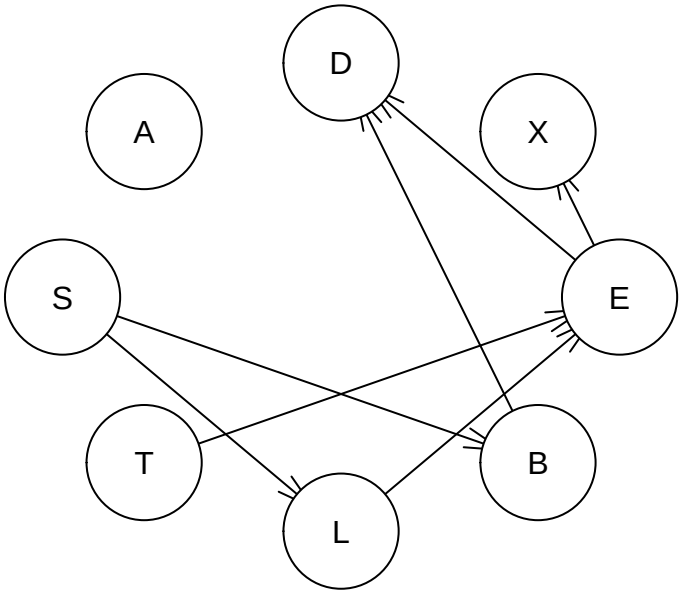
# Lab 1

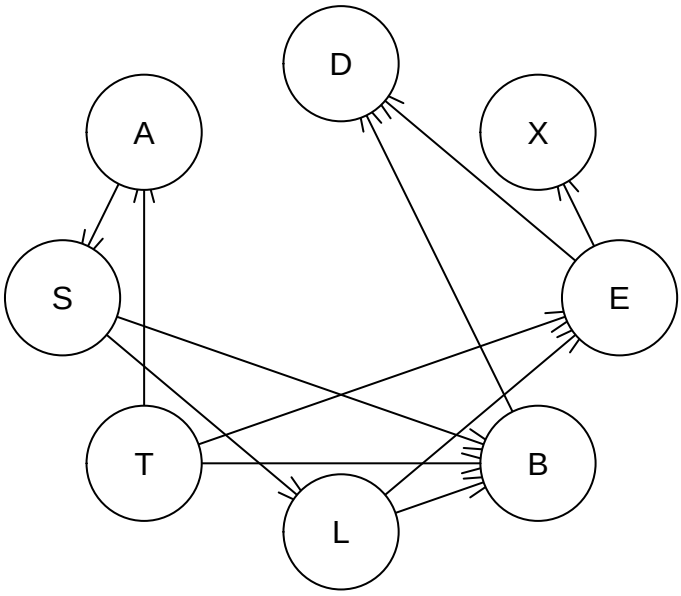
*Kristian Sikiric (krisi211)*

## Assignment 1

In this assignment we were to show that several runs of the hill climbing algorithm for structure learning can result in different bayesian networks. This was achived by using the hc function in the package bnlearn, the first network was trained with just the default setting, the second one was trained with 10 random restarts and the 'aic' score. This resultet in the following two networks.

**No random restarts, bic score**





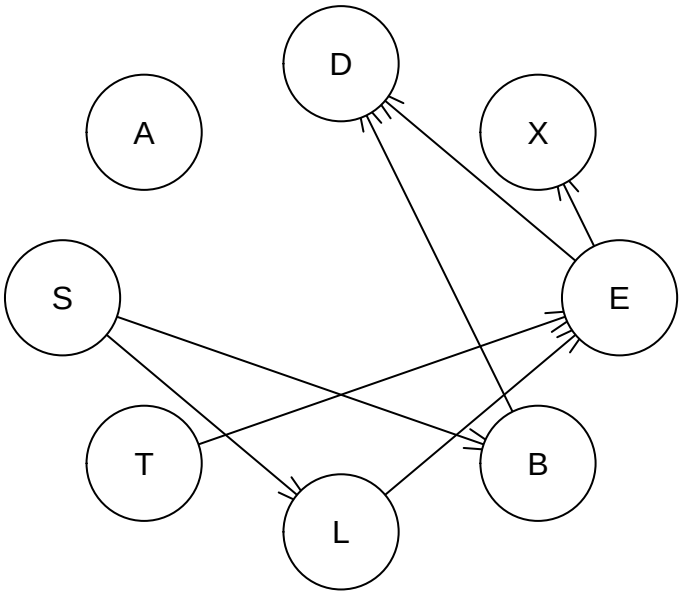

```
## [1] "Different number of directed/undirected arcs"
```

And as we can see, the two networks are not equivalent since the have different arc sets, the arc from node S to L does not have the same direction in both graphs. This is because the hill climbing algorithm cannot guarantee optimality, it can get stuck in a local maxima, and this can yield different networks if it get stuck in a different local maxima after a restart for example. The different scoreing options might give different local maxima as well. So a local maxima with the 'bic' score might is different from the local maxima with the 'aic' score.

