```r
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("RBGL", force = TRUE)
```

```
## Bioconductor version 3.19 (BiocManager 1.30.25), R 4.4.1 (2024-06-14 ucrt)

## Installing package(s) 'RBGL'

## package 'RBGL' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Hjalmar Öhman\AppData\Local\Temp\RtmpEb2ID5\downloaded_packages

## Old packages: 'digest', 'xfun'
```

```r
BiocManager::install("Rgraphviz", force = TRUE)
```

```
## Bioconductor version 3.19 (BiocManager 1.30.25), R 4.4.1 (2024-06-14 ucrt)

## Installing package(s) 'Rgraphviz'

## package 'Rgraphviz' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'Rgraphviz'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Hjalmar
## Öhman\AppData\Local\R\win-library\4.4\00LOCK\Rgraphviz\libs\x64\Rgraphviz.dll
## to C:\Users\Hjalmar
## Öhman\AppData\Local\R\win-library\4.4\Rgraphviz\libs\x64\Rgraphviz.dll:
## Permission denied

## Warning: restored 'Rgraphviz'

##
## The downloaded binary packages are in
##  C:\Users\Hjalmar Öhman\AppData\Local\Temp\RtmpEb2ID5\downloaded_packages

## Old packages: 'digest', 'xfun'
```

```r
BiocManager::install("gRain", force = TRUE)
```

```
## Bioconductor version 3.19 (BiocManager 1.30.25), R 4.4.1 (2024-06-14 ucrt)

## Installing package(s) 'gRain'

## package 'gRain' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'gRain'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Hjalmar
## Öhman\AppData\Local\R\win-library\4.4\00LOCK\gRain\libs\x64\gRain.dll to
## C:\Users\Hjalmar
## Öhman\AppData\Local\R\win-library\4.4\gRain\libs\x64\gRain.dll: Permission
## denied

## Warning: restored 'gRain'

##
## The downloaded binary packages are in
##  C:\Users\Hjalmar Öhman\AppData\Local\Temp\RtmpEb2ID5\downloaded_packages

## Old packages: 'digest', 'xfun'
```

1

```
rm(list=ls())
library(bnlearn)
set.seed(1)
data("asia")

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

```
##      from to
## [1,] "B"  "D"
## [2,] "L"  "E"
## [3,] "E"  "X"
## [4,] "S"  "B"
## [5,] "T"  "E"
## [6,] "E"  "D"
## [7,] "S"  "L"
```

```
vstructs(bn1)
```

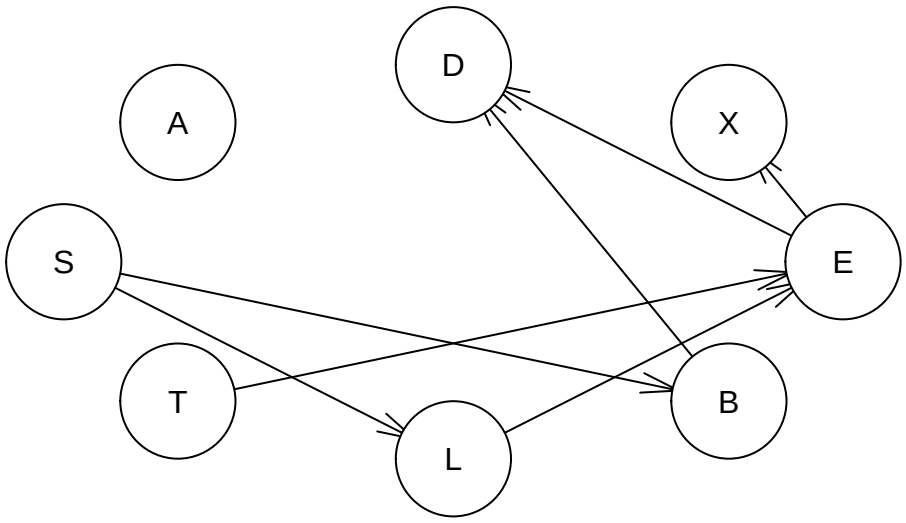
```
##      X   Z   Y
## [1,] "T" "E" "L"
## [2,] "B" "D" "E"
```

