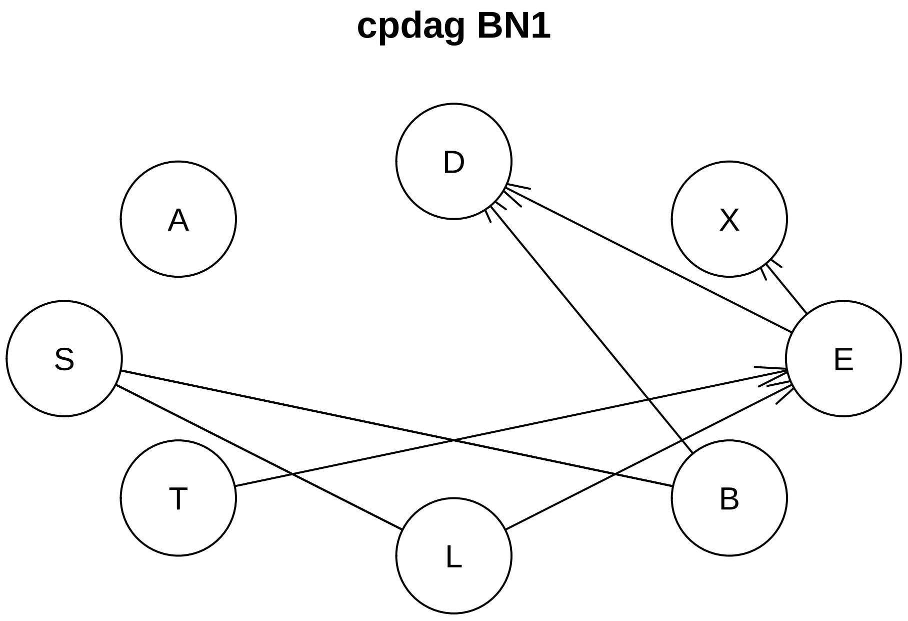
Group Report - Lab 1

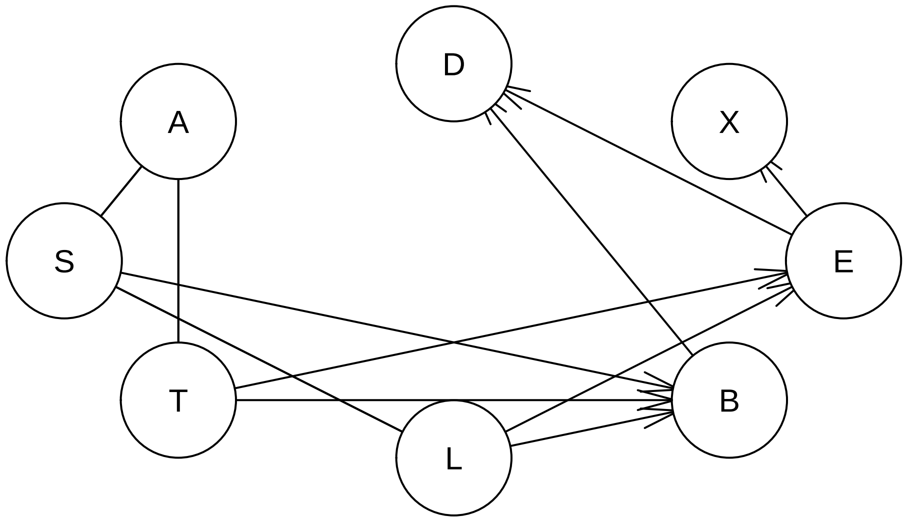
axeho681, wilha401, davha130 and guswa204

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 included in the bnlearn package. To load the data, run data("asia").



cpdag BN2



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

As one can see in the print statement above, as well as in the plots, the hill-climbing algorithm can return non-equivalent BN-structures. The reason for this is mainly the score evaluation method as that changes how the hill-climbing method is made. In addtion, the restarts indicates how many times the algorithm restarts, which also contributes to how the BN ends up at the end of the algorithm as that can prevent the algorithm from getting stuck in "bad" local optimums.

