TDDE15-Lab 1

frera064, oscho091, hjaoh082, olosw720

Contribution

All four members solved the problems on their own and discussed the answers. Task 1: frera064, Task 2 & 3: hjaoh082, Task 4: oscho091, Task 5: olosw720

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.*

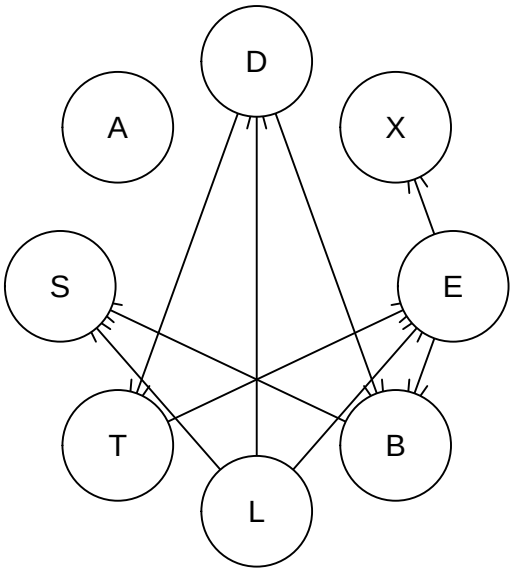
Hint: *Check the function* ***hc*** *in the* ***bnlearn*** *package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sam- ple size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You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

```
random_restarts <- c(0,10,100)
score <- c("bic","aic","bde")
im_sample_size <- c(1,10,100)
init <- random.graph(colnames(asia))#model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
```

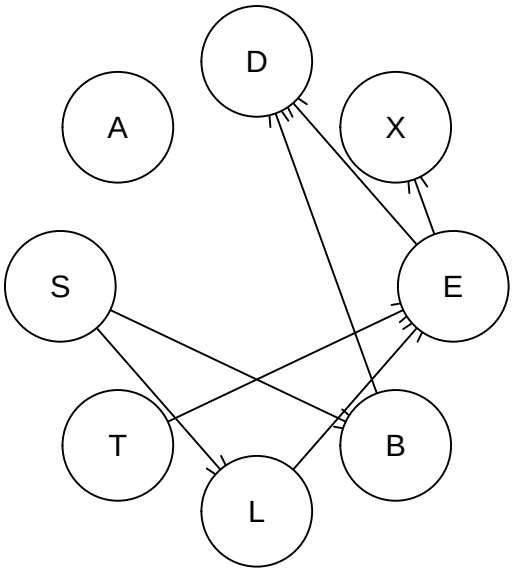
Initialized graph or not

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

par(mfrow = c(1,2))
plot(init_dag, main = "Random initialization for S")
plot(uninit_dag, main = "No initialization")
```

Random initialization for S



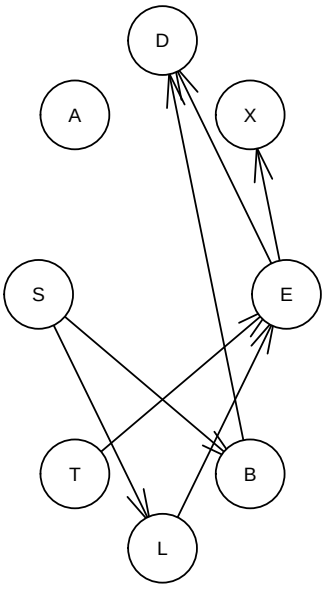
No initialization



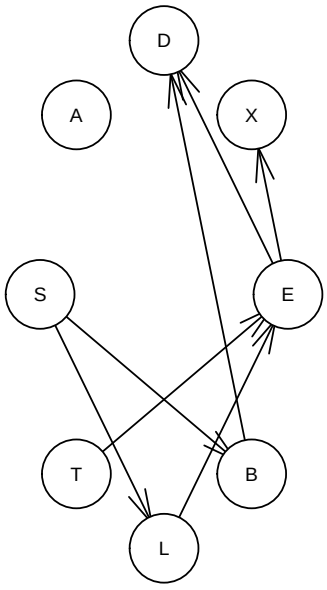
Different random restarts

