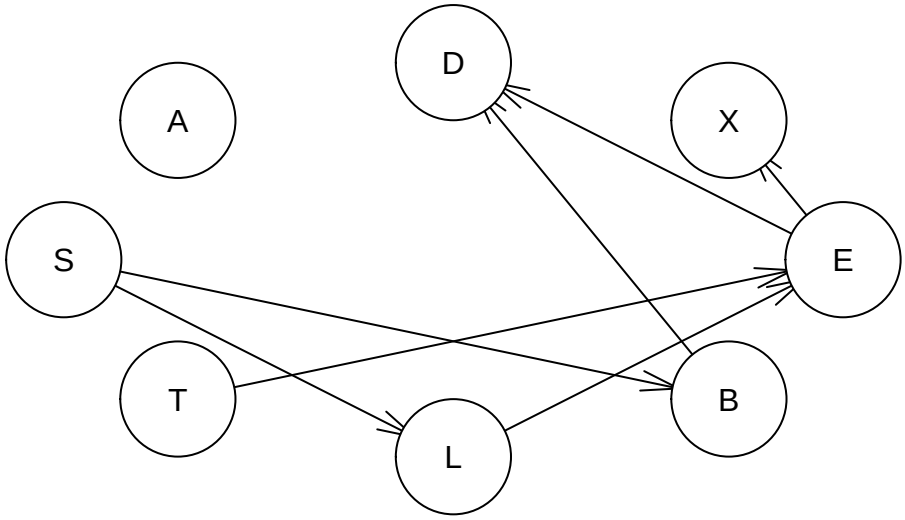
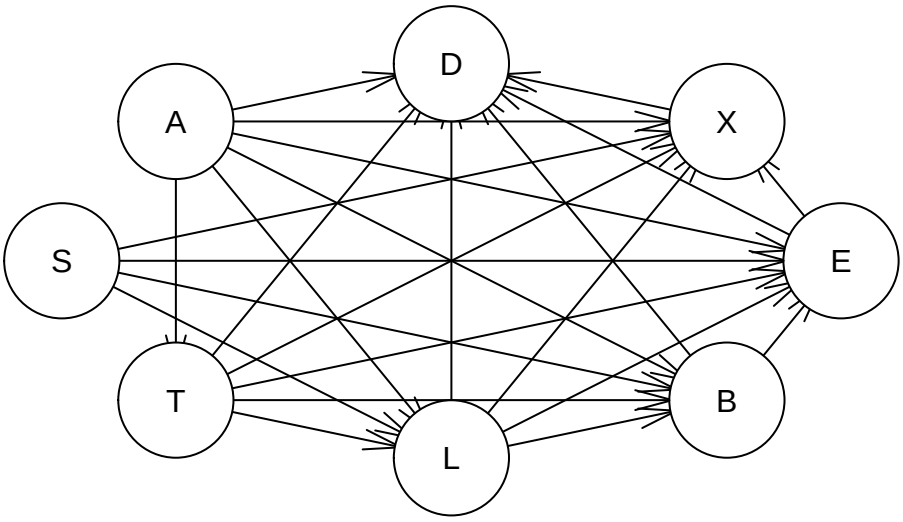
1

```
rm(list=ls())
library(bnlearn)
set.seed(12345)
data("asia")

bn1 = hc(asia, start = NULL, score = "bde", restart = 1, iss = 1)
# arcs(bn1)
# vstructs(bn1)
plot(bn1)
```




```
bn2 = hc(asia, start = NULL, score = "bde", restart = 1, iss = 1000)
# arcs(bn2)
# vstructs(bn2)
plot(bn2)
```




```
# Cpdag to see equivalent class to get.
cat("Different ISS gives non-equivalent classes: ", all.equal(cpdag(bn1), cpdag(bn2)))
```

```
## Different ISS gives non-equivalent classes:  Different number of directed/undirected arcs
```

Definition of equivalent BN structure: "[...] the same adjacencies and unshielded colliders" They are non-equivalent. For example, in the first BN A,D is not adjacent, which they are in the second. The reason for the second graph being way more complex is due to the iss term being set very high, reducing regularization. This can happen due to HC finding local optimas, not global.

```
start1 = random.graph(colnames(asia))
startbn1 = hc(asia, start = start1)
start2 = random.graph(colnames(asia))
startbn2 = hc(asia, start = start2)

cat("\nDifferent start gives non-equivalent classes: ", all.equal(cpdag(startbn1), cpdag(startbn2)))
```

```
##
## Different start gives non-equivalent classes:  Different number of directed/undirected arcs
```

