TDDE15-Lab 1

Fredrik Ramberg

Libraries and data imports.

```
library(bnlearn)
library(gRain)
```

```
## Loading required package: gRbase

##
## Attaching package: 'gRbase'

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

```
data("asia")
```

Task 1

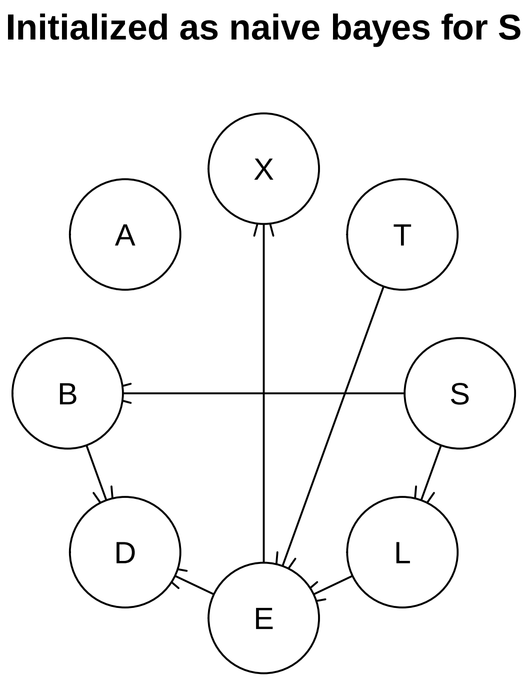
Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is in- cluded in the ***bnlearn*** *package. To load the data, run* ***data("asia")****. Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning.*

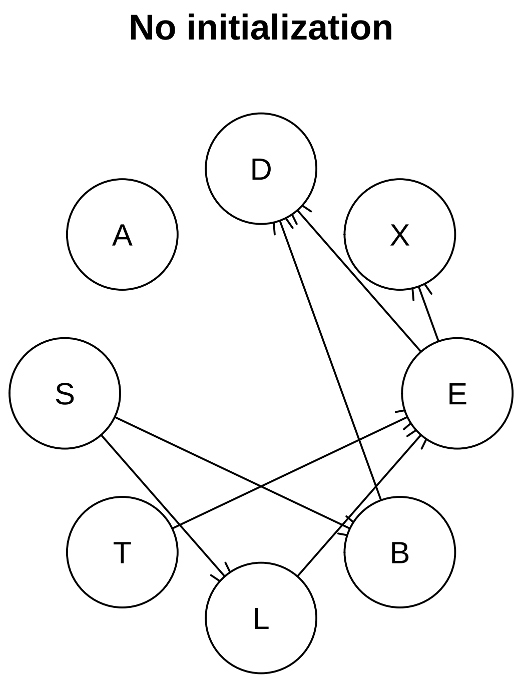
Hint: *Check the function* ***hc*** *in the* ***bnlearn*** *package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sam- ple size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions* ***plot****,* ***arcs****,* ***vstructs****,* ***cpdag*** *and* ***all.equal***

```
random_restarts <- c(1,10,100)
score <- c("bic","aic","bde")
im_sample_size <- c(1,10,100)
init <- model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
```

Initialized graph or not

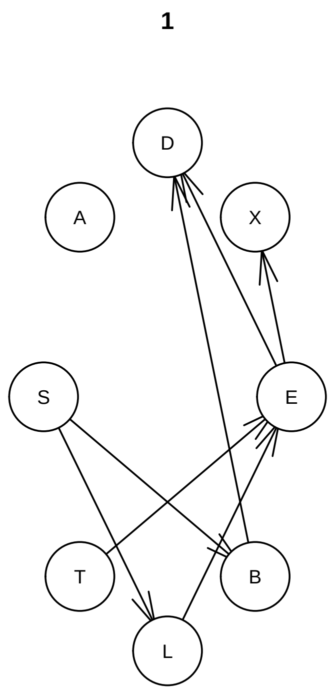
```
init_dag <- hc(asia, start = init)
uninit_dag <- hc(asia)