```
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])
```

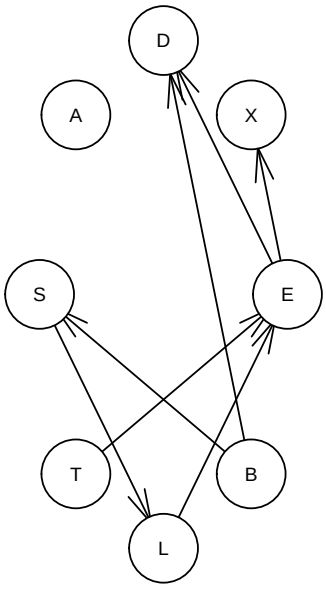
0



10



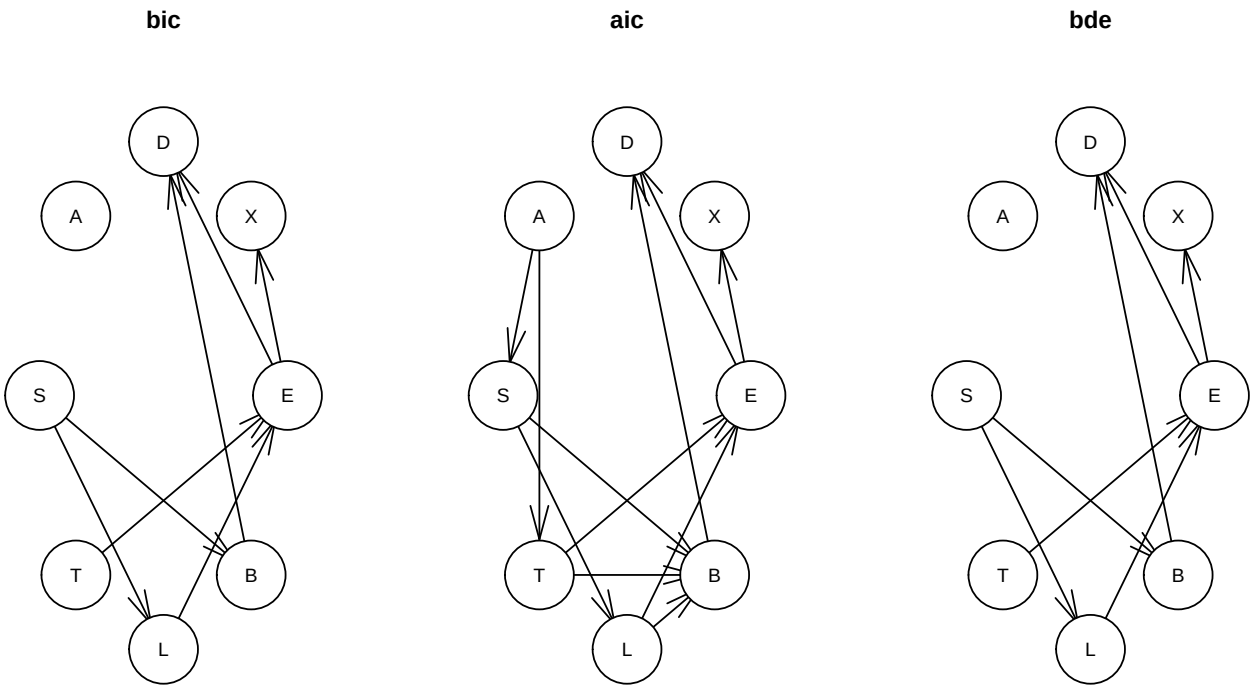
100



