LAB 1: GRAPHICAL MODELS

xiali125@student.liu.se linfr259@student.liu.se qinzh916@student.liu.se
huali824@student.liu.se

2025-09-14

Contents

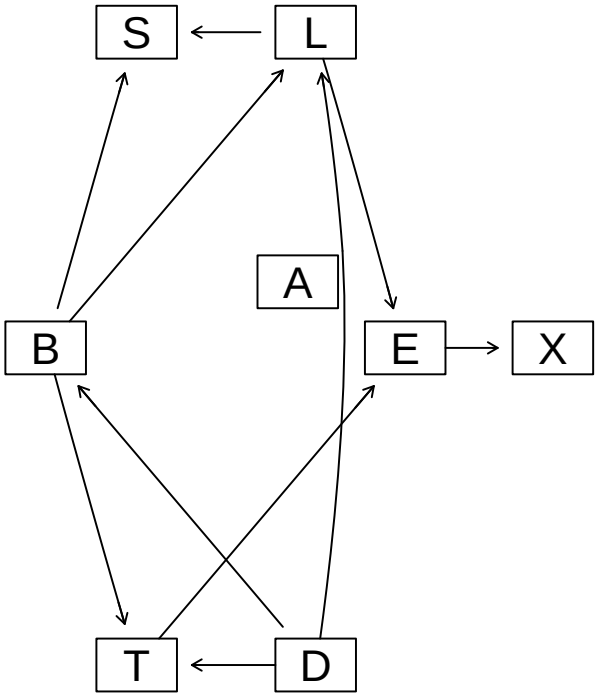
Hill-Climbing Algorithm and Non-Equivalent BN Structures

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"). Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning.

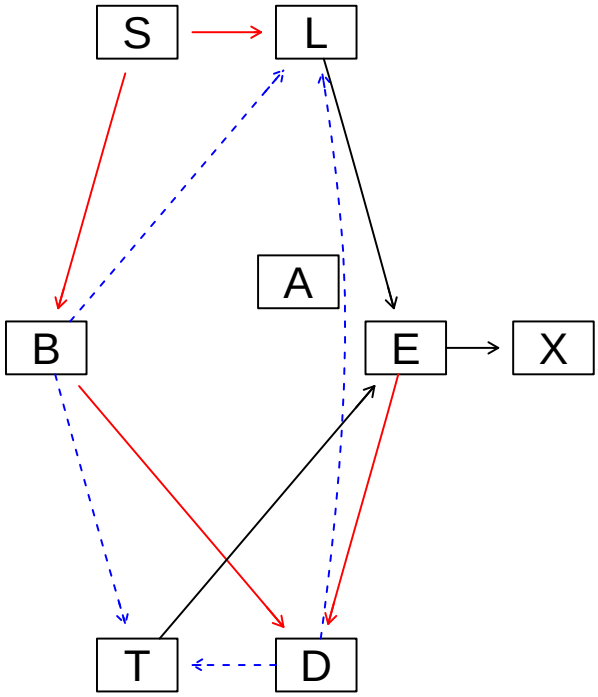
Different the hill-climbing (hc) algorithm have been constructed:

- with and without an initial graph,
- different score methods: Bayesian Dirichlet equivalent score(BDeu) and Bayesian Information Criterion (BIC) score,
- different imaginary sample size(iss) values for BDeu score, which represents the strength of the prior belief.