par(mfrow = c(1,2))
plot(init_dag, main = "Initialized as naive bayes for S")
plot(uninit_dag, main = "No initialization")
```

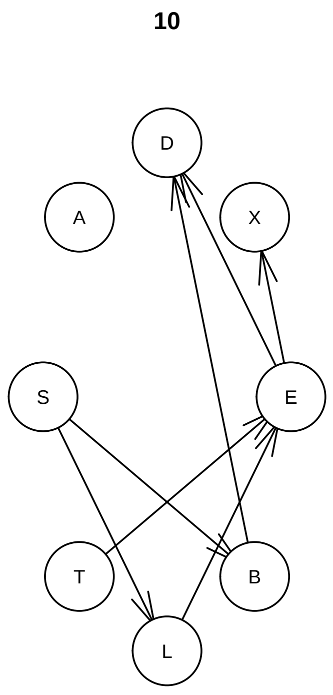


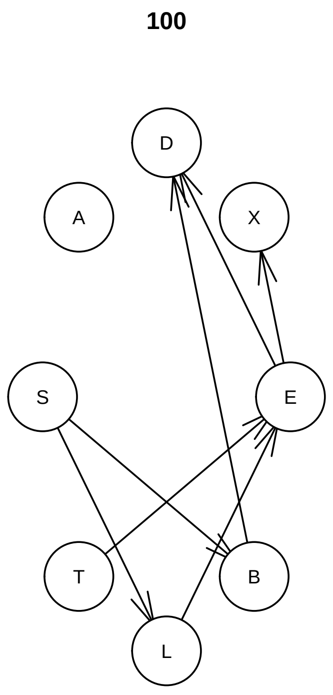


Different random restarts

```
restart_dag1 <- hc(asia, restart = random_restarts[1])
restart_dag2 <- hc(asia, restart = random_restarts[2])
restart_dag3 <- hc(asia, restart = random_restarts[3])