```
#all.equal(cpdag(restart_dag1), cpdag(restart_dag2)) #Compares equivalence
#all.equal(restart_dag1, restart_dag2)
```

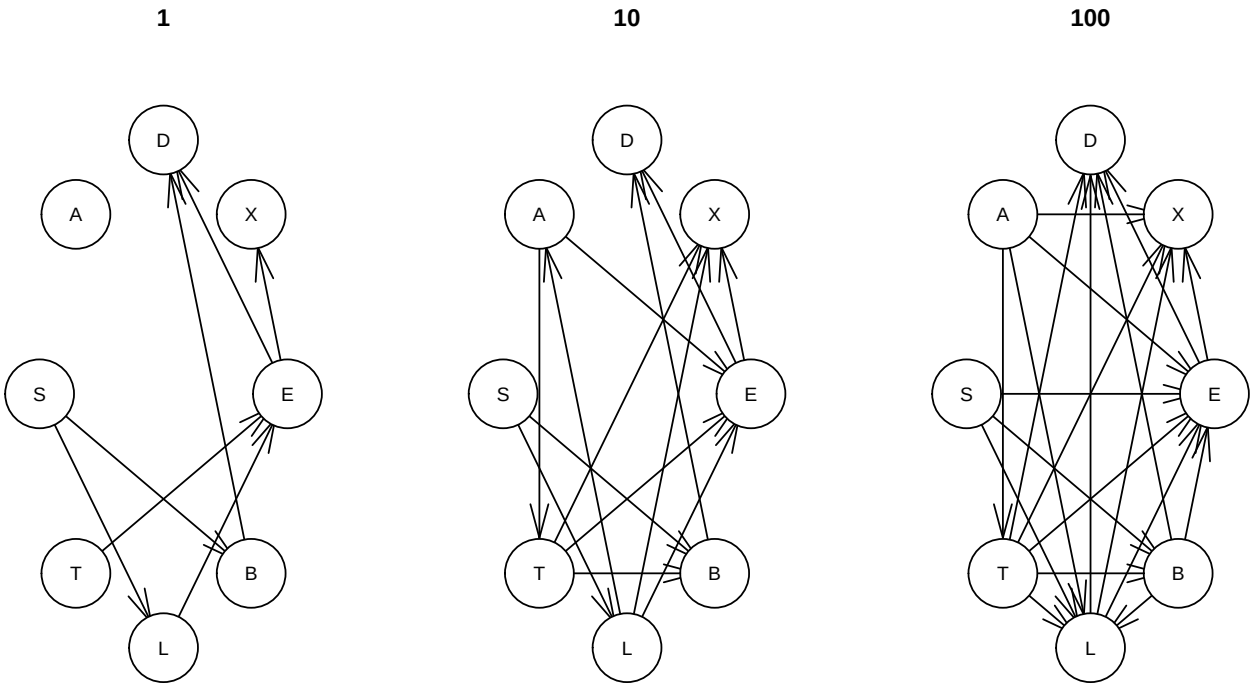
Different scores

```
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)

```
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])
```




```
#all.equal(cpdag(im_sample_dag1), cpdag(im_sample_dag3)) #Compares equivalence
```

Discussion

The hill climbing algorithm is a heuristic and can find local optimas that are not global optimum and therefore produce different non-equivalent bns. 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 on the other hand has a large impact on the graph drastically reducing the number of arcs for lower iss values.

2

```
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]")

#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()
```

```r
  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
```

