TDDE15 - Lab 1

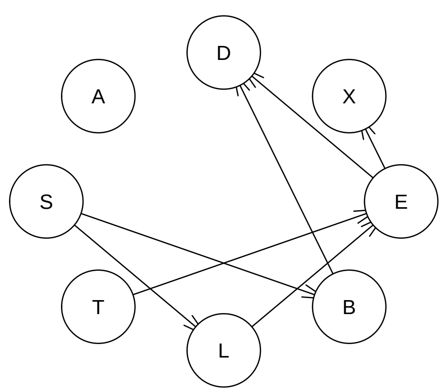
Question 1

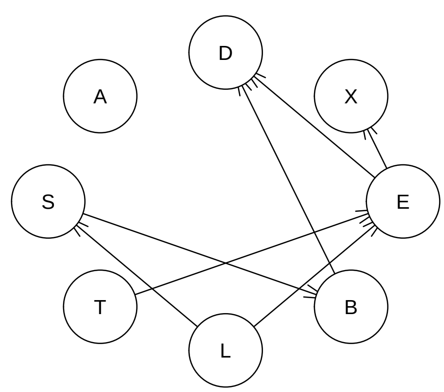
```
library(bnlearn)
data("asia")
structure = hc(asia, restart = 3) #start = initial structure, restart = random restarts, score = score,

b=T
for (i in 1:100) {
  structure2 = hc(asia, restart = 1)
  b = all.equal(structure, structure2)
  if (b!=TRUE) {
    print("Different network found")
    break
  }
}
```

```
## [1] "Different network found"
```

```
plot(structure)
plot(structure2)
```





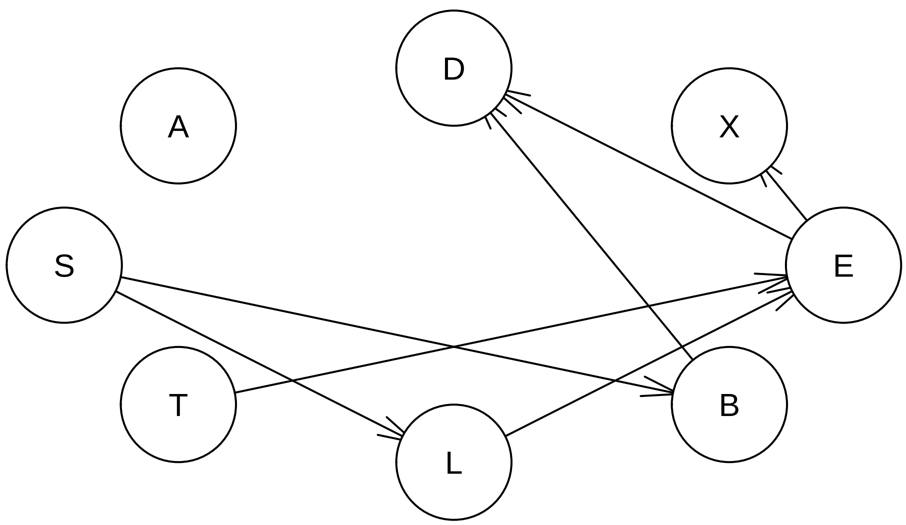
The HC return two different networks, because it can get trapped in a local optimum. HC gives you all independencies that exist in the true model. it does not give you any fals indenpendancies. In this case an edge is reversed, which would give the same score in the HC algorithm.

Question 2

```r
library(gRain)
N = dim(asia)[1]

#
train = asia[1:floor(N*0.8),]
test = asia[(floor(N*0.8)+1):N,]

structure = hc(train, restart = 0)
plot(structure)
```




```r
#Learn conditional probabiliies given the nodes parents
fit = bn.fit(structure, data=train)
#fit
#coefficients(fit)

# Create Graphical independance network ( grain object )
fit_grain = as.grain(fit)
#fit_grain

# create a junction tree and est. potential clique ( grain object )
# junc_tree = compile(fit_grain)
#print(junc_tree$cptlist)
```

```r
#remove S from test-data
test_ans = test[,"S"]
test_evid = subset(test, select = -2)

#Predict S
pred_s =c()
for (j in 1:dim(test_evid[1])) {

# finding/evidance or potentials
#need to extract observed values correctly.....
  obs = c()
  for (i in 1:7) {
    obs = c(obs, as.character(test_evid[j,i]))
  }

  nodes_ev = names(test_evid)
  evid = setEvidence(fit_grain, nodes_ev, states = obs)

  # quergrain to get conditional distributon
  node = c("S")
  prob_s = querygrain(evid, nodes = node)

  if (prob_s$S[1]>prob_s$S[2]) {
    pred_s=c(pred_s,"no")
  }else{
    pred_s=c(pred_s,"yes")
  }
}

table = table(pred_s,test_ans)

#Correct DAG
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
#Learn conditional probabiliies given the nodes parents
fit = bn.fit(dag, data=train)
#fit
#coefficients(fit)

# Create Graphical independance network ( grain object )
fit_grain = as.grain(fit)
#fit_grain

# create a junction tree and est. potential clique ( grain object )
# junc_tree = compile(fit_grain)
#print(junc_tree$cptlist)

#remove S from test-data
test_ans = test[,"S"]
test_evid = subset(test, select = -2)

#Predict S
pred_s =c()
for (j in 1:dim(test_evid[1])) {
```

```
# finding/evidance or potentials
#need to extract observed values correctly.....
  obs = c()
  for (i in 1:7) {
    obs = c(obs, as.character(test_evid[j,i]))
  }

  nodes_ev = names(test_evid)
  evid = setEvidence(fit_grain, nodes_ev, states = obs)
  #pEvidence(evid)

  # quergrain to get conditional distributon
  node = c("S")
  prob_s = querygrain(evid, nodes = node)

  if (prob_s$S[1]>prob_s$S[2]) {
    pred_s=c(pred_s,"no")
  }else{
    pred_s=c(pred_s,"yes")
  }
}

corr_table = table(pred_s,test_ans)
table
```

```
##        test_ans
## pred_s  no yes
##    no  358 120
##    yes 147 375
```

```
corr_table
```

```
##        test_ans
## pred_s  no yes
##    no  358 120
##    yes 147 375
```

