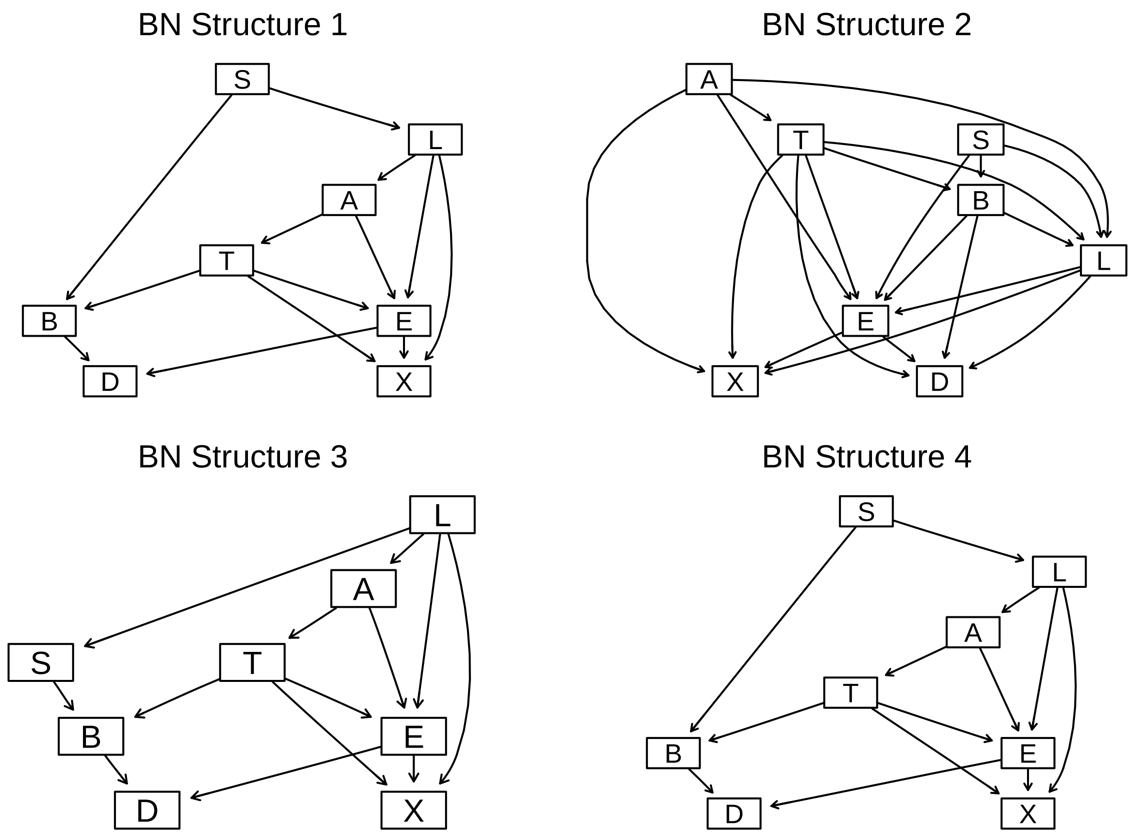
Lab1

2024-09-10

1 Non-equivalent BN structures using Hill Climb algorithm

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

# Define paramters
nIter = 4
num_restarts_list = c(5, 5, 5, 100)
iss_value_list = c(10, 100, 10, 10)

runHC = function(data, nIter, num_restarts, iss_value) {
  results_list = list()
  for (i in 1:nIter) {
    hc = hc(asia, score = "bde", iss = iss_value[i], restart = num_restarts[i])
    results_list[[i]] = hc
  }
  return(results_list)
}

hc_results = runHC(data, nIter, num_restarts_list, iss_value_list)

par(mfrow = c(2, 2))
for (i in 1:nIter) {
  graphviz.plot(hc_results[[i]], main = paste("BN Structure", i))
}
```

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