2

```r
rm(list=ls())
library(bnlearn)
library(gRain)
```

```
## Loading required package: gRbase
```

```
##
## Attaching package: 'gRbase'
```

```
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, nodes, parents
```

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

n=dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train_data=asia[id,]
test_data=asia[-id,]

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

fit = bn.fit(x = bn, data = train_data)
true_fit = bn.fit(x = dag, data = train_data)

grain = as.grain(fit)
true_grain = as.grain(true_fit)

compile = compile(grain)
true_compile = compile(true_grain)

predict = function (network){
  predictions = c()
  for (i in 1:nrow(test_data)) {
    evidence = setEvidence(network, colnames(test_data)[-2], as.vector(as.matrix(test_data[i, -2])))
    query = querygrain(evidence, colnames(test_data)[2])$S

    predictions = c(predictions, ifelse(query["yes"]>query["no"], "yes", "no"))
  }
  return(predictions)
}

table(predict(compile), test_data$S)
```

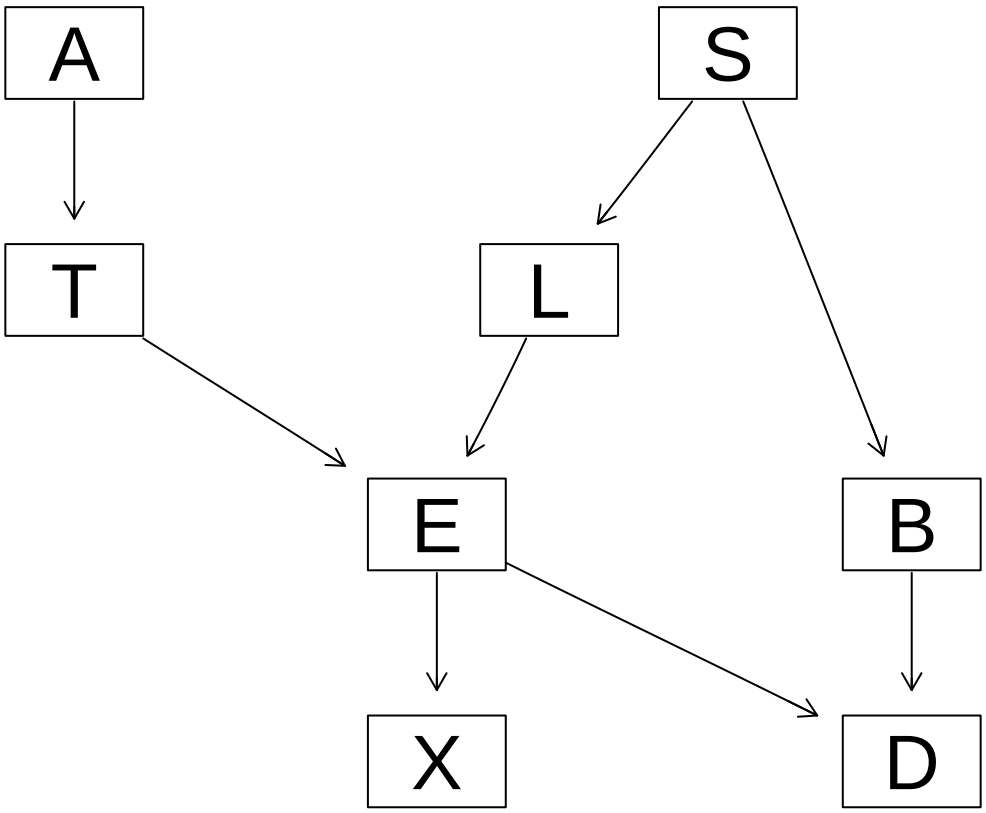
```
##
##        no yes
##   no  337 121
##   yes 176 366
```