par(mfrow = c(1,3))
plot(restart_dag1, main = random_restarts[1])
plot(restart_dag2, main = random_restarts[2])
plot(restart_dag3, main = random_restarts[3])
```

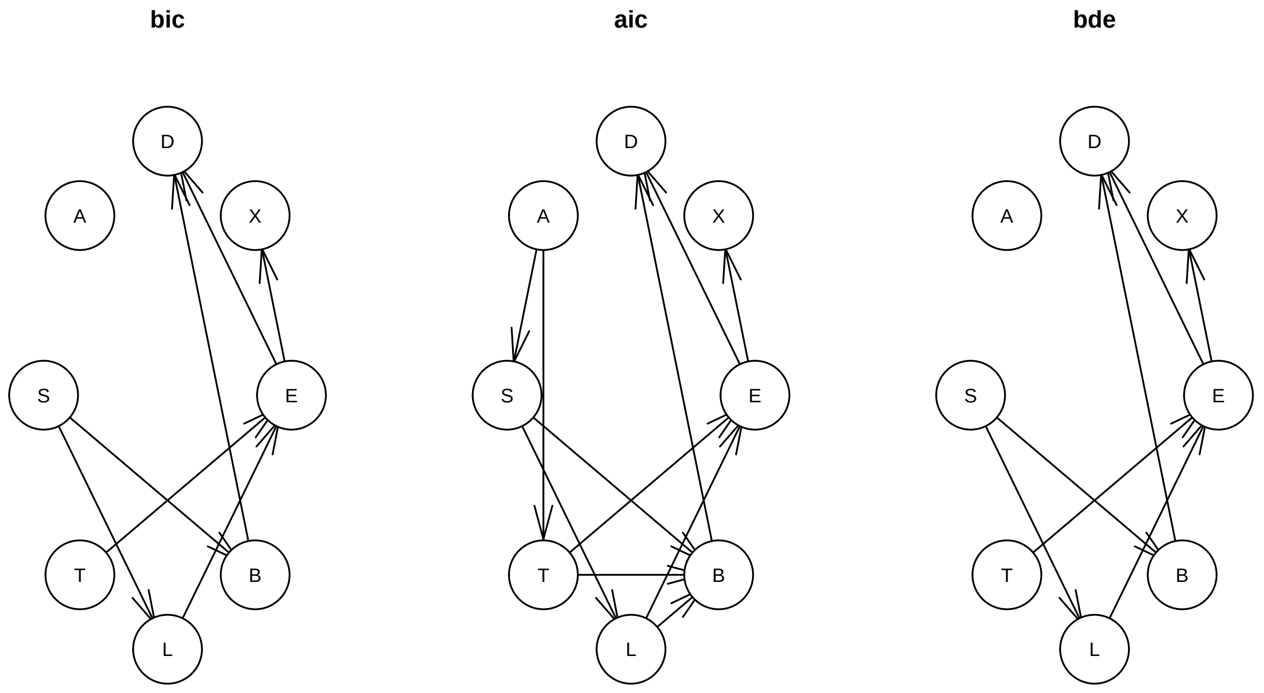






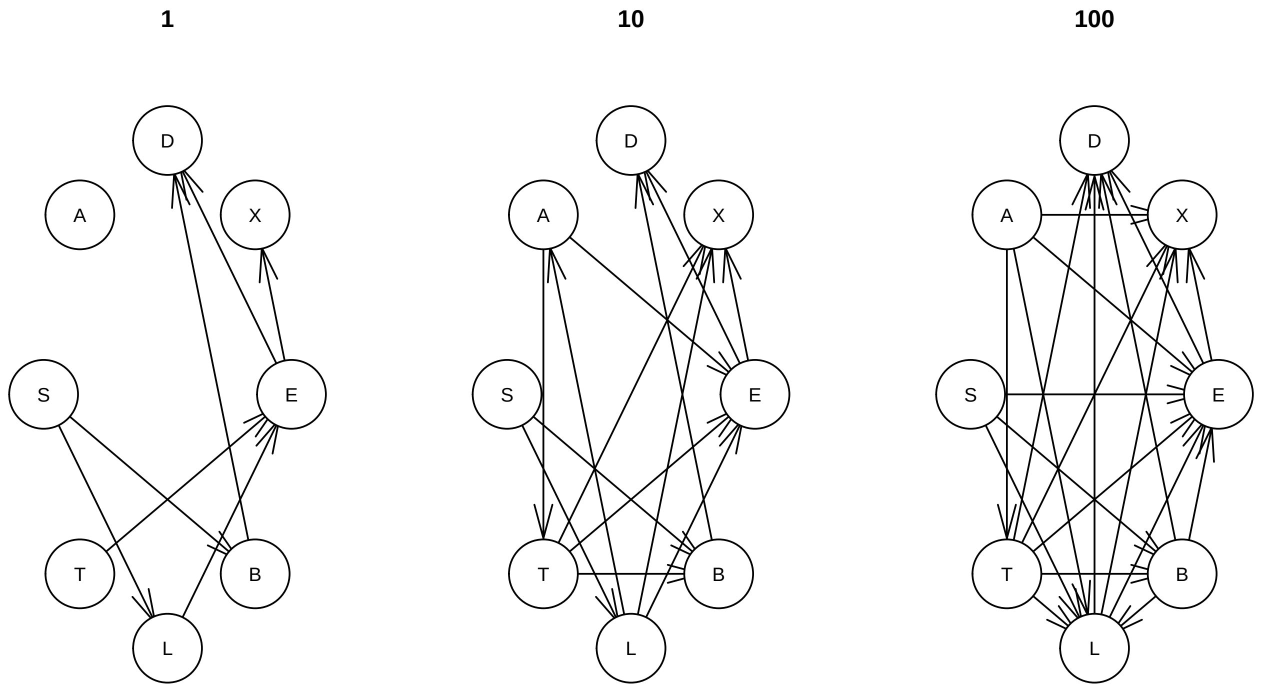
Different scores

```r
score_dag1 <- hc(asia, score = score[1])
score_dag2 <- hc(asia, score = score[2])
score_dag3 <- hc(asia, score = score[3])

par(mfrow = c(1,3))
plot(score_dag1, main = score[1])
plot(score_dag2, main = score[2])
plot(score_dag3, main = score[3])
```



Different imaginary sample sizes (iss)

```r
im_sample_dag1 <- hc(asia, score = "bde", iss = im_sample_size[1])
im_sample_dag2 <- hc(asia, score = "bde", iss = im_sample_size[2])
im_sample_dag3 <- hc(asia, score = "bde", iss = im_sample_size[3])