```r
#Compare arcs
arcs1 = arcs(hc_results[[1]])
arcs2 = arcs(hc_results[[2]])
arcs3 = arcs(hc_results[[3]])
arcs4 = arcs(hc_results[[4]])
arc_matrix = matrix(NA, nrow = max(nrow(arcs1), nrow(arcs2), nrow(arcs3), nrow(arcs4)), ncol = 8,
                    dimnames = list(NULL, c("From_1", "To_1", "From_2", "To_2","From_3", "To_3",
                                            "From_4", "To_4")))
arc_matrix[1:nrow(arcs1), 1:2] = arcs1
arc_matrix[1:nrow(arcs2), 3:4] = arcs2
arc_matrix[1:nrow(arcs3), 5:6] = arcs3
arc_matrix[1:nrow(arcs4), 7:8] = arcs4
print(arc_matrix)
```

```
##       From_1 To_1 From_2 To_2 From_3 To_3 From_4 To_4
##  [1,] "B"    "D"  "L"    "E"  "B"    "D"  "L"    "E"
##  [2,] "L"    "E"  "E"    "X"  "E"    "X"  "B"    "D"
##  [3,] "S"    "B"  "S"    "B"  "S"    "B"  "T"    "B"
##  [4,] "T"    "E"  "T"    "E"  "T"    "E"  "A"    "T"
##  [5,] "E"    "D"  "S"    "L"  "E"    "D"  "A"    "E"
##  [6,] "A"    "E"  "A"    "T"  "A"    "E"  "E"    "X"
##  [7,] "L"    "X"  "A"    "E"  "L"    "X"  "L"    "X"
##  [8,] "T"    "X"  "T"    "L"  "T"    "X"  "E"    "D"
##  [9,] "T"    "B"  "A"    "X"  "T"    "B"  "T"    "X"
## [10,] "E"    "X"  "L"    "X"  "L"    "A"  "T"    "E"
## [11,] "L"    "A"  "T"    "X"  "L"    "E"  "S"    "B"
## [12,] "S"    "L"  "L"    "D"  "L"    "S"  "S"    "L"
## [13,] "A"    "T"  "T"    "D"  "A"    "T"  "L"    "A"
```

```
## [14,] NA     NA   "S"    "E"  NA     NA   NA     NA
## [15,] NA     NA   "B"    "L"  NA     NA   NA     NA
## [16,] NA     NA   "B"    "E"  NA     NA   NA     NA
## [17,] NA     NA   "E"    "D"  NA     NA   NA     NA
## [18,] NA     NA   "B"    "D"  NA     NA   NA     NA
## [19,] NA     NA   "A"    "L"  NA     NA   NA     NA
## [20,] NA     NA   "T"    "B"  NA     NA   NA     NA
```

```
# check for colliders
vstructs(hc_results[[1]])
```

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

```
vstructs(hc_results[[2]])
```

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

```
#CPDAG
cpdag(hc_results[[3]])
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  13
##     undirected arcs:                     3
##     directed arcs:                       10
##   average markov blanket size:           4.00
##   average neighbourhood size:            3.25
##   average branching factor:              1.25
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 Bayesian Dirichlet (BDe)
##   graph prior:                           Uniform
##   imaginary sample size:                 10
##   tests used in the learning procedure:  202
##   optimized:                             TRUE
```

```
graphviz.plot(cpdag(hc_results[[4]]))

#all equal (in terms of network equivalence, all graphs are converted to CPDAGs for comparison)
all.equal(cpdag(hc_results[[1]]), cpdag(hc_results[[2]]))
```

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

```r
all.equal(cpdag(hc_results[[1]]), cpdag(hc_results[[3]]))
```

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

```r
all.equal(cpdag(hc_results[[1]]), cpdag(hc_results[[4]]))
```

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

