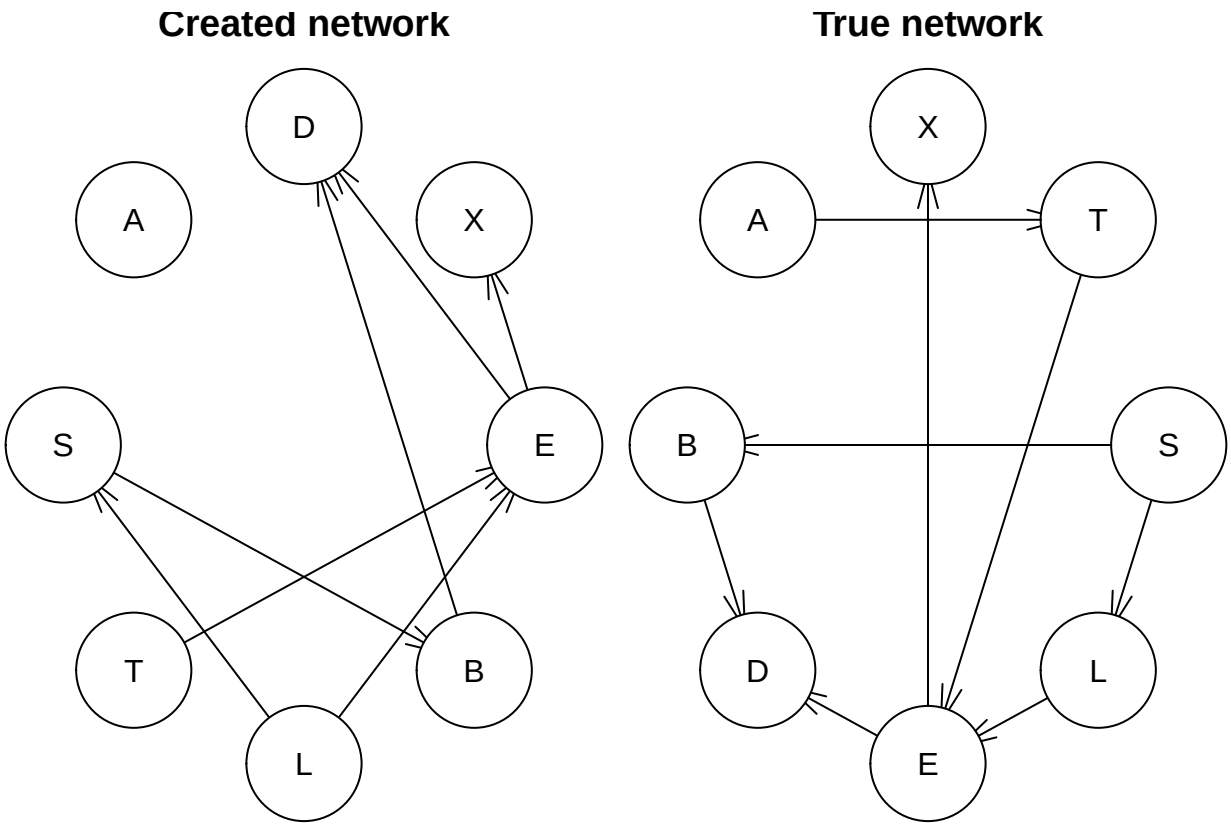
```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#split dataset into traning and test
n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))

train=asia[id,]
test=asia[-id,]

#install.packages("gRain")
library(gRain)

#create network from train
network = hc(train, start=NULL, restart = 10, optimized = FALSE)
#learn parameters for created network, based on same dataset,
```

```r
#get conditional probabiities in BN - needed for cliques
network_parameters = bn.fit(network, data = train)
#parameters for created network, see dependencies in matrices below
#coefficients(network_parameters)

#find potentials for cliques, checking all independencies
#Moralize, triangulaize - we get MN - Graphical independence network (grain)
graphical_Ind_network = as.grain(network_parameters)

#Create junction tree - R, seperators - needed to estimate potential cliques - information on how nodes
junc_tree = compile(graphical_Ind_network)
#We get our final Markov Network:
plot(junc_tree)
```




```r
#Classification for testdata!
pred_node=c("S")
answer = test[,"S"]
observed = test[,-2]
nodes = c("A", "T", "L", "B", "E", "X", "D")

missclass_rate = function(table){
  return(1-sum(diag(table))/sum(table))
}

predict_BN=function(network, nodes, pred_nodes, answer, observed){
```

```
  classify =c()
  nrows=nrow(observed)
  for (i in 1:nrows){
    #Sorry for fulkod
    obs = as.character(observed[i,])
    values=c()
    for (j in 1:length(obs)){
      if(obs[j]==1){
        values=c(values, "no")
      }else{
        values= c(values,"yes")
      }
    }

    #predict node S
    #Given observations - we update clique potentials
    evid = setEvidence(network, nodes = nodes, states = values)
    #Given new clique potential - we predict a specific node.
    pred = querygrain(evid, nodes=pred_node)

    if(pred$S[1] > 0.5){ #[1] = predict is NO
      classify = c(classify, "no")
    } else {
      classify = c(classify, "yes")
    }
  }

  #Get confusion matrix
  length(answer)
  length(classify)
  conf_matrix=table(answer, classify)

  return(confusionmatrix=conf_matrix)
}

par(mfrow=c(1,2))
plot(network, main="Created network")
plot(dag, main="True network")
```




```
conf_matrix=predict_BN(junc_tree,nodes,pred_node, answer, observed)
#Confusionmatrix for the created network:
print(conf_matrix)
```

```
##        classify
## answer  no yes
##    no  337 176
##    yes 121 366
```

```
#Misclassificationrate for the created network:
missclass_rate(conf_matrix)
```

