Advanced Machine Learning

Lab 1

Rasmus Holm

2017-09-10

Assignment 1

```
true_net <- empty.graph(names(learning.test))
modelstring(true_net) = "[A][C][F][B|A][D|A:C][E|B:F]"
```

```
graph_equality <- function(g1, g2) {
    all.equal(cpdag(g1), cpdag(g2)) == TRUE
}

print("No restarts")
#> [1] "No restarts"
sapply(1:10, function(i) {
    g1 <- hc(alarm, score="bde", iss=1, restart=0)
    g2 <- hc(alarm, score="bde", iss=1, restart=0)
    graph_equality(g1, g2)
})
#>  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

print("Restarts with equivalent seed")
#> [1] "Restarts with equivalent seed"
sapply(1:10, function(i) {
    set.seed(i)
    g1 <- hc(alarm, score="bde", iss=1, restart=10)
    set.seed(i)
    g2 <- hc(alarm, score="bde", iss=1, restart=10)
    graph_equality(g1, g2)
})
#>  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

print("Restarts with distinct seed")
#> [1] "Restarts with distinct seed"
sapply(1:10, function(i) {
    set.seed(i)
    g1 <- hc(alarm, score="bde", iss=1, restart=10)
    set.seed(2 * i)
    g2 <- hc(alarm, score="bde", iss=1, restart=10)
    graph_equality(g1, g2)
})
#>  [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
```

The hill climbing algorithm is completely deterministic so unless we do random restarts the result is the same. The random restarts means that we begin the algorithm with different initial graph structures which may possibly end up in different local optimums.

Assignment 2

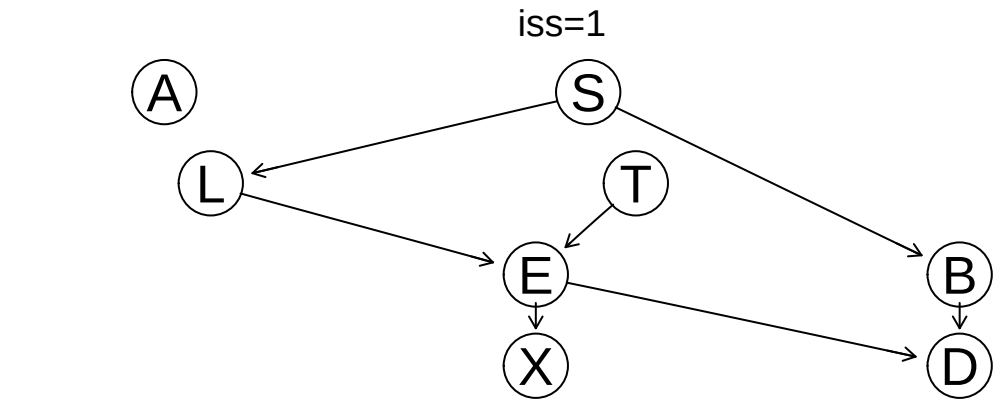
```
data <- asia

g1 <- hc(data, score="bde", iss=1, restart=10)
g10 <- hc(data, score="bde", iss=10, restart=10)
g100 <- hc(data, score="bde", iss=100, restart=10)
g1000 <- hc(data, score="bde", iss=1000, restart=10)

bnlearn::score(g1, data, type="bde")
#> [1] -11095.79
bnlearn::score(g10, data, type="bde")
#> [1] -11132.15
bnlearn::score(g100, data, type="bde")
#> [1] -11467.13
bnlearn::score(g1000, data, type="bde")
#> [1] -13301.65
```

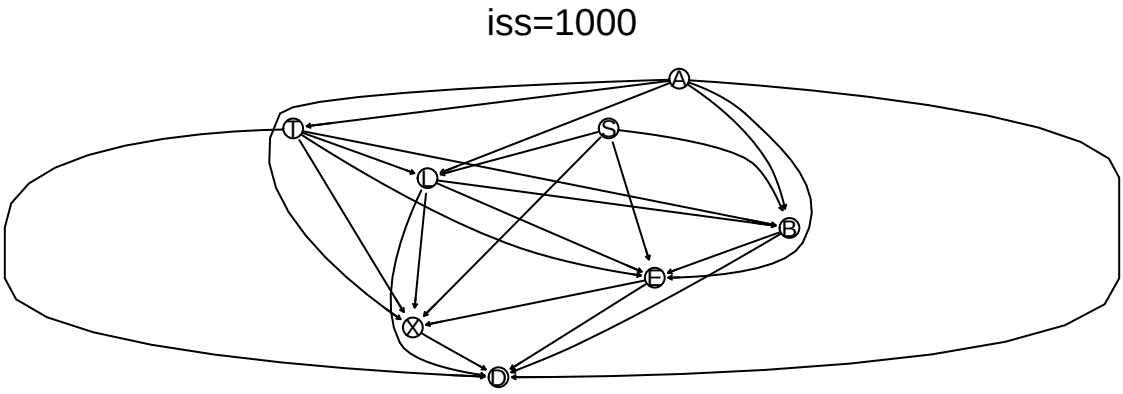
In the BDe (*Bayesian Dirichlet equivalent uniform*) score we assume a-priori that the probabilities follow a Dirichlet(α) where α is the *imaginary sample size* (iss) and the posterior

$$P(A|Data) = \frac{iss}{n+iss}P_{\text{prior}}(A) + \frac{n}{n+iss}P_{\text{empirical}}(A),$$

where n is the number of observations in the data. This means that the iss controls how certain we are a-priori of the probabilities indicating that having a high iss means the data have less influence on the posterior distribution. Since the Dirichlet is chosen such that it is uniform a higher iss result in more edges in the graph, therefore iss can be seen as a regularization term. A low value entails a sparse graph, i.e. less arcs/dependencies, inferred from data.