```
cpdag(bn1)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
```

```
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     2
##     directed arcs:                       5
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.62
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 Bayesian Dirichlet (BDe)
##   graph prior:                           Uniform
##   imaginary sample size:                 1
##   tests used in the learning procedure:  91
##   optimized:                             TRUE
```

```
plot(bn1)
```




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

```
##       from to
##  [1,] "B"  "D"
##  [2,] "L"  "E"
##  [3,] "E"  "X"
##  [4,] "S"  "B"
```

```
##  [5,] "T"  "E"
##  [6,] "A"  "T"
##  [7,] "A"  "L"
##  [8,] "L"  "X"
##  [9,] "A"  "E"
## [10,] "E"  "D"
## [11,] "T"  "X"
## [12,] "T"  "L"
## [13,] "S"  "L"
## [14,] "L"  "D"
## [15,] "A"  "X"
## [16,] "T"  "D"
## [17,] "T"  "B"
## [18,] "A"  "D"
## [19,] "A"  "B"
## [20,] "S"  "E"
## [21,] "B"  "E"
## [22,] "S"  "X"
## [23,] "L"  "B"
## [24,] "X"  "D"
```

```
vstructs(bn2)
```

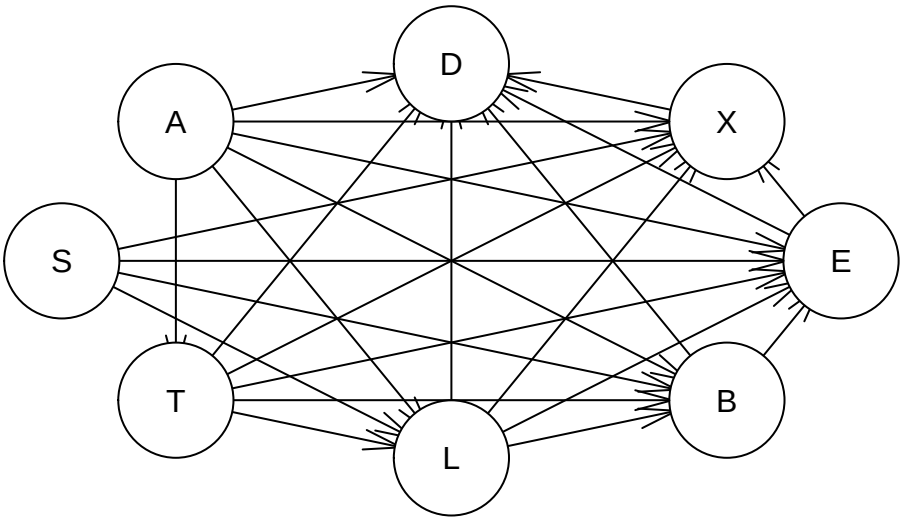
```
##       X   Z   Y
##  [1,] "A" "L" "S"
##  [2,] "S" "L" "T"
##  [3,] "A" "B" "S"
##  [4,] "S" "B" "T"
##  [5,] "A" "E" "S"
##  [6,] "S" "E" "T"
##  [7,] "A" "X" "S"
##  [8,] "S" "X" "T"
##  [9,] "B" "D" "X"
```

```
cpdag(bn2)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  24
##     undirected arcs:                     6
##     directed arcs:                       18
##   average markov blanket size:           6.75
##   average neighbourhood size:            6.00
##   average branching factor:              2.25
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 Bayesian Dirichlet (BDe)
##   graph prior:                           Uniform
##   imaginary sample size:                 1000
```

```
##   tests used in the learning procedure:  210
##   optimized:                             TRUE
```

```
plot(bn2)
```




```
all.equal(bn1, bn2)
```

```
## [1] "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.

2

```
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(1)
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  345 117
##   yes 165 373
```

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

```
##
##        no yes
##   no  345 117
##   yes 164 372
```

The confusion matrices are very similar.

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  345 117
##   yes 165 373
```

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

```
##
##        no yes
##   no  345 117
##   yes 165 373
```

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  381 181
##   yes 129 309
```

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

```
##
##        no yes
##   no  345 117
##   yes 164 372
```

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 non-complex network. See below.

```
plot(nb_network)
```