```r
all.equal(cpdag(hc_results[[2]]), cpdag(hc_results[[3]]))
```

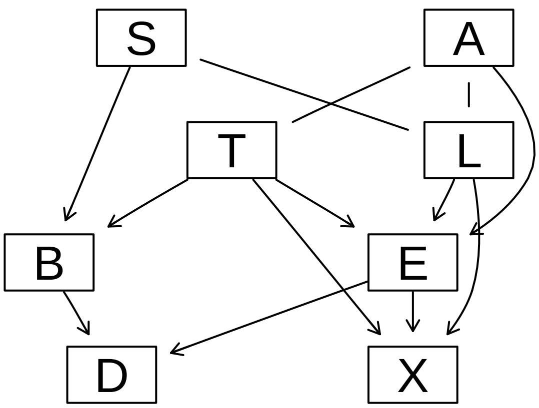
```
## [1] "Different number of directed/undirected arcs"
```

```r
all.equal(cpdag(hc_results[[2]]), cpdag(hc_results[[4]]))
```

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

```r
all.equal(cpdag(hc_results[[3]]), cpdag(hc_results[[4]]))
```

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



The Imaginary Sample Size (ISS) affects the number of arcs (directed edges) being applied in the BN. The larger ISS input the more edges (vizualised by BN 1 & 2). Since the HC algorithm does a greedy search and finds local optimums for the BNs, the ISS affects the results by giving different weights to the prior. When the ISS is small, the algorithm trusts the data more and is more conservative in adding edges. A large ISS makes the network more likely to include edges, even if the evidence in the data is weak, because it puts more weight on the prior.

The number of random restarts also affects the BN, in a less significant way, only changing one edge (L,S). (visualised in BN 3 & 4). The random restart affects the search by giving the HC algorithm more chances to find different optimums.

Furthermore, the as visualized by the networks and by the all.equal() function. BN 1&2, 2&3, 2&4 are non-equivalent.

2 Learn a BN from 80 % of the Asia dataset.

```r
set.seed(12345)
library(bnlearn)
library(gRain)
```

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

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

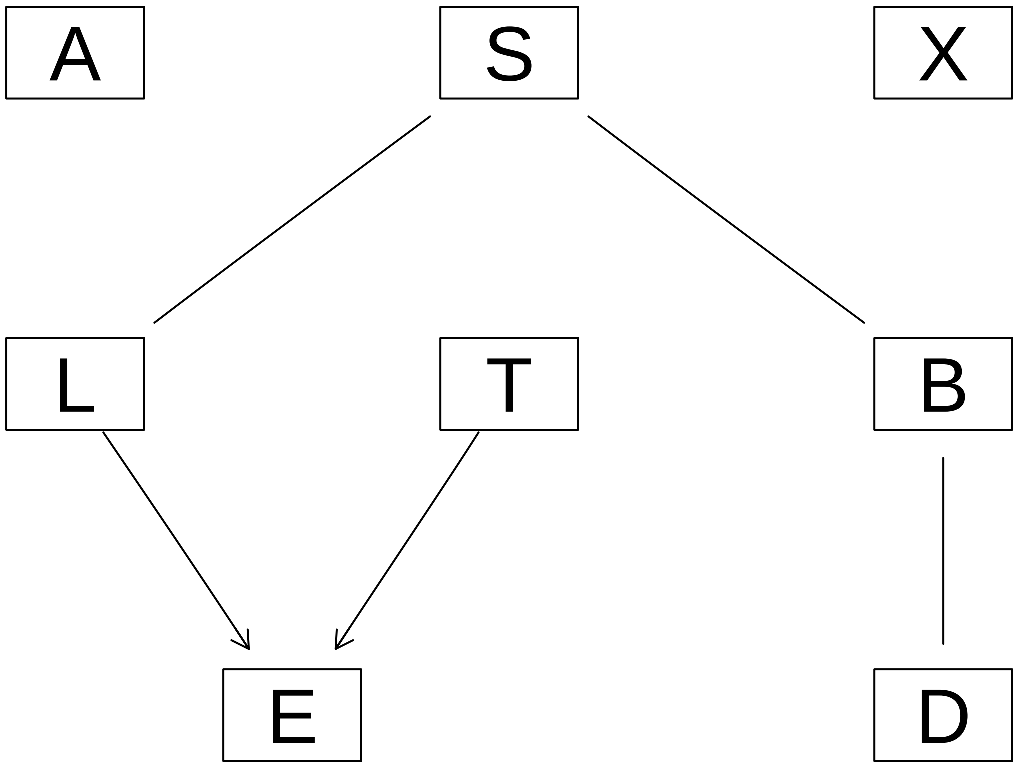
```
## The following objects are masked from 'package:bnlearn':
##
```