We get the same classification from the correct network. Thats because S is independend given the marcov blanket ("B" & "L"). The subgraph containing S, B and L is learned correctly.

Question 3

```
marc_blanc = mb(fit, node = c("S" ))
marc_blanc
```

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

```r
test_evid = subset(test_evid, select=marc_blanc)

pred_s =c()
for (j in 1:dim(test_evid[1])) {

# finding/evidance or potentials
#need to extract observed values correctly.....
  obs = c()
  for (i in 1:dim(test_evid)[2]) {
    obs = c(obs, as.character(test_evid[j,i]))
  }

  nodes_ev = names(test_evid)
  evid = setEvidence(fit_grain, nodes_ev, states = obs)
  #pEvidence(evid)

  # quergrain to get conditional distributon
  node = c("S")
  prob_s = querygrain(evid, nodes = node)

  if (prob_s$S[1]>prob_s$S[2]) {
    pred_s=c(pred_s,"no")
  }else{
    pred_s=c(pred_s,"yes")
  }
}

marcov_table = table(pred_s,test_ans)
table
```

```
##        test_ans
## pred_s  no yes
##    no  358 120
##    yes 147 375
```

```r
marcov_table
```

```
##        test_ans
## pred_s  no yes
##    no  358 120
##    yes 147 375
```

Question 4

```r
N = dim(asia)[1]
train = asia[1:floor(N*0.8),]
test = asia[(floor(N*0.8)+1):N,]
test_ans = test[,"S"]
test_evid = subset(test, select = -2)

#Crating an empty network
```

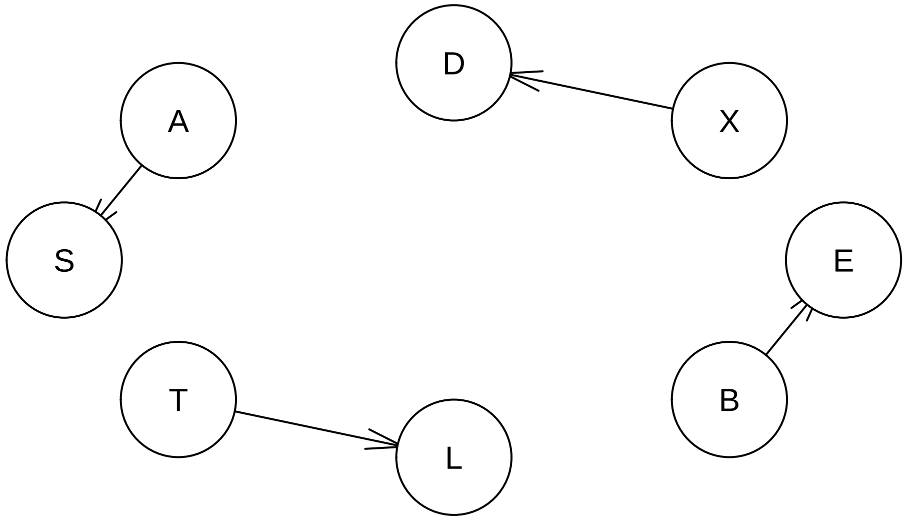
```r
library(bnlearn)
b_net = empty.graph(names(asia))
b_net
```

```
##
##   Random/Generated Bayesian network
##
##   model:
##    [A][S][T][L][B][E][X][D]
##   nodes:                                 8
##   arcs:                                  0
##     undirected arcs:                     0
##     directed arcs:                       0
##   average markov blanket size:           0.00
##   average neighbourhood size:            0.00
##   average branching factor:              0.00
##
##   generation algorithm:                  Empty
```

```r
class(b_net)
```

```
## [1] "bn"
```