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

```
#compare results with given dag
dag_parameters = bn.fit(dag, data = train)
graphical_Ind_dag = as.grain(dag_parameters)
dag_tree = compile(graphical_Ind_dag)


conf_matrix_true=predict_BN(dag_tree,nodes,pred_node, answer, observed) #We get the same results!

#Confusionmatrix network for the true network:
print(conf_matrix_true)
```

```
##       classify
## answer  no yes
##    no  337 176
##    yes 121 366
```

```
#Misclassificationrate for the true network:
missclass_rate(conf_matrix_true)
```

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

We get the same results (confusionmatrix and masclassification rate). This is bc S is independenten given markov blanket. Comparing the networks, S has the same markov blanket for both networks. Since the markov blanket is "the seperators" from the rest of the network, it only dependes on those specific nodes. This gives us the same results, no matter what network we use.

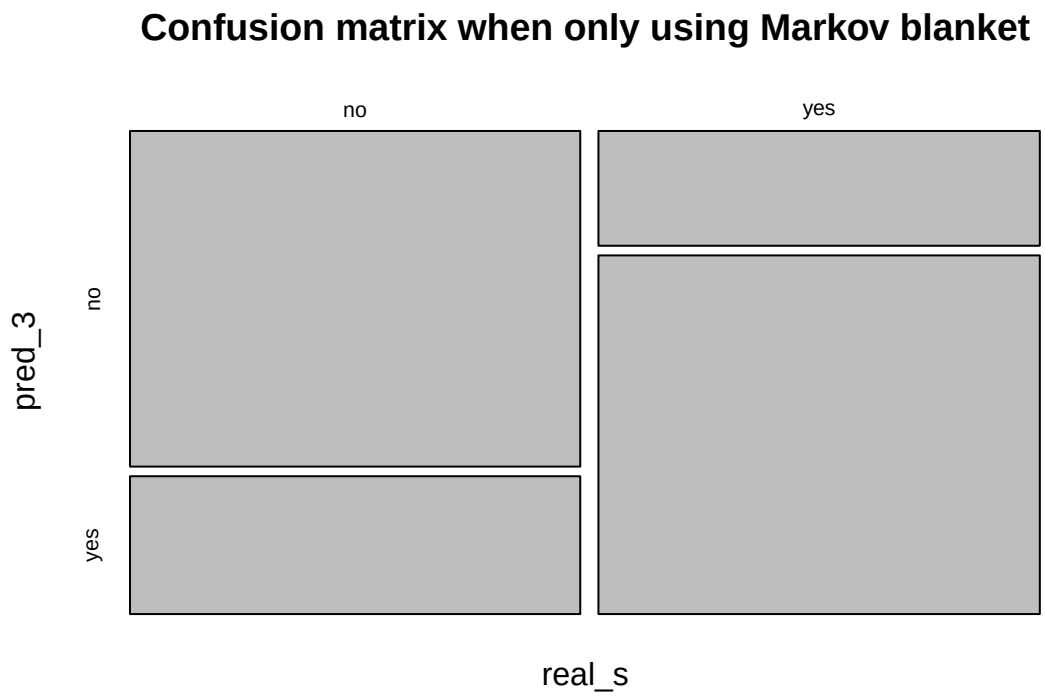
Question 3

In the previous exercise, you classified the variable S given observations for all the rest of the variables. Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.

```
classify = function(parameters, classify_data, info){
  # Cleaning data
  clean_data = c()
  for (i in 1:length(classify_data)){
    clean_data = c(clean_data, as.character(classify_data[1,i]));
  }

  grain = as.grain(parameters)
  structure = compile(grain) # creating junction tree, separators & residuals. Potentials for the cliqu

  goal = c("S")
  evi = setEvidence(structure, nodes = info, states = clean_data)
  dist = querygrain(evi, nodes = goal)
  if (dist$S[1] < 0.5){
    return("yes")
  }else{
    return("no")
  }
}

train = asia[(1:floor(0.8*length(asia[,1]))),]
test = asia[(floor(0.8*length(asia[,1]))+1):length(asia[,1])),]
real_s = test$S
new_parameters = bn.fit(network, train) #Network from question2.

mb = mb(network, node = c("S" ))
pred_3 = c()
for (i in 1:nrow(test)){
  pred_3 = c(pred_3, classify(new_parameters, test[i,][4:5], info = mb))
}
conf_matrix_3 = table(real_s, pred_3)
print(conf_matrix_3)
```

```
##       pred_3
## real_s  no yes
##    no  358 147
##    yes 120 375
```

```
plot(conf_matrix_3, main = "Confusion matrix when only using Markov blanket")
```




```
acc_3 = (conf_matrix_3[1]+conf_matrix_3[4])/sum(conf_matrix_3)
```