```
##     ancestors, children, nodes, parents
```

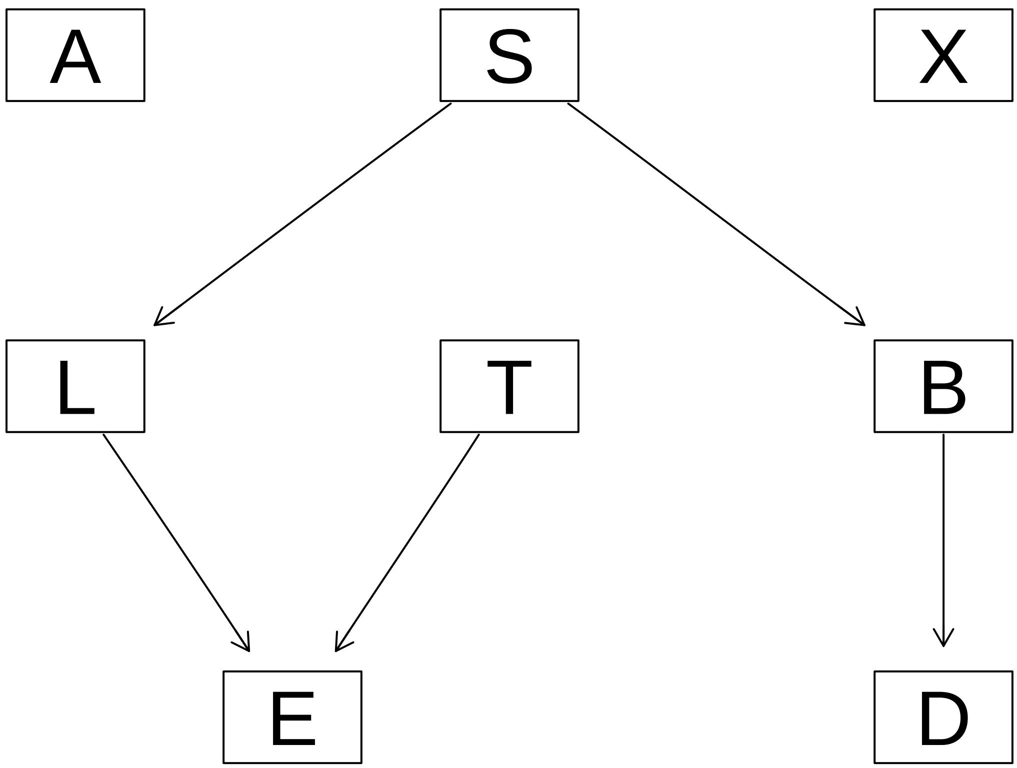
```r
data("asia")

