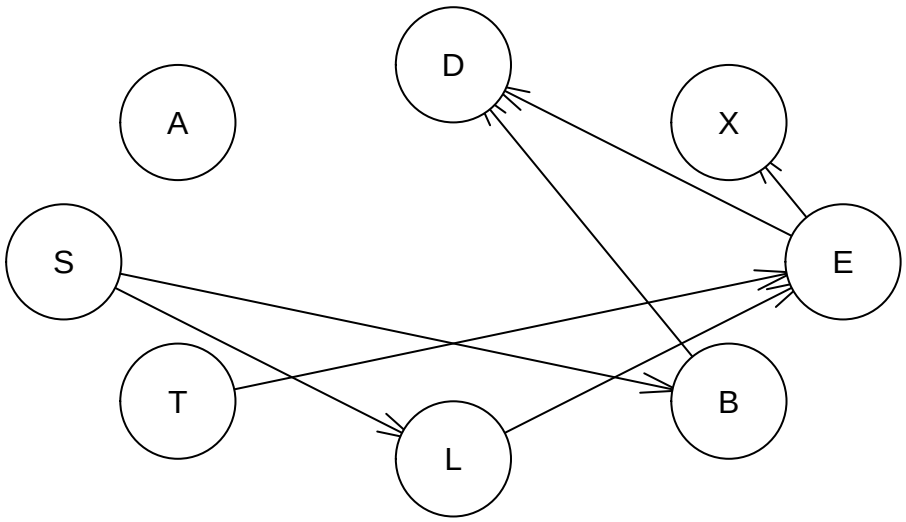
1.1

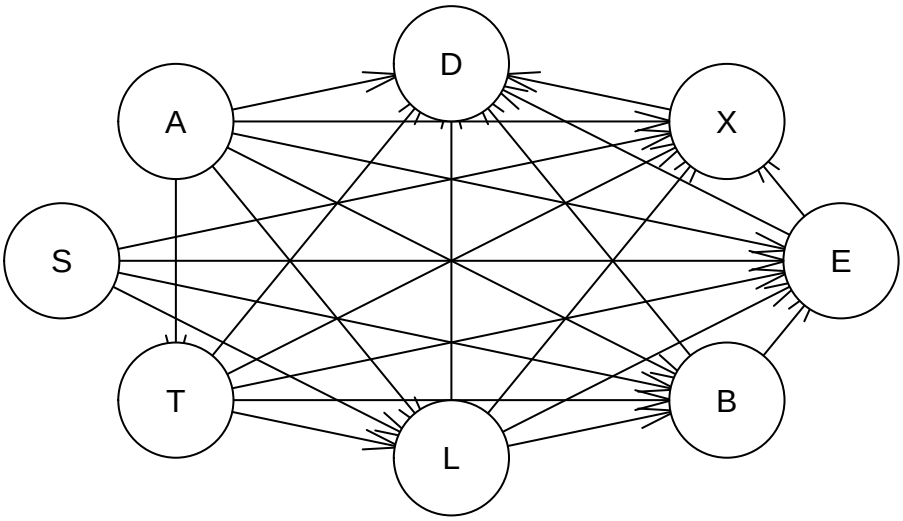
Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset, which is included in the `bnlearn` package. To load the data, run `data("asia")`. Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning. - Hint: Check the function `hc` in the `bnlearn` package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions `plot`, `arcs`, `vstructs`, `cpdag`, and `all.equal`.

```
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)))
```

Answer: 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
```

1.2

Learn a BN from 80% of the Asia dataset. 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 dataset in two classes: S = yes and S = no. Compute the posterior probability

distribution of S for each case and classify it in the most likely class. - You may use exact or approximate inference with the help of the `bnlearn` and `gRain` packages. Report the confusion matrix (true/false positives/negatives). Compare your results with those of the true Asia BN by running: `dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")`.