# Assignment 2

In this assignment we used 80% of the data to learn the structre and the parameters of the bayesian network, the learned network was then used to classify the variable 'S' with the remaining 20%. In other words, we were to compute the posterior probability of 'S' given all the other variables. The results were then compared with the true bayesian network provided in the lab. The resulting confusion matrices can be seen below, were 'prediction' is the predictions for the learned network and 'true.prediction' are the predictions from the true network.

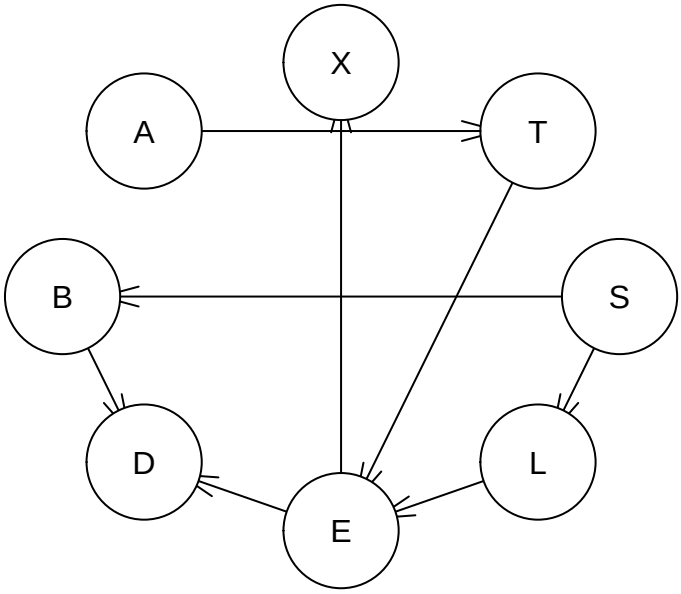
## Learned network




```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] "Classification rate:  0.724"
```

## True network




```
##           Actual
## Prediction  no yes
##        no  341 119
```

```
##        yes 157 383
```

```
## [1] "Classification rate:  0.724"
```

As we see the two networks give the same confusion matrix when classifying the node 'S', with a classification rate of 72,4%. This is because if we look at the two networks, we see that they are almost the same, just the arc from node 'A' to 'T' that is missing from our learned network.

As it turnes out, the arc from 'A' to 'T' only effects the value of 'T' and has nothing to do with 'S'. 'S' is only affected by the values of 'B' and 'L' since all values are already observed.

# Assignment 3

In this assignment we were to redo assignment 2, but instead of using all variables but 'S' as evidence in our classification, we were now only to use the markov blanket. It turns out that the markov blanket concists of nodes 'B' and 'L', this gives exactly the same results as in assignment 2 since we concluded that 'S' is only affected on 'B' and 'L' in our networks.

Confusion matrix for the learned network when using the markov blanket:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] 0.724
```

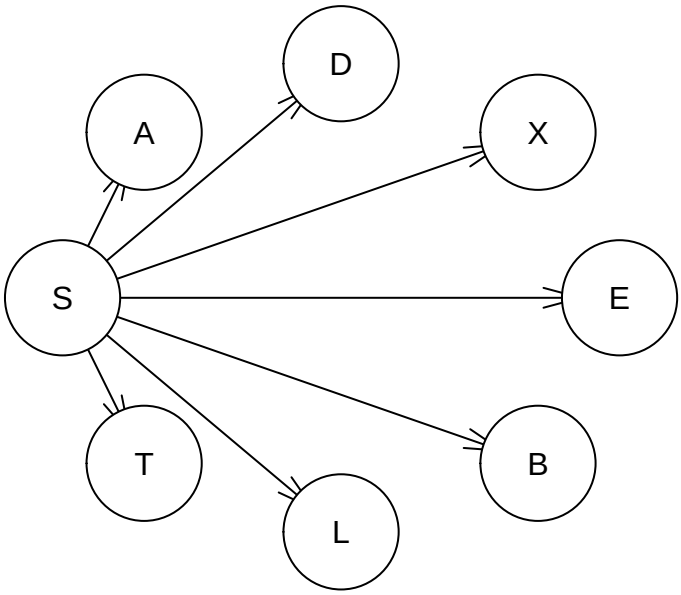
Confusion matrix for the true network when using the markov blanket:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] 0.724
```