par(mfrow = c(1,3))
plot(im_sample_dag1, main = im_sample_size[1])
plot(im_sample_dag2, main = im_sample_size[2])
plot(im_sample_dag3, main = im_sample_size[3])
```



Discussion

We see very little difference between different random restart values, this is probably because the nr of nodes are so small that few local optima exist other than global. The regularization term iss in the otherhand has a large impact on the graph drastically reducing the number of arcs for lower iss values.

Task 2

*Learn a BN from 80 % of the Asia dataset. The dataset is included in the bnlearn package. To load the data, run **data("asia")**. 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 Asia dataset in two classes: S = yes and S = no. In other words, compute the posterior probability distribution of S for each case and classify it in the most likely class. To do so, you have to use exact or approximate inference with the help of the bnlearn and gRain packages, i.e. you are not allowed to use functions such as predict. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asia BN, which can be obtained by running **dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]").***

***Hint:** You already know two algorithms for exact inference in BNs: Variable elimi- nation and cluster trees. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the functions **bn.fit** and **as.grain** from the **bnlearn package**, and the functions **compile**, **setEvidence** and **querygrain** from the package **gRain**. For approximate inference, you may need the functions **cpquery** or **cpdist**, **prop.table** and **table** from the **bnlearn** package.*

```r
set.seed(12345)

n <- nrow(asia)
train_indices <- sample(1:n, size = round(0.8 * n))

train_data <- asia[train_indices, ]
test_data <- asia[-train_indices, ]

#Learning the structure
dag <- hc(train_data)
true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
```

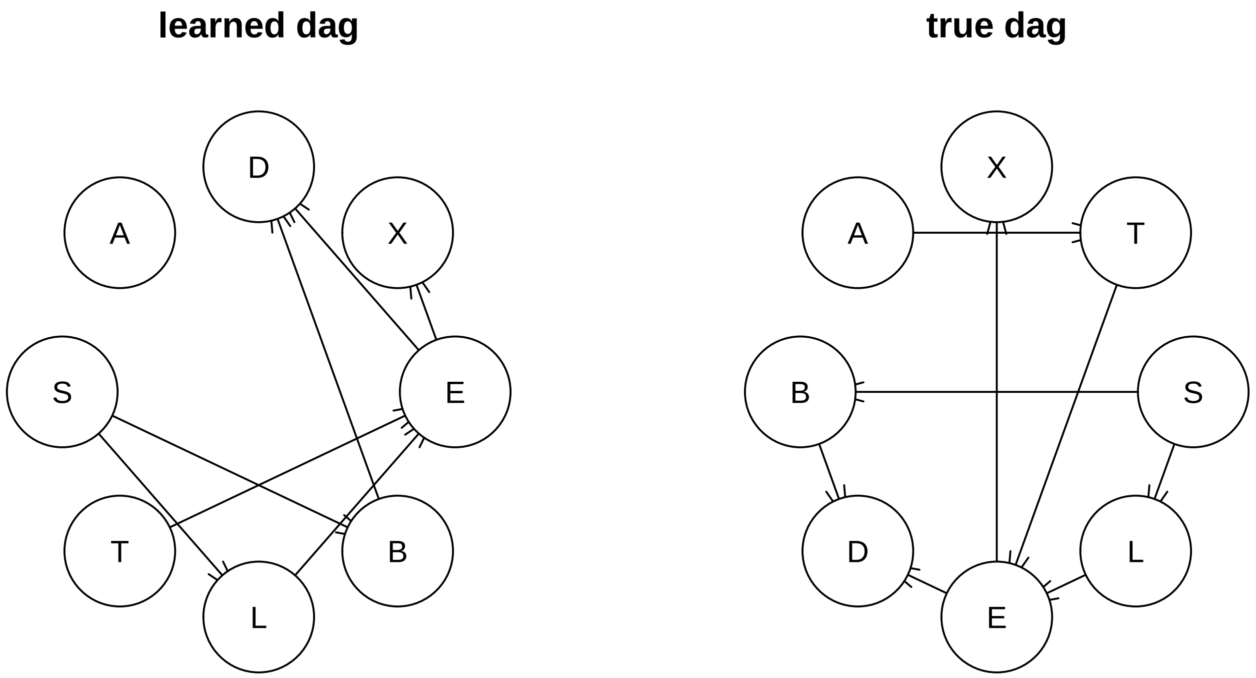
```r
#Learning the parameters
bn <- bn.fit(dag, train_data)
true_bn <- bn.fit(true_dag, train_data)

#function to predict S for any bn, data and condition
predict_S_from_bn <- function(bn, data, condition) {

  predicted_S <- c()

  bn_grain <- as.grain(bn) #convert to graintype for compatability

  #predict S for each row of data
  for (i in 1:nrow(data)){
    states <- as.vector(as.matrix(test_data[i,condition]))