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.

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

Above is the confusion matrix based on BN learned by the training data.

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

Above is the confusion matrix based on BN from the true Asia BN.

Both of the confusion matrices are the same, which can be attributed to that both of the networks has the same connections to node "S", so "S" is evaluated the same in both of the networks.

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.

```
## [1] "Markow blanket for learned structure"
```

```
## [1] "B" "L"
```

```
## [1] "Markow blanket for true structure"
```

```
## [1] "B" "L"
```

```
## [1] "Confusion matrix using learned structure"
```

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

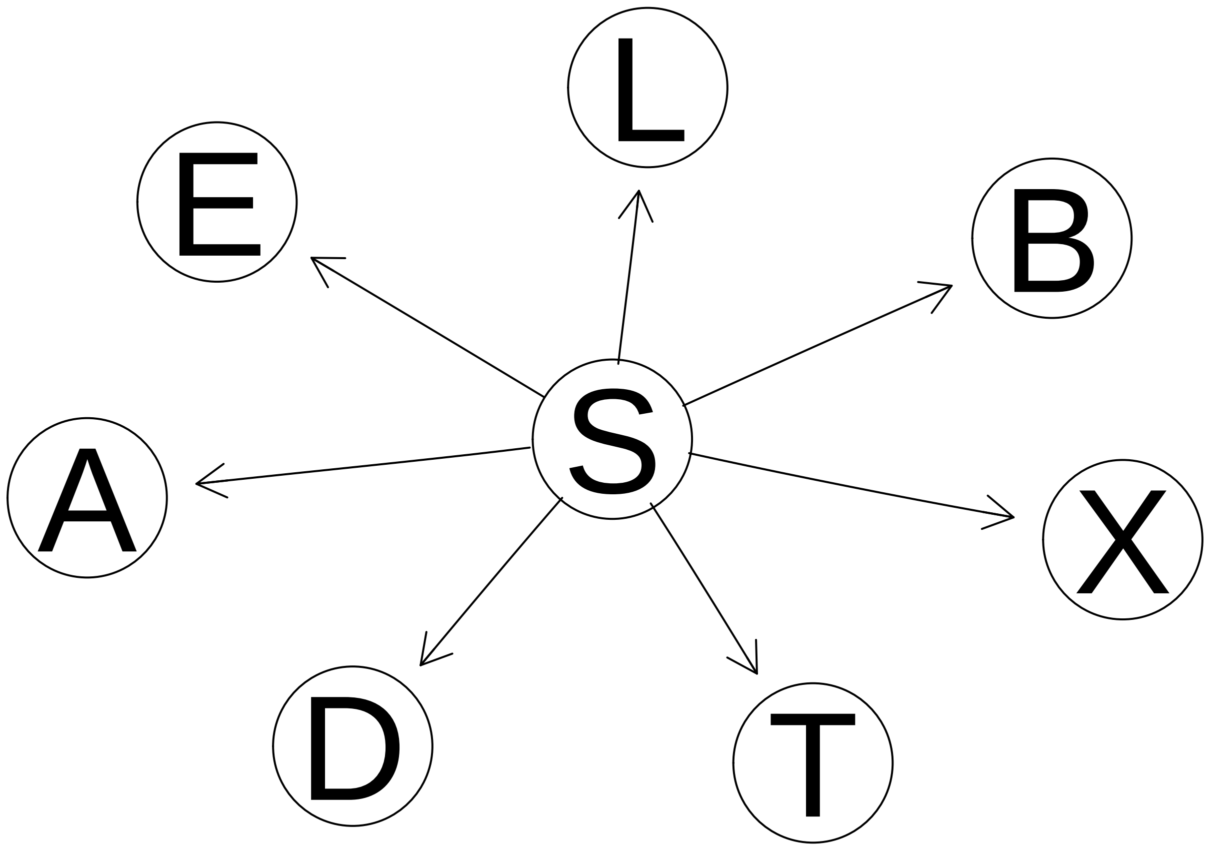
```
## [1] "Confusion matrix using true structure"
```

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

As one can observe, the two confusion matrices are the same when the Markow blanket for each sturcture was used.

