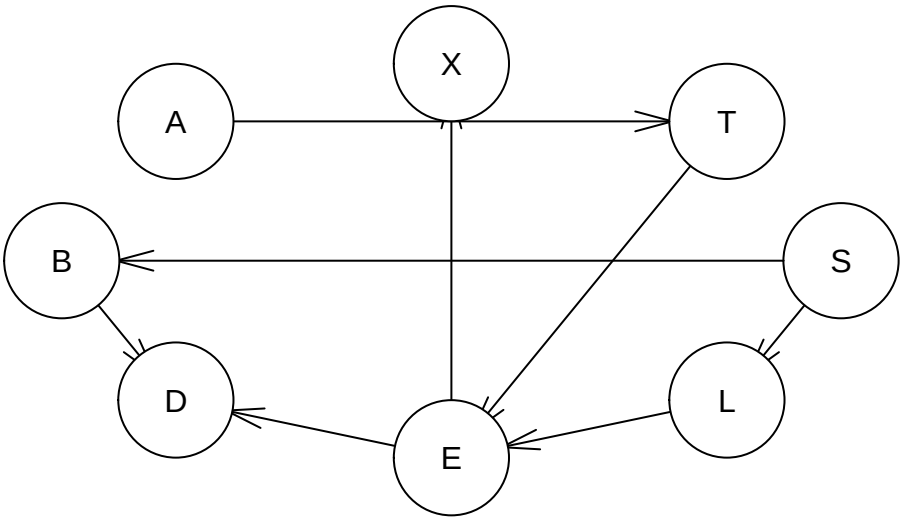
Lab1

Christian von Koch, William Anzen, Josefin Bladh

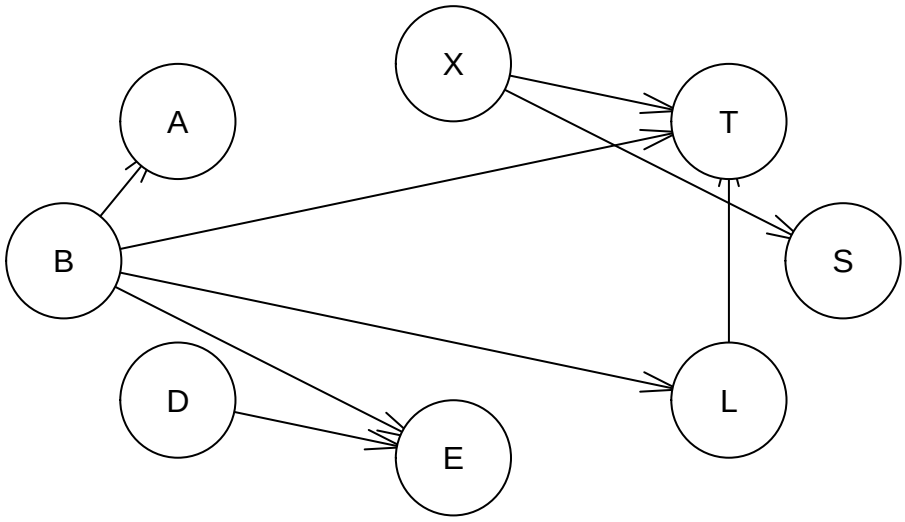
2020-09-11

Assignment 1 - Christian von Koch

```
data("asia")
init_dag1 = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
init_dag2 = model2network("[B][D][X][E|D:B][L|B][T|L:B:X][S|X][A|B]")
plot(init_dag1)
```




```
plot(init_dag2)
```