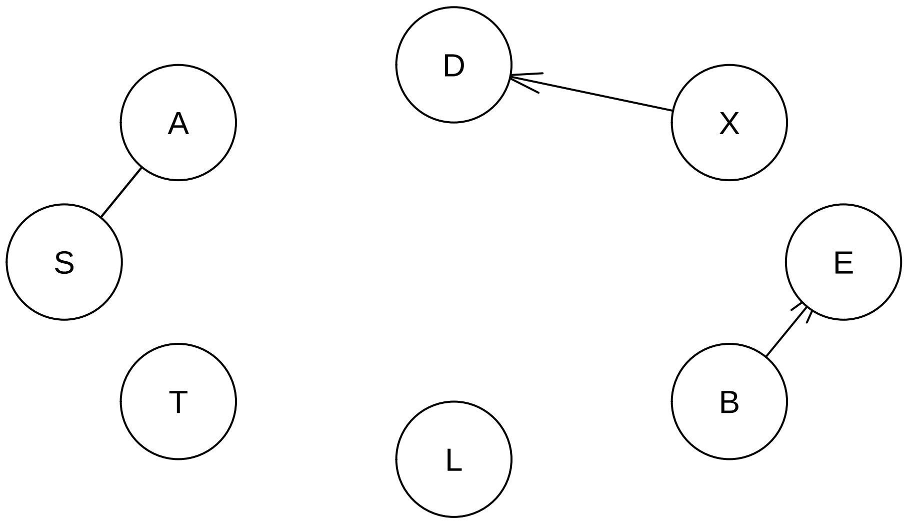
```r
#Createing directed edges
arc_set = matrix(names(asia),
                 ncol = 2, byrow = TRUE,
                 dimnames = list(NULL, c("from", "to")))
arcs(b_net) = arc_set
plot(b_net)
```




```
#Creating an undirected edge
a=names(asia)
a[3] = "S"
a[4] = "A"
arc_set = matrix(a,
                 ncol = 2, byrow = TRUE,
                 dimnames = list(NULL, c("from", "to")))
arc_set
```

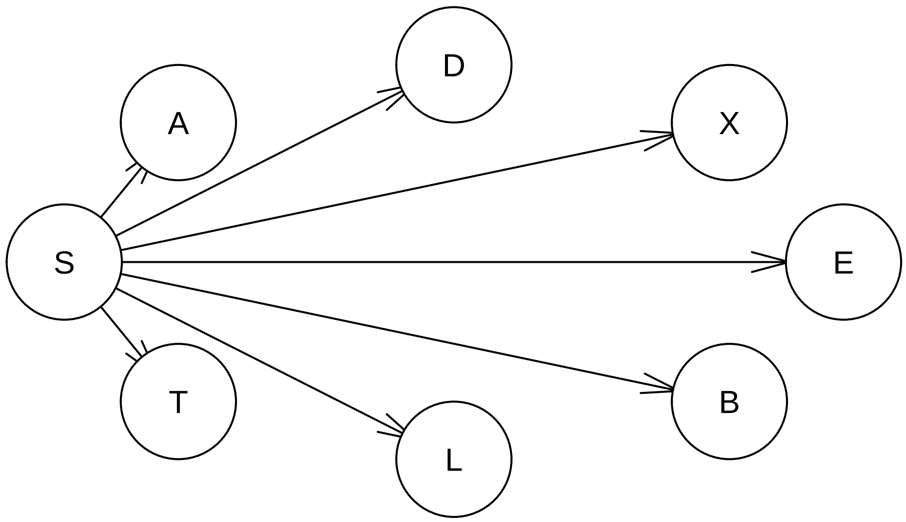
```
##      from to
## [1,] "A"  "S"
## [2,] "S"  "A"
## [3,] "B"  "E"
## [4,] "X"  "D"
```

```
arcs(b_net) = arc_set
plot(b_net)
```




```
# Adjacncy matrix (0L ensures that the number is stored as an integer instead of a double)
adj = matrix(0L, ncol = 8, nrow = 8,
             dimnames = list(names(asia), names(asia)))
amat(b_net) = adj

# Add edges in BN
for (i in names(test_evid)) {
  adj["S",i] = 1L
}
amat(b_net) = adj
plot(b_net)
```




```
bn_pot = bn.fit(b_net, data=train)
bn_grain = as.grain(bn_pot)

pred_s =c()
for (j in 1:dim(test_evid[1])) {
# finding/evidance or potentials
  obs = c()
  for (i in 1:dim(test_evid)[2]) {
    obs = c(obs, as.character(test_evid[j,i]))
  }

  nodes_ev = names(test_evid)
  evid = setEvidence(bn_grain, nodes_ev, states = obs)

  # quergrain to get conditional distributon
  node = c("S")
  prob_s = querygrain(evid, nodes = node)

  if (prob_s$S[1]>prob_s$S[2]) {
    pred_s=c(pred_s,"no")
  }else{
    pred_s=c(pred_s,"yes")
  }
}
naive_table = table(pred_s,test_ans)
table
```

```
##       test_ans
## pred_s  no yes
##    no  358 120
##    yes 147 375
```

```
naive_table
```

```
##       test_ans
## pred_s  no yes
##    no  389 180
##    yes 116 315
```