    #applying the states
    bn_grain_with_condition <- setEvidence(bn_grain, nodes = condition, states = states)

    # Posterior probability for "yes" and "no" for each datapoint
    posterior_S <- querygrain(bn_grain_with_condition, nodes = "S")$S

    # Classification based on the posterior probability
    predicted_class <- ifelse(posterior_S["yes"] > posterior_S["no"], "yes", "no")

    # Store the predicted class
    predicted_S <- c(predicted_S, predicted_class)
  }

  # Return the predicted classes for S
  return(predicted_S)
}

#conditional 'evidence'
condition <- colnames(test_data)[-2]

fitted_s <- predict_S_from_bn(bn, test_data, condition)
true_fitted_s <- predict_S_from_bn(true_bn, test_data, condition)


table(Predicted_FDR = fitted_s, Actual = test_data$S)
```

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

```r
table(Predicted_True = true_fitted_s, Actual = test_data$S)
```

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

```r
par(mfrow = c(1,2))
plot(dag, main="learned dag")
plot(true_dag, main = "true dag")
```



Task 3:

In the previous exercise, you classified the variable S given observations for all the rest of the variables. Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix. Hint: You may want to use the function mb from the bnlearn package.

```r
markov_blanket <- mb(bn, "S")

predicted_S_mb <- predict_S_from_bn(bn,test_data,markov_blanket)

table(Predicted = predicted_S_mb, Actual = test_data$S)
```

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

Task 4:

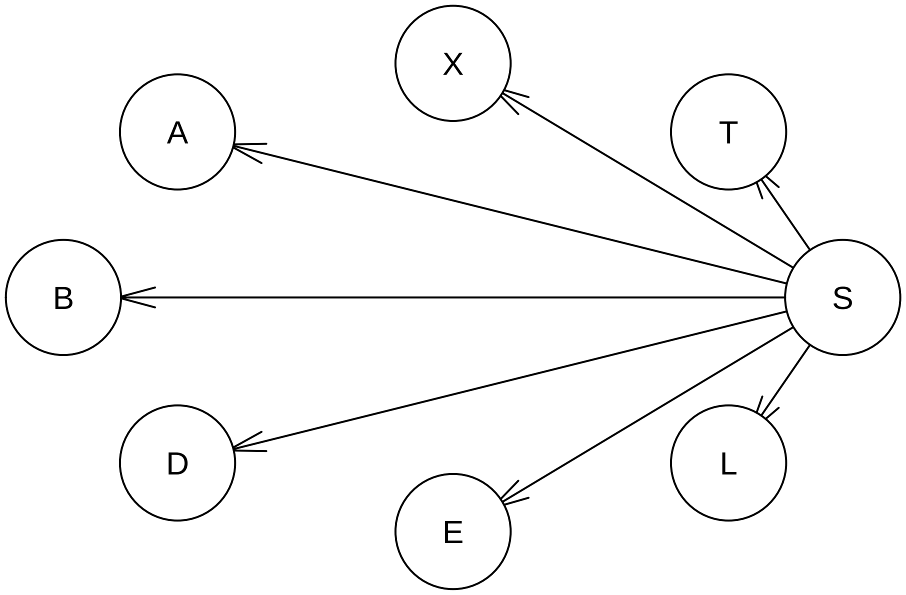
*Repeat the exercise (2) using a **naive Bayes classifier**, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You have to create the BN by hand, i.e. you are not allowed to use the function **naive.bayes** from the bnlearn package. Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand.*

```r
dag_naive_bayes <- model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")

bn_naive_bayes <- bn.fit(dag_naive_bayes, train_data)

predicted_S_naive_bayes <- predict_S_from_bn(bn_naive_bayes,test_data,condition)
```

```r
table(Prediction = predicted_S_naive_bayes, Actual = test_data$S)
```

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

```r
plot(dag_naive_bayes)
```



Task 5: Explain why you obtain the same or different results in the exercises (2-4).

2-3 Looks the same: Since the markov blanket contains all the parents and the children of the node S, we can be certain that that given the markov blanket, the node is seperated and therefore independent from the other nodes. In this case the markov blanket contains

4 Is different from 2-3: Since the Naive Bayes drastically simplifies the dependencies. We can also see that the model is biased towards "no" compared to the previous models.