```
table(predict(true_compile), test_data$S)
```

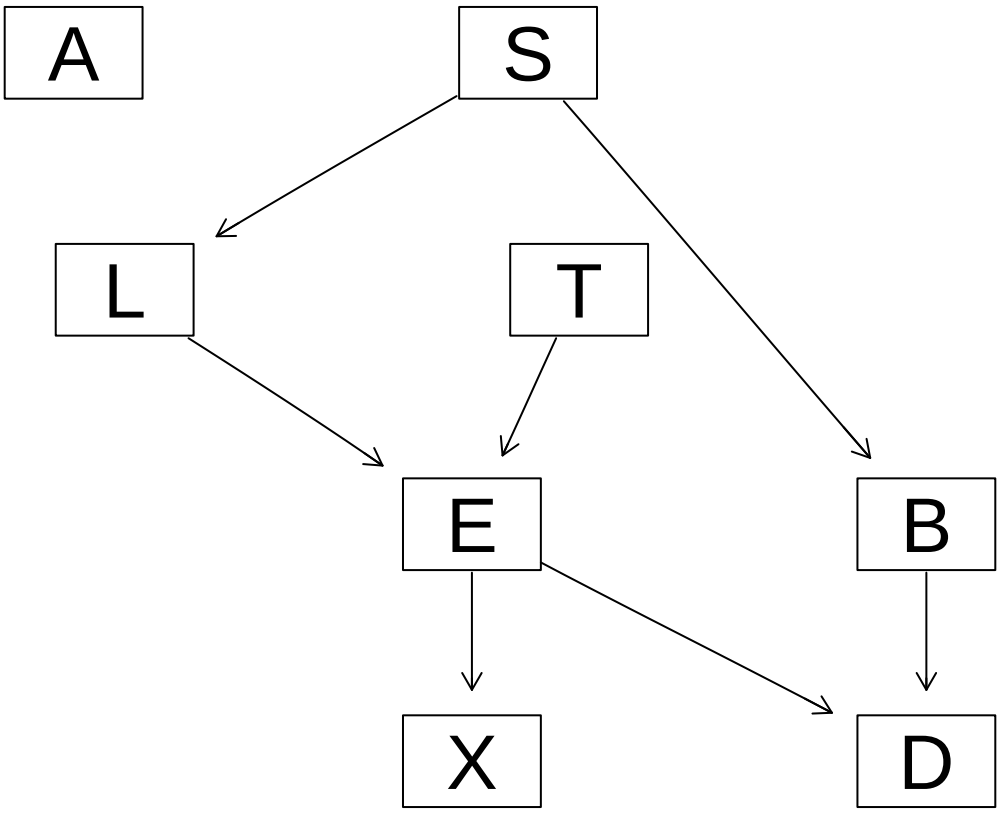
```
##
##        no yes
##   no  337 121
##   yes 176 366
```

```
graphviz.plot(dag)
```

```
## Loading required namespace: Rgraphviz
```




```
graphviz.plot(bn)
```



The confusion matrices are very similar. Only difference is edge A->T in the true graph.

3

"Find the minimal set of nodes that separates a given node from the rest. This set is called the Markov blanket of the given node."

```
mb = mb(fit, c("S"))
true_mb = mb(true_fit, c("S"))

predict_mb = function (network, m_b){
  predictions = c()
  for (i in 1:nrow(test_data)) {
    evidence = setEvidence(network, m_b, as.vector(as.matrix(test_data[i, m_b])))
    query = querygrain(evidence, colnames(test_data)[2])$S

    predictions = c(predictions, ifelse(query["yes"]>query["no"], "yes", "no"))
  }
  return(predictions)
}

table(predict_mb(compile, mb), test_data$S)
```

```
##
##        no yes
```

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

```
table(predict_mb(true_compile, true_mb), test_data$S)
```

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

The confusion matrices are once again similar. Even identical now.

4

```
# Create the Naive Bayes structure manually
nb_network = model2network("[S][A|S][B|S][D|S][E|S][L|S][T|S][X|S]")

# Learn the parameters from the training data
nb_fit <- bn.fit(nb_network, data = train_data)

# Convert to gRain object and compile it for inference
nb_grain <- compile(as.grain(nb_fit))

# Use the same prediction function as in Question 2
table(predict(nb_grain), test_data$S)
```

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

```
table(predict(true_compile), test_data$S)
```

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

The confusion matrices now differ.

5

In question 3 we get a similar result as to question 2, because when you know the states of the nodes in the Markov blanket, no other node outside this set can provide any additional information about the node in question (S), as it is then separated form the network.

Question 4 gives a different result because of the restriction "the predictive variables are independent given the class variable", which restricts us to a too underfitted network. See below.

```
graphviz.plot(nb_network)
```