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.



```
## [1] "Confusion matrix for prediction with naive bayesian network."
```

```
##             True labels
## Predictions  no yes
##         no  359 180
##         yes 154 307
```

```
## [1] "Confusion matrix for prediction with true network structure for the Asia dataset."
```

```
##             True labels
## Predictions  no yes
##         no  337 121
##         yes 176 366
```

Task 5

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

The reason for the confusion matrices being the same in task 2 and task 3 is because the dependence of "S" is the same in both cases, which is "L" and "B" (which is the Markow blanket for S). With knowledge about the state of L and B we have full information and therefore the result is the same when doing prediction for the whole network and for the Markow blanket. This applies as we complete data.

The confusion matrix is different in task 4 because of completely different dependency where all the nodes are dependent on "S" instead. Considering Bayes Theorem this affects the construction of the posterior distribution of S. We know the true structure of the BN and that the posterior should only consist of the distributions of L, B and the prior of S. We can then understand that the predictions are different.

Appendix for code

```
#########  EXERCISE 1  #########
# HC algorithm with different iss values. The different iss values gives
# different weights to the prior to the score
set.seed(12345)
BN1 <- hc(data,
          score = "bic",
          restart = 10)
set.seed(12345)
BN2 <- hc(data,
          score = "aic",
          restart = 100)

cpdag1 <- cpdag(BN1)
cpdag2 <- cpdag(BN2)
plot(cpdag1, main = "cpdag BN1")

plot(cpdag2, main = "cpdag BN2")
print(all.equal(cpdag1, cpdag2))

## Task 2

set.seed(12345)
n <- dim(data)[1]
id <- sample(1:n, floor(n * 0.8))
train <- data[id,]
test <- data[-id,]

init <- empty.graph(c("A", "S", "T", "L", "B", "E", "X", "D"))
bn_structure <- hc(train, start = init, restart = 5)
bn_fit <- bn.fit(bn_structure, train, method = "bayes")

# True Asia BN
bn_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bn_true_fit <- bn.fit(bn_structure, train, method = "bayes")


#Converts the fit from above into a gRain-object
bn_grain <- as.grain(bn_fit)
bn_true_grain <- as.grain(bn_true_fit)

#Compiles the junction
j_tree <- compile(bn_grain)
j_tree_true <- compile(bn_true_grain)



prediction <- function(tree, data, obs, pred) {
  predictions <- c()

  for (i in 1:dim(data)[1]) {
    x_vals = c()
    for (j in obs) {
      x_vals[j] = if (data[i, j] == "yes")
        "yes"
      else
        "no"
    }


    evidence = setEvidence(tree, obs, x_vals)
    probability = querygrain(evidence, pred)$S["yes"]
    predictions[i] = if (probability >= 0.5)
      "yes"
    else
      "no"

  }
  return(predictions)
}

bn_pred <-
  prediction(j_tree, test, c("A", "T", "L", "B", "E", "X", "D"), c("S"))
bn_pred_true <-
  prediction(j_tree_true, test, c("A", "T", "L", "B", "E", "X", "D"), c("S"))

# Confusion matrices by comparing with original data
confusion_matrix_fit <- table(bn_pred, test$S)
confusion_matrix_fit
confusion_matrix_true <- table(bn_pred_true, test$S)
confusion_matrix_true

## Task 3

junction.tree.own <- j_tree
junction.tree.true <- j_tree_true

bn.structure <- bn_true
bn.true.structure <- bn_true

predictfunction <-
  function(junction.tree,
           data,
           target,
           bn.for.markowblanket = NULL) {
    predictions <- c()

    # Setting observation variables
    observ.nodes <- names(data)[-which(names(data) == target)]