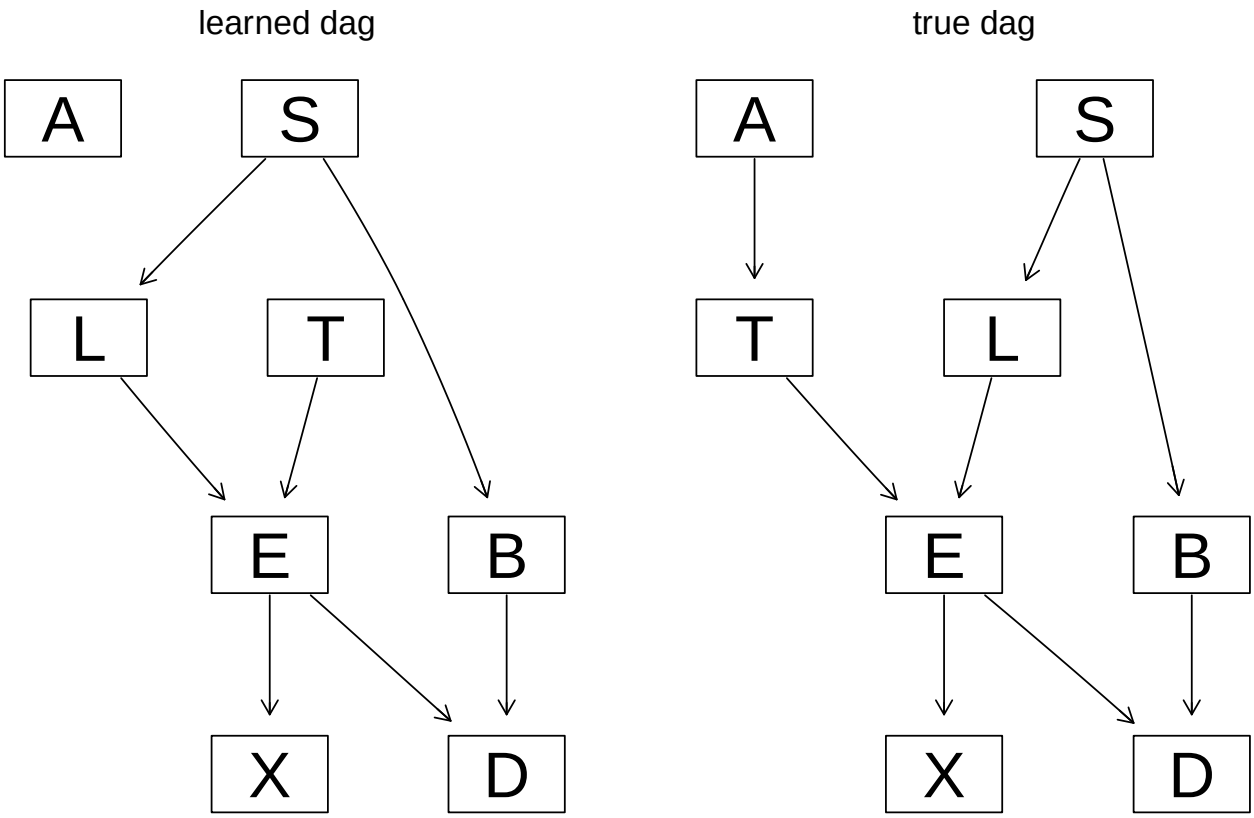
```r
table(Predicted_True = true_fitted_s, Actual = test_data$S)
```

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

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

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

```r
graphviz.plot(true_dag, main = "true dag")
```



The confusion matrices are very similar. Only missing from true is A->T in the dags.

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."

```
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
```

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

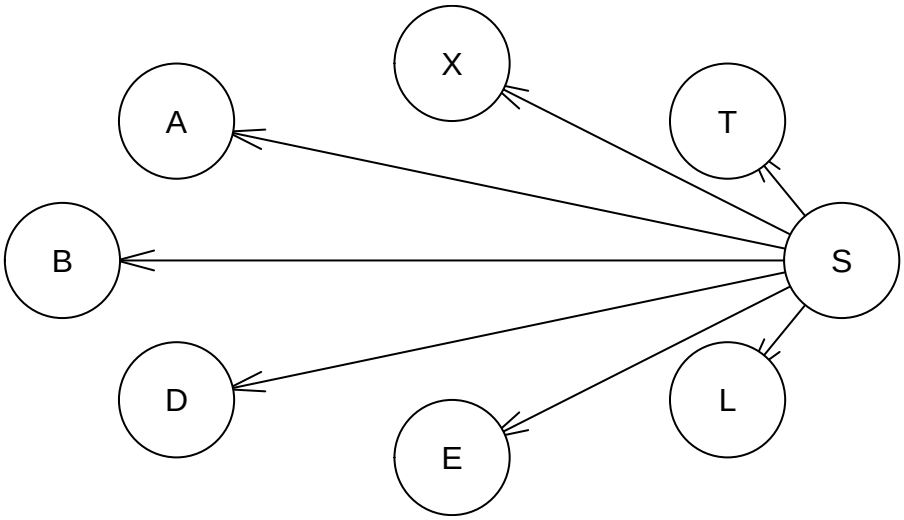
4

```
#Q4,
# Naive Bayes classifier
nb <- model2network("[S][A|S][B|S][D|S][E|S][L|S][T|S][X|S]")
nbc <- bn.fit(x = nb, data = train_data)

# Confusion matrix
table(predict_S_from_bn(bn = nbc, data = test_data, condition = condition), test_data$S)
```

```
##
```

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

```
plot(nb)
```



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

The Markov blanket approach and the full Bayesian network yielded similar results because both rely on a more fitted structure of dependencies in the network, with the Markov blanket focusing on the minimal sufficient set of variables 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.