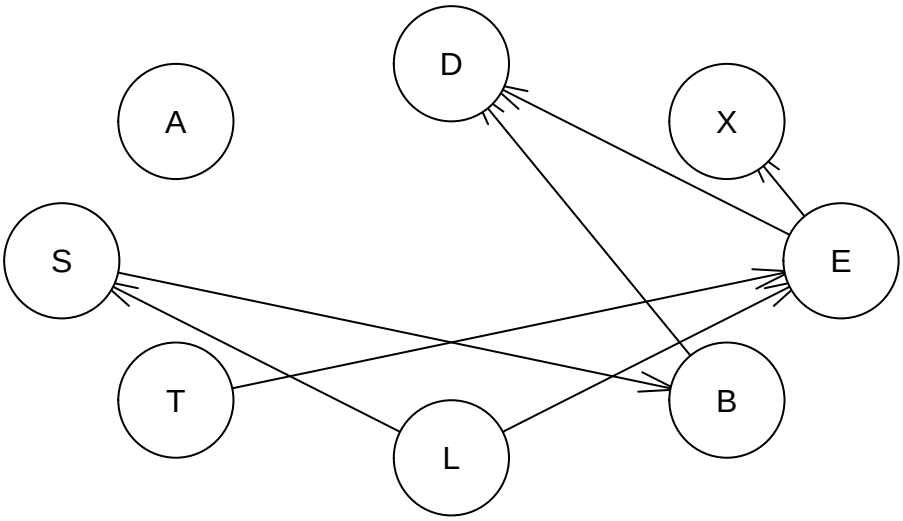

```
random_restarts=c(1,10,1000)
scores=c("aic", "bic", "bde")
iss=c(1,10,100)

# Testing different initial structures
model1_init=hc(asia, start=init_dag1)
model2_init=hc(asia, start=init_dag2)
all.equal(model1_init, model2_init)
```

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

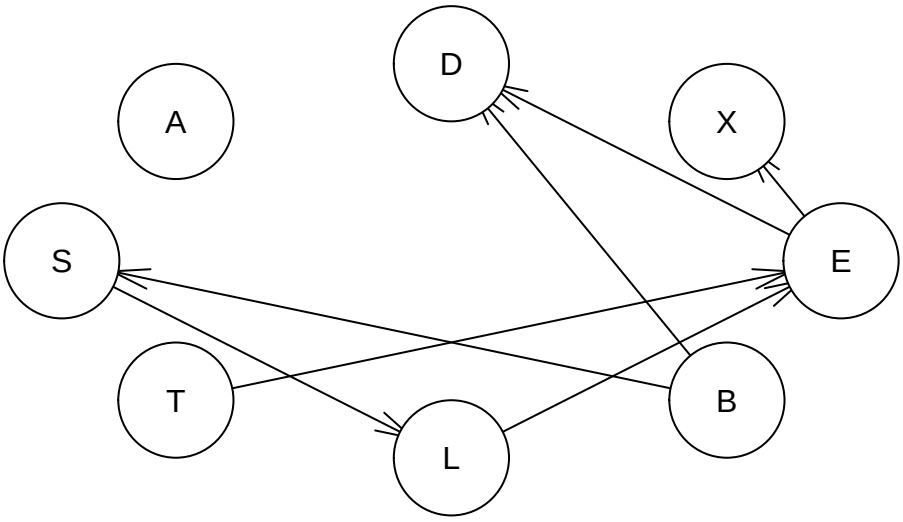
```
# Changed the resulting network
```

It is clear that different initial structures inputted to the algorithm can result in different non-equivalent Bayesian Networks outputted by the Hill Climbing Algorithm. This is due to the fact that the hill climbing algorithm can terminate at different *local* optima points, i.e. the algorithm is not guaranteed to find the *global* optimum. By choosing different starting points for the algorithm it is obvious that the algorithm might end up in different *local* optima outputs.

```
# Testing different random restarts
model1_random=hc(asia, restart=random_restarts[1])
model2_random=hc(asia, restart=random_restarts[2])
model3_random=hc(asia, restart=random_restarts[3])
plot(model3_random)
```




```
plot(model2_random)
```




```
all.equal(model1_random, model2_random)
```

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

```
all.equal(model1_random, model3_random)
```

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

```
all.equal(model2_random, model3_random)
```

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

```
# Changed the resulting network
```

By choosing different values of random restarts we tell the algorithm to do the algorithm a specified number of times since the different paths which the algorithm can choose for each step can be many and are chosen randomely. This is due to the fact that many different proposed networks from a certain point in the algorithm can result in the same overall score for that specific network after which the algorithm chooses a structure randomely from the set of structures with equal (and highest) scores. By defining the number of random restarts the algorithm might end up in different *local* optima, i.e. different networks with different scores after which the algorithm outputs the network with the highest score. As seen above the network modelled with 1000 random restarts was different in comparison to the models which only used 1 and 10 restarts respectively. This particular model also had a *higher* score than the other ones which also can be confirmed by comparing to the true network.

```
# Testing different score models
model1_score=hc(asia, score=scores[1])
model2_score=hc(asia, score=scores[2])
model3_score=hc(asia, score=scores[3])
```

```
all.equal(model1_score, model2_score)
```

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

```
all.equal(model1_score, model3_score)
```