#Divide data into training and validation data
n_rows = nrow(asia)
train_indices = sample(1:n_rows, size = 0.8 * n_rows)
learningData = asia[train_indices, ]
testingData = asia[-train_indices, ]

pdag = pc.stable(learningData)
print(pdag)
```

```
##
##   Bayesian network learned via Constraint-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  5
##     undirected arcs:                     3
##     directed arcs:                       2
##   average markov blanket size:           1.50
##   average neighbourhood size:            1.25
##   average branching factor:              0.25
##
##   learning algorithm:                    PC (Stable)
##   conditional independence test:         Mutual Information (disc.)
##   alpha threshold:                       0.05
##   tests used in the learning procedure:  128
```

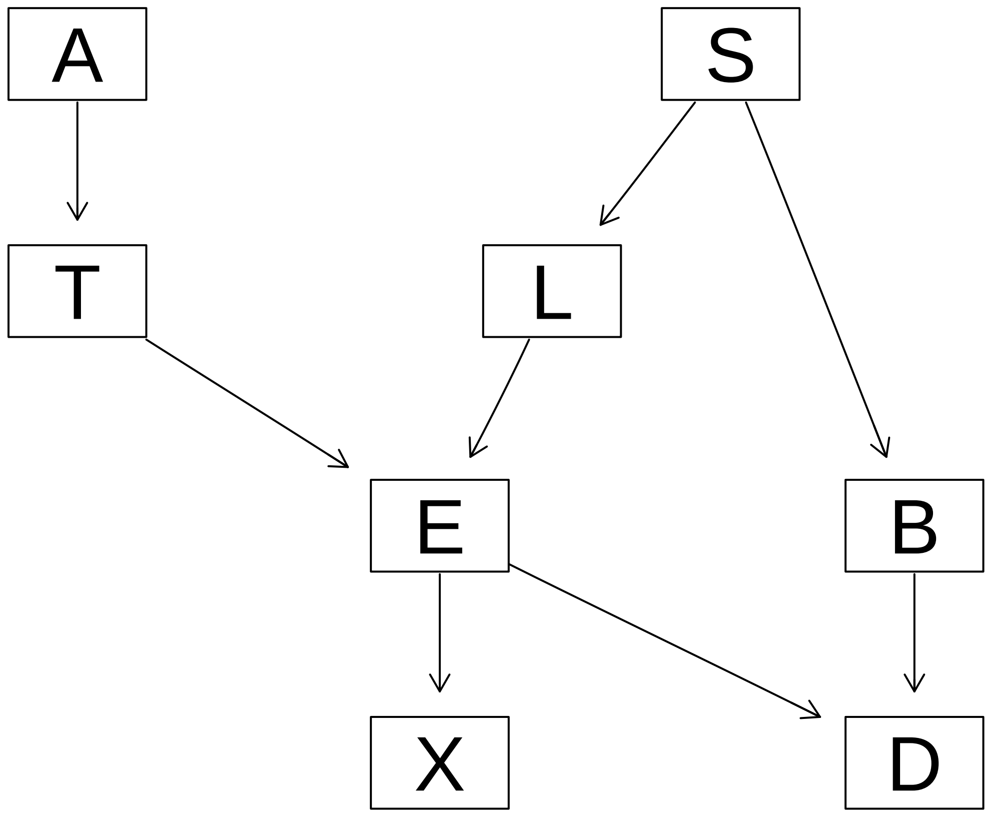
```r
graphviz.plot(pdag)
```




```
# The PC returns 3 undirected archs, (IAMB returns 0),
# but cextend will be used to arbitrarily choose the last directions
pcdag = cextend(pdag)
graphviz.plot(pcdag)
```



```r
classify = function(network, nodes_in, test_data_row) {
  evidence = test_data_row[nodes_in]
  numbers = as.character(evidence)
  evidence_state = c(NULL, length = length(numbers)) # convert the factors to yes or no
  for (i in 1:length(as.character(evidence))) {
    if (numbers[i] == 2) {
      evidence_state[i] = "yes"
    } else {
      evidence_state[i] = "no"
    }
  }
  obs_evidence = setEvidence(network, nodes = nodes_in,
                             states = evidence_state)
  posterior = querygrain(obs_evidence, nodes = "S")$S
  return(ifelse(posterior["yes"] > posterior["no"], "yes", "no"))
}