```
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
```

```
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)
```

```
##
```

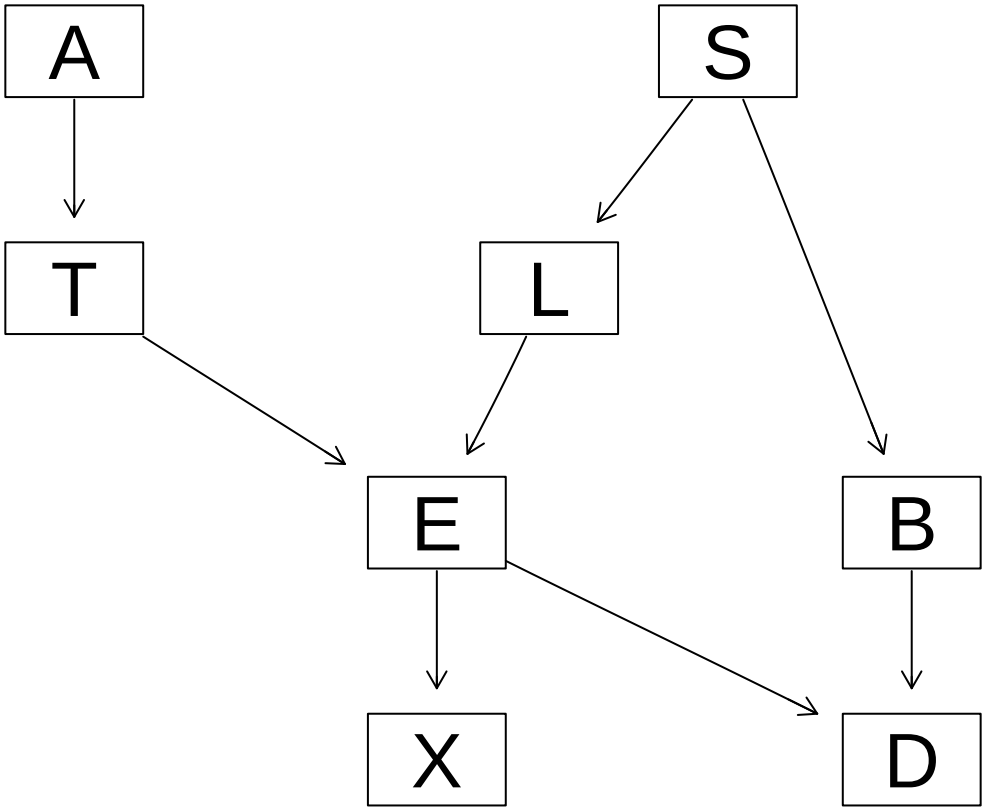
```
##        no yes
##   no  337 121
##   yes 176 366
```

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

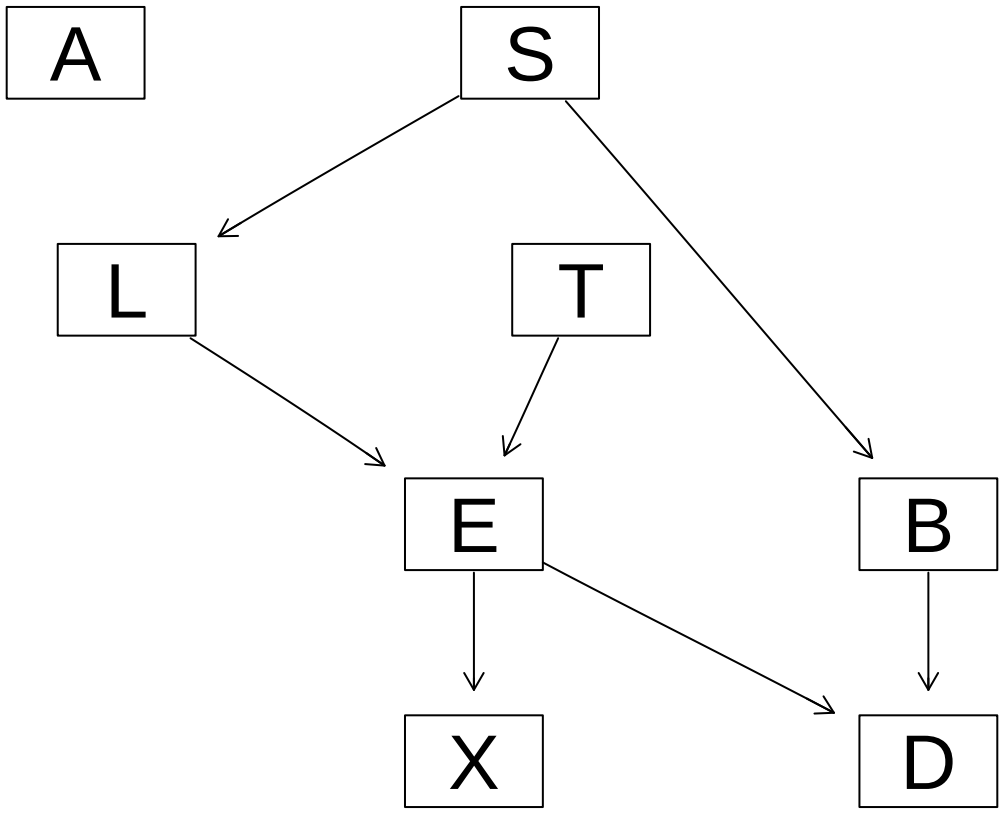
```
##
##        no yes
##   no  337 121
##   yes 176 366
```

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

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




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



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

1.3

Classify S given observations only for the Markov blanket of S (its parents, children, and the parents of its children minus S itself). Report the confusion matrix again. - Hint: You may want to use the function `mb` from the `bnlearn` package.

Definition: "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
```

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

1.4

Repeat exercise (2) using a naive Bayes classifier. Model it as a BN (Bayesian Network). Create the BN by hand; do not use `naive.bayes` from the `bnlearn` package. - Hint: Check http://www.bnlearn.com/examples/dag/ for information on how to create a BN by hand.

```
# 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
```

Answer: The confusion matrices now differ.

1.5

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

Answer: 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)
```