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

```
all.equal(model2_score, model3_score)
```

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

```
# Fundamentally changed the resulting network
```

By choosing different score models we see from the output above that the HC algorithm might output different non-equivalent BNs. This is because by choosing different score models the networks are evaluated differently for each step, which evidently can result in the algorithm to be terminated at different *local* optimas, i.e. output different non-equivalent Bayesian Networks.

```
# Testing different imaginary sample sizes
model1_iss=hc(asia, score="bde", iss=iss[1])
model2_iss=hc(asia, score="bde", iss=iss[2])
model3_iss=hc(asia, score="bde", iss=iss[3])
all.equal(model1_iss, model2_iss)
```

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

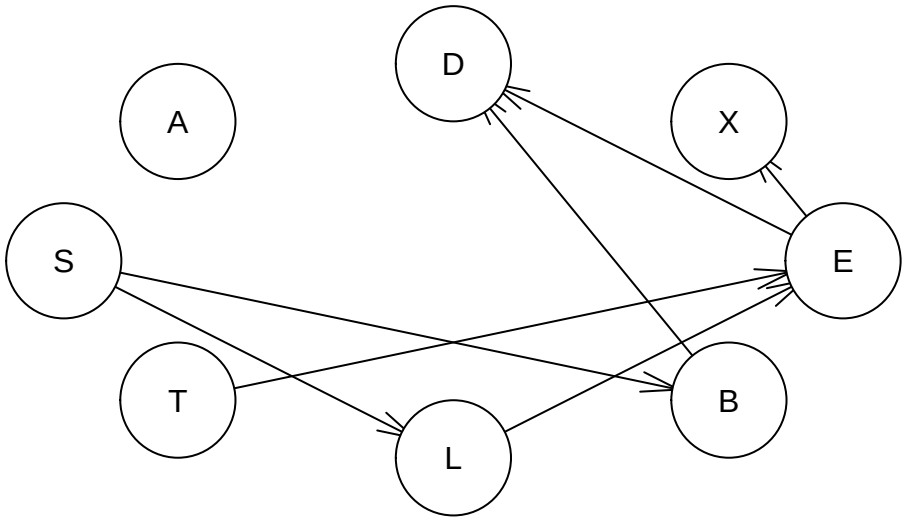
```
all.equal(model1_iss, model3_iss)
```

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

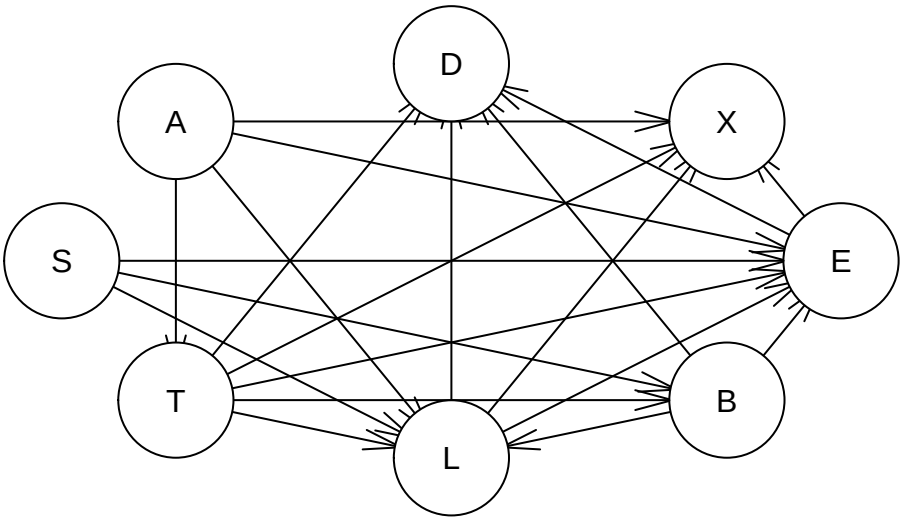
```
all.equal(model2_iss, model3_iss)
```

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

```
plot(model1_iss)
```




```
plot(model3_iss)
```