confusionMatrix1 = function(actual, predicted) {
  confusion_matrix = matrix(0, nrow = 2, ncol = 2)
  for (i in 1:length(actual)) {
    if (actual[i] == "yes" && predicted[i] == "yes") {
      confusion_matrix[1, 1] = confusion_matrix[1, 1] + 1  # TP
    } else if (actual[i] == "no" && predicted[i] == "no") {
      confusion_matrix[2, 2] = confusion_matrix[2, 2] + 1  # TN
    } else if (actual[i] == "yes" && predicted[i] == "no") {
      confusion_matrix[2, 1] = confusion_matrix[2, 1] + 1  # FN
    } else if (actual[i] == "no" && predicted[i] == "yes") {
      confusion_matrix[1, 2] = confusion_matrix[1, 2] + 1  # FP
    }
  }
  rownames(confusion_matrix) <- c("Predicted Positive", "Predicted Negative")
  colnames(confusion_matrix) <- c("Actual Positive", "Actual Negative")
  return(confusion_matrix)
}

#Constructed Network
#Fit the BN structure using maximum likelihood estimators
fitted = bn.fit(pcdag, learningData, method = "mle")
# Fit as grain
grain_fit = as.grain(fitted)
compiled_grain = compile(grain_fit)

# The true Aisian Network
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.plot(dag)
```




```r
#Fit the BN structure using maximum likelihood estimators
fitted_true = bn.fit(dag, learningData, method = "mle")
# Fit as grain
grain_fit_true = as.grain(fitted_true)
compiled_grain_true = compile(grain_fit_true)

prediction = c(NULL, nrow(testingData))
prediction_T = c(NULL, nrow(testingData))

for (i in 1:nrow(testingData)) {
  prediction[i] = classify(compiled_grain,c("A", "T", "L", "B", "E", "X", "D"), testingData[i, ])
  prediction_T[i] = classify(compiled_grain_true,c("A", "T", "L", "B", "E", "X", "D"), testingData[i, ]
}

true_values = as.character(testingData$S)
confusionMatrix_own_model = confusionMatrix1(true_values, prediction)
confusionMatrix_true_model = confusionMatrix1(true_values, prediction_T)
print(confusionMatrix_own_model)
```

```
##                     Actual Positive Actual Negative
## Predicted Positive              366             176
## Predicted Negative              121             337
```

```r
print(confusionMatrix_true_model)
```

```
##                     Actual Positive Actual Negative
## Predicted Positive              366             176
## Predicted Negative              121             337
```

Even though the BNs are different, they perform exactly the the same in classifying S. This could be because the causal relationships that are missing in the reconstructed graph are not that strong in the true graph.

3 Classifying S using Markov blanket (predicting from parents)

```
markov_blanket = mb(fitted, node = "S")
markov_blanket_T = mb(fitted_true, node = "S")

mbclassification = c(NULL, nrow(testingData))
mbclassification_T = c(NULL, nrow(testingData))

for (i in 1:nrow(testingData)) {
  mbclassification[i] = classify(compiled_grain, markov_blanket, testingData[i, ])
  mbclassification_T[i] = classify(compiled_grain_true, markov_blanket_T, testingData[i, ])
}