# Assignment 4

In this assignment we were to use the naive bayes classifier, i.e the predictive variable is independent given the class variables. The naive bayes classifier was modeled as a bayesian network and this network was the classified in the same way as assignment 2. The network and the confusion matrix of the naive bayes network is shown below, as well as the confusion matrix of the ture network from assignment 2.




```
##           Actual
## Prediction  no yes
##        no  374 181
##        yes 124 321
```

```
## [1] 0.695
```

Confusion matrix for the true network from assignment 2:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] 0.724
```

We no longer get the same confusion matrix for the two predictions, because 'S' is now dependent on all other variables and not just the markov blanket. The classification rate is now 69,5%.

# Assignment 5

In assginment 2 and 3 we got the same results, in assigment 3, we were to classify using only the markov blanket and in assigment 2 the node 'S' is only dependent on the markov blanket. This is why we get the same results in assigments 2 and 3. The rest of the nodes does not matter in assigment 2. In assigment 4 on the other hand, 'S' is now dependent on all other nodes, not just the markov blanket, giving this network different classifications. We also see that we get a bit worse classification rate for the naive bayes classification, this is reasonable since this classifier asumes independence when this is not the case.

## Appendix - Code

```
###### Init ######
library(bnlearn)
library(gRain)
data('asia')

###### Functions ######
predict.bn = function(junction.tree,data,target_nodes, evidence_nodes)
{
  predictions = rep(0,dim(data)[1])
  for (i in 1:dim(data)[1]) {
    states = NULL
    for (node in evidence_nodes) {
      states[node] = ifelse(data[i,node]=="yes","yes","no")
    }
    evidence = setEvidence(junction.tree,
                           nodes = evidence_nodes,
                           states = states)
    prob = querygrain(evidence,
                      nodes = target_nodes)
    predictions[i] = ifelse(prob$S["yes"] >= prob$S["no"],"yes","no")
  }
  return(predictions)
}

###### Exercise 1 ######
set.seed(1234)
bn = hc(x = asia)
bn_with_restart = hc(x = asia, restart=10, score = 'aic')
plot(bn)
plot(bn_with_restart)
all.equal(bn,bn_with_restart)


###### Exercise 2 ######
smp_size = dim(asia)[1]
set.seed(123)
id = sample(1:smp_size,floor(smp_size*0.8))
train = asia[id,]
test = asia[-id,]

bn = hc(x=train)
fitted.bn = bn.fit(bn,data=train)
junction.tree = compile(as.grain(fitted.bn)) #RIP

true.bn = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true.fitted.bn = bn.fit(true.bn,data=train)
true.junction.tree=compile(as.grain(true.fitted.bn))

nodes = colnames(asia)
evidence_nodes = nodes[nodes != "S"]
target_nodes = nodes[nodes == "S"]
```

```
prediction = predict.bn(junction.tree = junction.tree,
                        data = test,
                        target_nodes = target_nodes,
                        evidence_nodes = evidence_nodes)
true.prediction = predict.bn(junction.tree =  true.junction.tree,
                             data = test,
                             target_nodes = target_nodes,
                             evidence_nodes = evidence_nodes)
print(table(prediction,test$S))
print(table(true.prediction,test$S))

###### Exercise 3 ######
markov.blanket = mb(fitted.bn,"S")
true.markov.blanket = mb(true.fitted.bn,"S")
prediction.mb = predict.bn(junction.tree = junction.tree,
                      data = test,
                      target_nodes = target_nodes,
                      evidence_nodes = markov.blanket)
true.prediction.mb = predict.bn(junction.tree = true.junction.tree,
                      data = test,
                      target_nodes = target_nodes,
                      evidence_nodes = true.markov.blanket)
print(table(prediction.mb,test$S))
print(table(true.prediction.mb,test$S))

###### Exercise 4 ######
naive.bayes.dag = empty.graph(nodes = nodes)
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "A")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "D")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "X")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "E")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "B")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "L")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "T")
plot(naive.bayes.dag)
nbd.fit = bn.fit(naive.bayes.dag,data=train)
junc.tree = compile(as.grain(nbd.fit))

predict.nbd = predict.bn(junc.tree, test, target_nodes, evidence_nodes)
print(table(predict.nbd,test$S))
print(table(true.prediction,test$S))
```