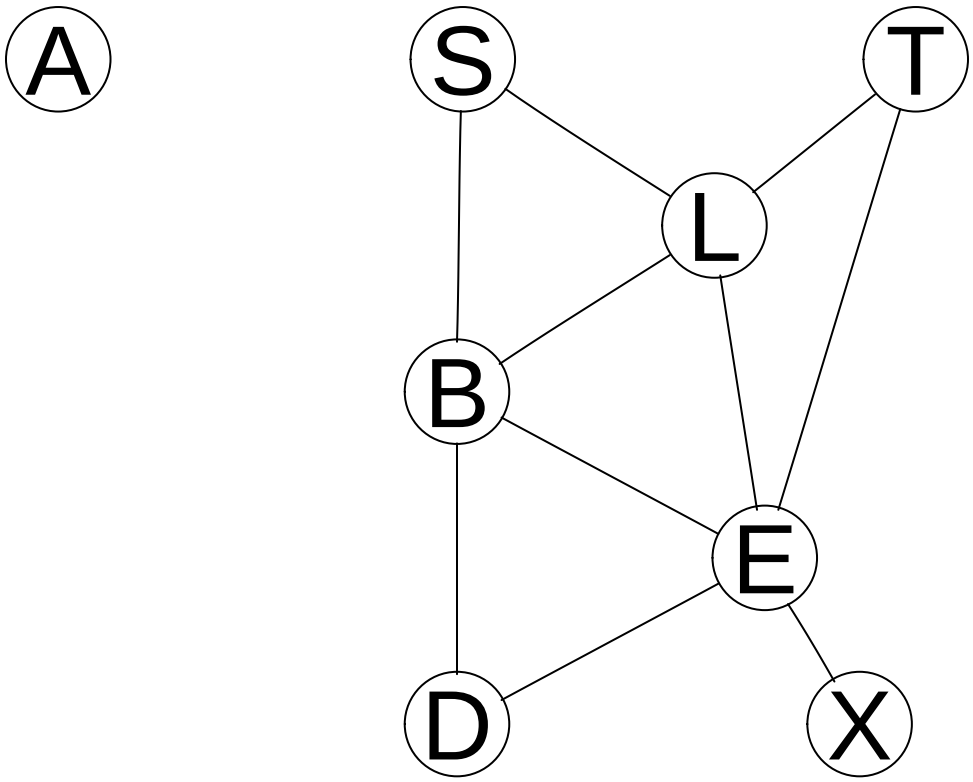
This can clearly be shown by the plot below. Given that there are 8 nodes in the graph we would expect the average branching factor to be around $7/2 = 3.5$ and we get 3 with a imaginary sample size of 1000 which is close and Bayesian networks also have certain constraints that prevent edges from being added arbitrarly.





Assignment 3

```
data <- asia
graph <- hc(data, score="bde", iss=1, restart=10)
bayes_net <- bn.fit(graph, data, method="bayes", iss=1)
junction_tree <- compile(as.grain(bayes_net))
```




```
## Exact inference
querygrain(junction_tree, nodes=c("B"), type="marginal")
#> $B
#> B
#>       no      yes
#> 0.490202 0.509798

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("B"), evidence=TRUE)
prop.table(table(dist))
#> dist
#>     no    yes
#> 0.4888 0.5112

dist <- cpdist(fitted=bayes_net, nodes=c("B"), evidence=TRUE, method="lw")
prop.table(table(dist))
#> dist
#>     no    yes
#> 0.4884 0.5116

## Exact inference
querygrain(junction_tree, nodes=c("L", "T"), type="joint")
#>      T
#> L             no          yes
#>   no  0.92560305 0.0083101656
#>   yes 0.06549873 0.0005880548

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=TRUE)
```

```
prop.table(table(dist))
#>      T
#> L         no    yes
#>   no  0.9224 0.0076
#>   yes 0.0694 0.0006

dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=TRUE, method="lw")
prop.table(table(dist))
#>      T
#> L         no    yes
#>   no  0.9230 0.0064
#>   yes 0.0704 0.0002

## Exact Inference
querygrain(setEvidence(junction_tree, nodes="E", states="yes"),
           nodes=c("L", "T"), type="joint")
#>      T
#> L               no         yes
#>   no  0.0003360683 0.111418174
#>   yes 0.8805694921 0.007676265

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=(E == "yes"))
prop.table(table(dist))
#>      T
#> L            no       yes
#>   no  0.0000000 0.1336898
#>   yes 0.8663102 0.0000000
```

In the approximate inference methods we use simulated data to infer the queries and the samples are random so the inference will also be random.

Assignment 4

```
n <- 1000
rgraphs <- random.graph(nodes=c("1", "2", "3", "4", "5"), num=n,
                        method="ic-dag", burn.in=500, every=50)
length(unique(lapply(rgraphs, cpdag))) / n
#> [1] 0.864
```