confusionMatrix_mb_model = confusionMatrix1(true_values, mbclassification)
confusionMatrix_mb_true_model = confusionMatrix1(true_values, mbclassification_T)
print(confusionMatrix_mb_model)
```

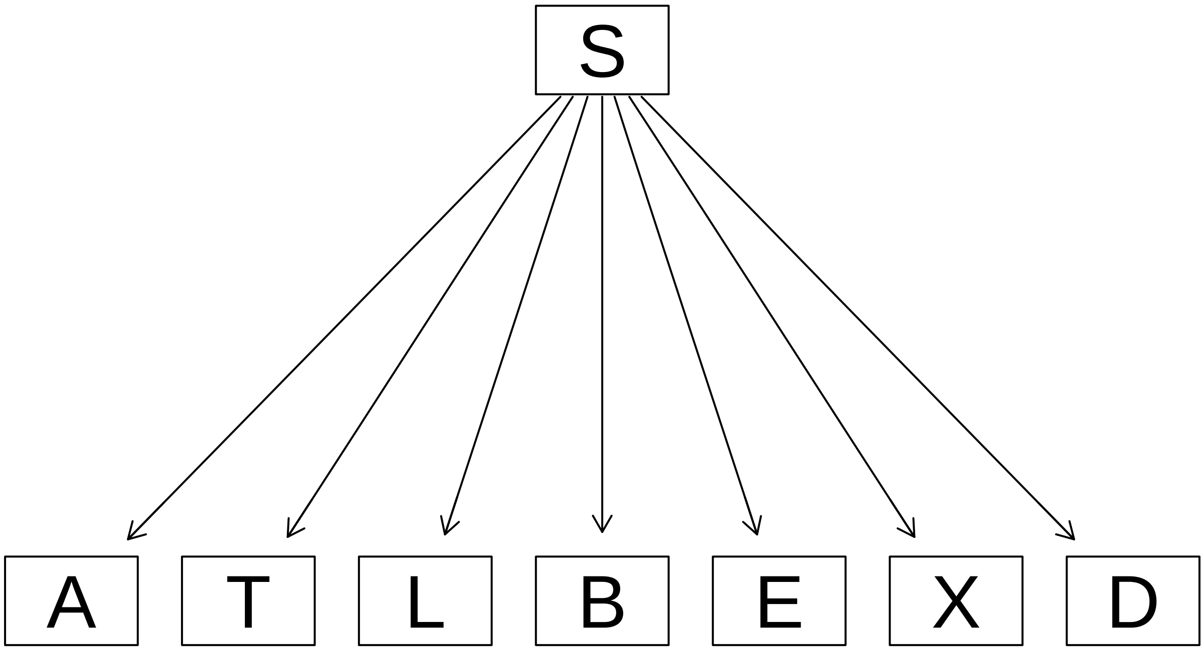
```
##                    Actual Positive Actual Negative
## Predicted Positive             366             176
## Predicted Negative             121             337
```

```
print(confusionMatrix_mb_true_model)
```

```
##                    Actual Positive Actual Negative
## Predicted Positive             366             176
## Predicted Negative             121             337
```

4 Using a naive Bayes classifier

```
#Naive assumption: independence between all features
arc.set = matrix(c("S", "A",
                   "S", "T",
                   "S", "L",
                   "S", "B",
                   "S", "E",
                   "S", "X",
                   "S", "D"),
                 ncol = 2, byrow = TRUE,
                 dimnames = list(NULL, c("from", "to")))

e = empty.graph(c("A", "T", "L", "B", "E", "X", "D", "S"))
arcs(e) = arc.set
graphviz.plot(e)
```




```r
fitted_naive = bn.fit(e, learningData)
grain_naive = as.grain(fitted_naive)
compiled_naive = compile(grain_naive)

prediction_naive = c(NULL, nrow(testingData))
for (i in 1:nrow(testingData)) {
  prediction_naive[i] = classify(compiled_naive,c("A", "T", "L", "B", "E", "X", "D"), testingData[i, ])
}

confusionMatrix_naive_model = confusionMatrix1(true_values, prediction_naive)
print(confusionMatrix_naive_model)
```

```
##                    Actual Positive Actual Negative
## Predicted Positive             307             154
## Predicted Negative             180             359
```

```r
print(confusionMatrix_true_model)
```

```
##                    Actual Positive Actual Negative
## Predicted Positive             366             176
## Predicted Negative             121             337
```

5 Discussion

The Markov blanket approach and the full Bayesian network yield similar results because both rely on the correct structure of dependencies in the network, with the Markov blanket focusing on the minimal sufficient set of nodes needed to predict S. On the other hand, the naive Bayes classifier produces different results because it makes the unrealistic assumption that all predictive variables are conditionally independent given S, leading to a loss of dependency information.