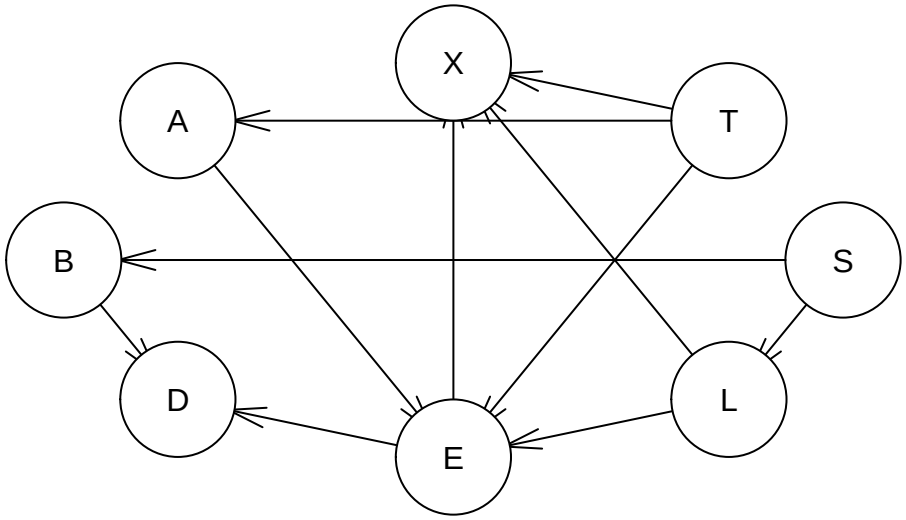


It is evident from the above result that different sizes of imaginary sample size also might result in different outputted BNs by the HC algorithm. This is due to the fact that the imaginary sample size represent how much regularization that should be applied when running the algorithm. By specifying a high imaginary sample size the model outputted are more complex and contains more parameters which might result in an overfitted model. A low imaginary sample size on the other hand corresponds to high regularization applied to the model which results in a more sparse model outputted by the algorithm. Above, this is shown by the outputs when using imaginary sample size 1 and 100 where it is evident that the second output which used ISS of 100 is much more complex and contains more edges (which means more parameters) in comparison to the more sparse model shown in the first output.

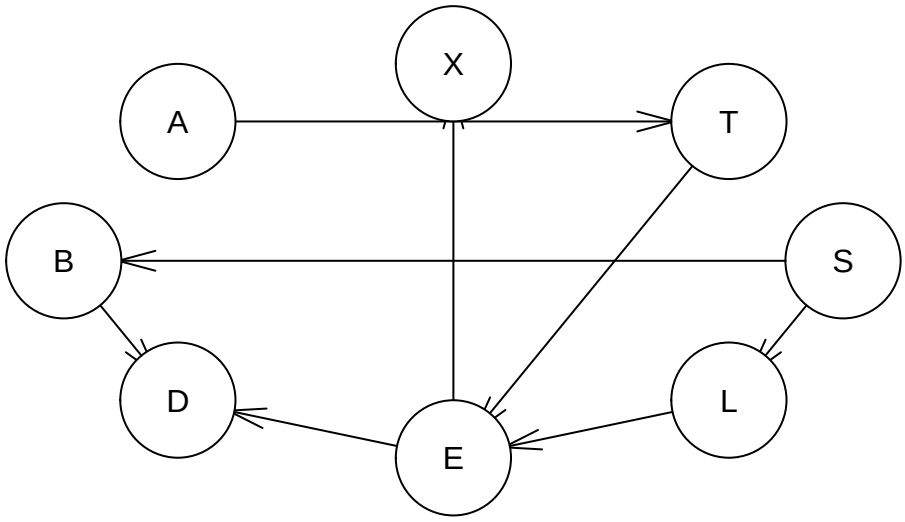
Assignment 2 - William Anzen

```
predictNet <- function(juncTree, data, features, target){
  predArray <- matrix(nrow=nrow(data),ncol=1)
  for(i in 1:nrow(data)){
    obsStates <- NULL
    for(p in features){
      if(data[i,p]=="yes"){
        obsStates <- c(obsStates,"yes")
      } else{
        obsStates <- c(obsStates,"no")
      }
    }
```

```
    obsEvidence <- setEvidence(object = juncTree,
                               nodes = features,
                               states = obsStates)
    obsPredProb <- querygrain(object = obsEvidence,
                              nodes = target)$S
    predArray[i] <- if(obsPredProb["yes"]>=0.5) "yes" else "no"
  }
  return(predArray)
}

n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
hcBNTest <- hc(train, start = model2network("[B][T][A|T][S|A][L|A][D|B:L][E|T:L][X|E]"), score="bde",
iss=5)
BNTrue <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(hcBNTest)
```




```
plot(BNTrue)
```




```
fittedTest <- bn.fit(hcBNTest, train) #Fitting parameters in structure with traindata
fittedTrue <- bn.fit(BNTrue, train)
BNGrainTest = as.grain(fittedTest) #Graphical independence network
```

```
## Warning in as.grain.bn.fit(fittedTest): NaN conditional probabilities in E,
## replaced with a uniform distribution.
```

```
## Warning in as.grain.bn.fit(fittedTest): NaN conditional probabilities in X,
## replaced with a uniform distribution.
```

```
BNGrainTrue = as.grain(fittedTrue)
junctionTreeTest <- compile(BNGrainTest) #Get cliques with potentials (Lauritzen-Spiegelhalter algorith
junctionTreeTrue <- compile(BNGrainTrue)

obsVars=c("A", "T", "L", "B", "E", "X", "D")
tarVar=c("S")
predictionTest <- predictNet(junctionTreeTest, test, obsVars, tarVar)
predictionTrue <- predictNet(junctionTreeTrue, test, obsVars, tarVar)
confusionTest <- table(predictionTest, test$S)
confusionTest
```