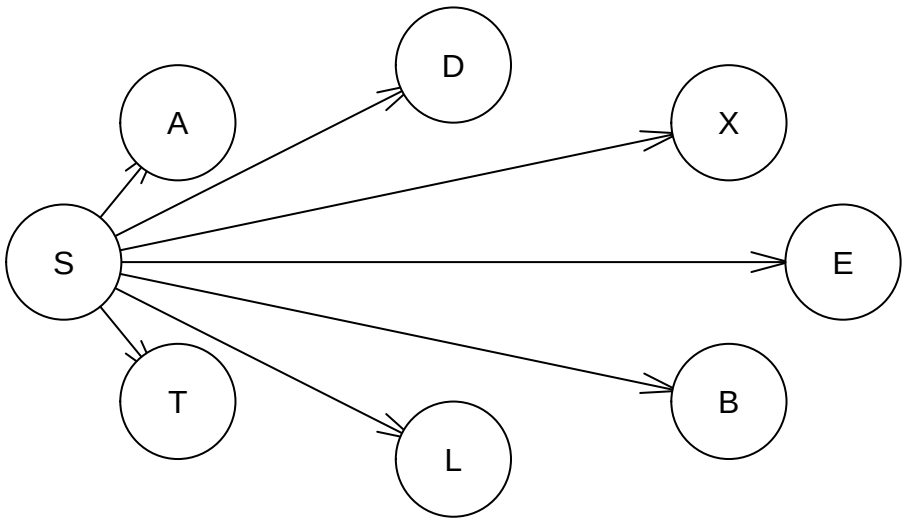
Here, information from node B and L is used. The result is just like the previous ones, meaning that we have don't lose any information by only using the Markov blanket when doing inference from the graph.

Question 4

Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You have to create the BN by hand, i.e. you are not allowed to use the function naive.bayes from the bnlearn package.

```
miss_class = function(conf_matr){
  return(1-sum(diag(conf_matr))/sum(conf_matr))
}

N = dim(asia)[1]
```

```
train = asia[1:floor(N*0.8),]
test = asia[(floor(N*0.8)+1):N,]
test_ans = test,"S"]
test_evid = subset(test, select = -2)

#Crating an empty network
library(bnlearn)
b_net = empty.graph(names(asia))

# Adjacncy matrix (0L ensures that the number is stored as an integer instead of a double)
adj = matrix(0L, ncol = 8, nrow = 8,
             dimnames = list(names(asia), names(asia)))
amat(b_net) = adj

# Add edges in BN
for (i in names(test_evid)) {
  adj["S",i] = 1L
}
amat(b_net) = adj
plot(b_net)
```




```
bn_pot = bn.fit(b_net, data=train)
bn_grain = as.grain(bn_pot)

pred_s =c()
```

```
for (j in 1:dim(test_evid[1])) {
# finding/evidance or potentials
  obs = c()
  for (i in 1:dim(test_evid)[2]) {
    obs = c(obs, as.character(test_evid[j,i]))
  }

  nodes_ev = names(test_evid)
  evid = setEvidence(bn_grain, nodes_ev, states = obs)

  # quergrain to get conditional distributon
  node = c("S")
  prob_s = querygrain(evid, nodes = node)

  if (prob_s$S[1]>prob_s$S[2]) {
    pred_s=c(pred_s,"no")
  }else{
    pred_s=c(pred_s,"yes")
  }
}
naive_table = table(pred_s,test_ans)

conf_matrix
```

```
##       classify
## answer  no yes
##    no  337 176
##    yes 121 366
```

```
miss_class(conf_matrix)
```

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

```
naive_table
```

```
##       test_ans
## pred_s  no yes
##    no  389 180
##    yes 116 315
```

```
miss_class(naive_table)
```

```
## [1] 0.296
```

The classification of the naive bayes classifer is predicting worse, with a higher missclassification rate, than the BN generated form the HC-algorithm. That is becaues Naive Bayes assumes that all varibles is independend given “S” and that there is a possible dependence between “S” and all other varibles. And in the correct DAG we saw that S is only dependent on “B” and “L”.

Task 5 - Explain why you obtain the same or different results in the exercises (2-4).

In exercise 2-3 we obtain the same results, this is due to all networks having the same Markov blanket, which are the nodes S depend on. In Exercise 5 we have a different markov blanket, with a naive assumption of independence between nodes in the network (except for S), which gives us a higher misclassification rate (and less accurate predictions).