    if (!is.null(bn.for.markowblanket)) {
      print("With Markow blanket")
      observ.nodes <- mb(bn.for.markowblanket, target)
    }

    # print(observ.nodes) # DEBUG
    for (r in 1:nrow(data)) {
      observ.nodes.state <- c()


      for (c in observ.nodes) {
        observ.nodes.state[c] = ifelse(data[r, c] == "yes", "yes", "no")
      }

      # print(observ.nodes.state)
      evidence = setEvidence(junction.tree, observ.nodes, observ.nodes.state)
      probability = querygrain(evidence, target)$S["yes"]
      #print(probability)
      predictions[r] = ifelse (probability >= 0.5, "yes", "no")

    }
    return(predictions)
  }


make.confusion.matrix <- function(true.values, predicted.values) {
  return(table(true = true.values, prediction = predicted.values))
}
# Making predictions using helper-function (separate file)
prediction.own <-
  predictfunction(junction.tree.own, test, "S", bn.for.markowblanket = bn.structure)
prediction.true <-
  predictfunction(junction.tree.true, test, "S", bn.for.markowblanket = bn.true.structure)
# Comparing predictions with true values
conf.own <- make.confusion.matrix(test[, "S"], prediction.own)
conf.true <- make.confusion.matrix(test[, "S"], prediction.true)


## Task 4

data("asia")

# Structuring data
train_set = train
test_set = test
test_labels = test_set["S"]
test_set = subset(test_set, select = -c(S))
test_columns = colnames(test_set)

bn_structure_learning <- function() {
  model_structure = empty.graph(c("A", "S", "T", "L", "B", "E", "X", "D"))
  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"))
  )
  arcs(model_structure) = arc.set
  #plot(bn_nodes)
```

```r
  return(model_structure)
}

bn_parameter_learning <- function(bn_structure, training_set) {
  # Parameter learning
  model = bn.fit(bn_structure, training_set)

  # Converting model to grain objects (avoiding NaN-values?)
  grained_model = as.grain(model)

  # Compiling model to an BN?
  compiled_model = compile(grained_model)

  return(compiled_model)
}

bn_precdict <- function(bn_model, testing_set, columns) {
  predictions = c()

  # Predictions
  for (i in 1:dim(testing_set)[1]) {
    # Collecting ith obs states in a vector
    states = NULL
    for (j in columns) {
      if (testing_set[i, j] == "yes") {
        states = c(states, "yes")
      }  else {
        states = c(states, "no")
      }
    }

    # Prediction of var "S" on the ith obs
    evidence = setEvidence(bn_model, nodes = columns, states = states)

    probabilities = querygrain(evidence, c("S"))

    if (probabilities$S["no"] >= 0.5) {
      predictions[i] = "no"
    } else {
      predictions[i] = "yes"
    }
  }
  return(predictions)
}

# Learning a BN from dataset
bn_naive_structure = bn_structure_learning()
graphviz.plot(bn_naive_structure, layout = "neato")
bn = bn_parameter_learning(bn_naive_structure, train_set)
preds = bn_precdict(bn, test_set, test_columns)

# Plotting result
confusion_matrix = table(preds, test_labels$S, dnn = c("Predictions", "True labels"))
```

```r
#plot(confusion_matrix)
print("Confusion matrix for prediction with naive bayesian network.")
confusion_matrix

# True BN network for asia dataset
true_asia_bn_struct = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_asia_bn = bn_parameter_learning(true_asia_bn_struct, train_set)
preds_asia = bn_precdict(true_asia_bn, test_set, test_columns)

# Plotting result, true BN
confusion_matrix_true_bn = table(preds_asia, test_labels$S, dnn = c("Predictions", "True labels"))
#plot(confusion_matrix_true_bn)
print("Confusion matrix for prediction with true network structure for the Asia dataset.")
confusion_matrix_true_bn
```