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

```
confusionTrue <- table(predictionTrue, test$S)
confusionTrue
```

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

```
#Same confusion tables
```

The results were no difference in the predictions. Markov Blanket (MB) consists of the nodes of importance for the target nodes' dependencies in the network. If we do not change the MB, the outcome of the prediction on the target node will not change. Since the MB does not change from real DAG and modeled DAG, the confusion matrices does not change between these networks.

Assignment 3 - Josefin Bladh

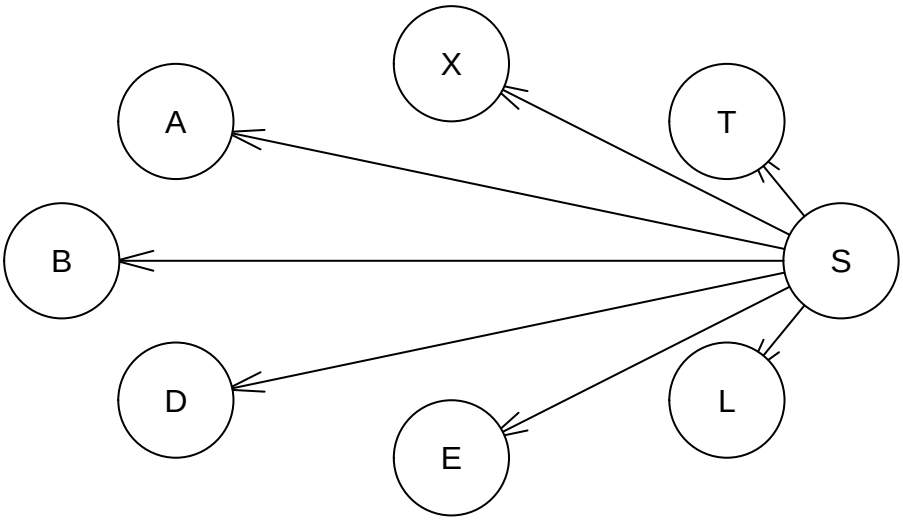
```
obs2=mb(fittedTrue,"S")
S2=predictNet(junctionTreeTest, test, obs2, tarVar)
table(S2,test$S)
```

```
##
## S2     no yes
##   no  337 121
##   yes 176 366
```

The reason we get the same results in the confusion matrix when only using only the Markov blankets as observations are that the dependences does not change for S. It is only the Markov blankets that will affect the conditional probabilities for the target, S. Meaning the only dependent variables in this case is B and L and both BNs have the same structure in this regard.

Assignment 4 - Josefin Bladh

```
set.seed(12345)
bnNaive=model2network("[S][A|S][B|S][X|S][T|S][L|S][E|S][D|S]")
plot(bnNaive)
```




```
naiveFit=bn.fit(bnNaive,train)
grainNaive = as.grain(naiveFit) # Grapical independence network
junctionNaive= compile(grainNaive) # Get cliques with potential values.
Snaive=predictNet(junctionNaive, test, obsVars, tarVar)
table(Snaive,test$S)
```

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

In this task we can now see a different result in the confusion matrix. As we are using the naïve assumption that all features are independent given S, we can see a DAG where all other nodes are the children of S. Meaning all other nodes are the Markov blankets of S, due to the conditional probabilities.

Assignment 5 - Discussed in group

This has been explained in each of the exercises - the results vary depending on the resulting markov blanket of S from the model used.

Appendix

```
## Assignment 1

library(bnlearn)
library(Rgraphviz)
library(gRain)
library(RBGL)

data("asia")
init_dag1 = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
init_dag2 = model2network("[B][D][X][E|D:B][L|B][T|L:B:X][S|X][A|B]")
plot(init_dag1)
plot(init_dag2)
random_restarts=c(1,10,1000)
scores=c("aic", "bic", "bde")
iss=c(1,10,100)

# Testing different initial structures
model1_init=hc(asia, start=init_dag1)
model2_init=hc(asia, start=init_dag2)
all.equal(model1_init, model2_init)

# Changed the resulting network

# Testing different random restarts
model1_random=hc(asia, restart=random_restarts[1])
model2_random=hc(asia, restart=random_restarts[2])
model3_random=hc(asia, restart=random_restarts[3])
plot(model3_random)
plot(model2_random)
all.equal(model1_random, model2_random)
all.equal(model1_random, model3_random)
all.equal(model2_random, model3_random)

# Changed the resulting network

# Testing different score models
model1_score=hc(asia, score=scores[1])
model2_score=hc(asia, score=scores[2])
model3_score=hc(asia, score=scores[3])
all.equal(model1_score, model2_score)
all.equal(model1_score, model3_score)
all.equal(model2_score, model3_score)

# Fundamentally changed the resulting network

# Testing different imaginary sample sizes
model1_iss=hc(asia, score="bde", iss=iss[1])
model2_iss=hc(asia, score="bde", iss=iss[2])
model3_iss=hc(asia, score="bde", iss=iss[3])
all.equal(model1_iss, model2_iss)
all.equal(model1_iss, model3_iss)
all.equal(model2_iss, model3_iss)
```

```r
plot(model1_iss)
plot(model3_iss)

## Assignment 2

predictNet <- function(juncTree, data, features, target){
  predArray <- matrix(nrow=nrow(data),ncol=1)
  for(i in 1:nrow(data)){
    obsStates <- NULL
    for(p in features){
      if(data[i,p]=="yes"){
        obsStates <- c(obsStates,"yes")
      } else{
        obsStates <- c(obsStates,"no")
      }
    }


    obsEvidence <- setEvidence(object = juncTree,
                               nodes = features,
                               states = obsStates)
    obsPredProb <- querygrain(object = obsEvidence,
                              nodes = target)$S
    predArray[i] <- if(obsPredProb["yes"]>=0.5) "yes" else "no"
  }
  return(predArray)
}

n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
hcBNTest <- hc(train, start = model2network("[B][T][A|T][S|A][L|A][D|B:L][E|T:L][X|E]"), score="bde",
iss=5)
BNTrue <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(hcBNTest)
plot(BNTrue)
fittedTest <- bn.fit(hcBNTest, train) #Fitting parameters in structure with traindata
fittedTrue <- bn.fit(BNTrue, train)
BNGrainTest = as.grain(fittedTest) #Graphical independence network
BNGrainTrue = as.grain(fittedTrue)
junctionTreeTest <- compile(BNGrainTest) #Get cliques with potentials (Lauritzen-Spiegelhalter algorith
junctionTreeTrue <- compile(BNGrainTrue)

obsVars=c("A", "T", "L", "B", "E", "X", "D")
tarVar=c("S")
predictionTest <- predictNet(junctionTreeTest, test, obsVars, tarVar)
predictionTrue <- predictNet(junctionTreeTrue, test, obsVars, tarVar)
confusionTest <- table(predictionTest, test$S)
confusionTest
confusionTrue <- table(predictionTrue, test$S)
confusionTrue
```

```r
#Same confusion tables

## Assignment 3

obs2=mb(fittedTrue,"S")
S2=predictNet(junctionTreeTest, test, obs2, tarVar)
table(S2,test$S)

## Assignment 4

set.seed(12345)
bnNaive=model2network("[S][A|S][B|S][X|S][T|S][L|S][E|S][D|S]")
plot(bnNaive)
naiveFit=bn.fit(bnNaive,train)
grainNaive = as.grain(naiveFit) # Grapical independence network
junctionNaive= compile(grainNaive) # Get cliques with potential values.
Snaive=predictNet(junctionNaive, test, obsVars, tarVar)
table(Snaive,test$S)
```