With initial start

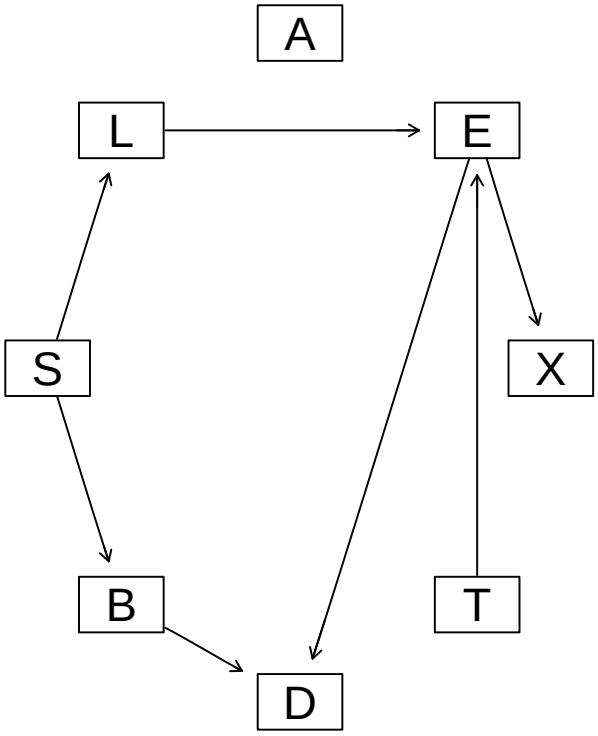


Without initial start

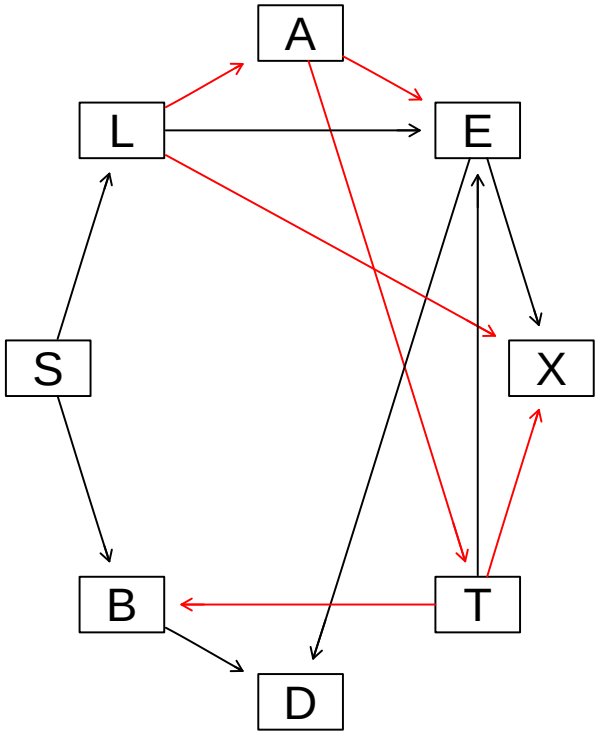


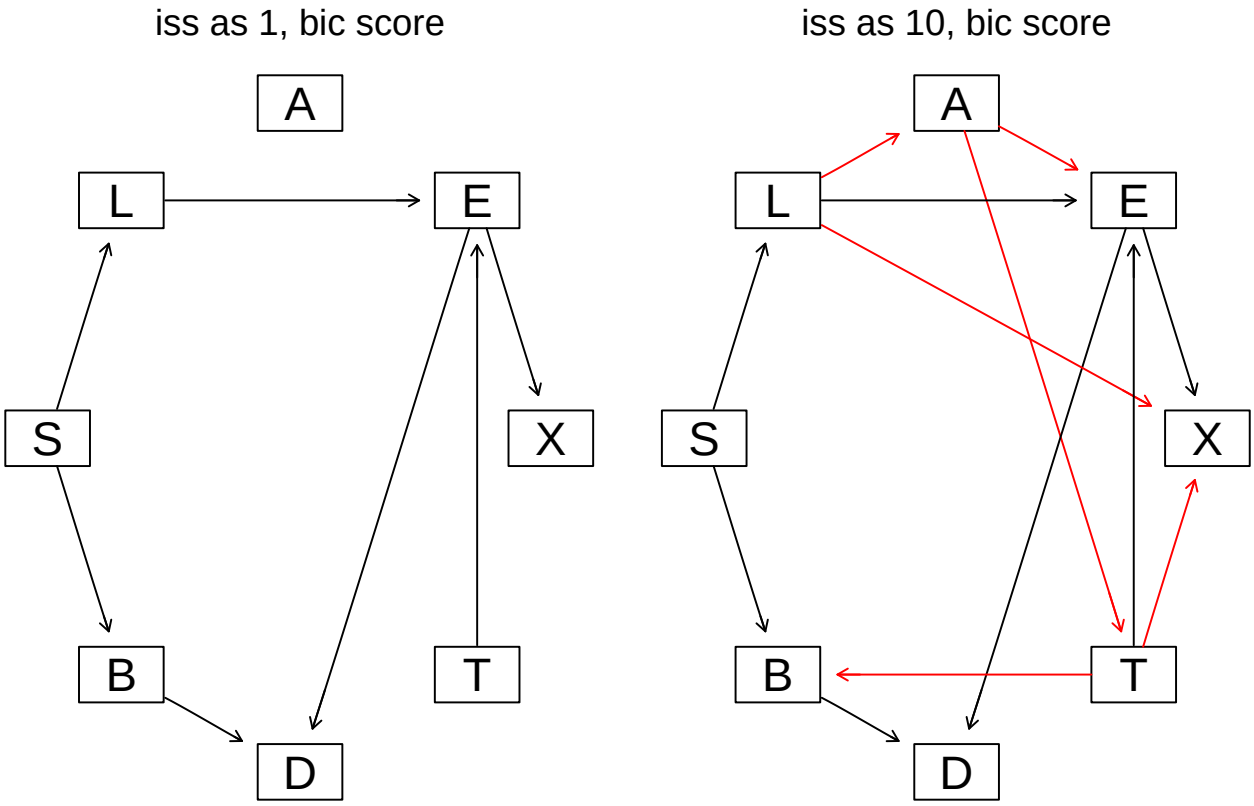

```
## [1] "Different number of directed/undirected arcs"
```

With bic score



with bde score





It is found that different methods of structure learning can yield different results, leading to non-equivalent BN structures, see the. This is because Hill-climbing is a greedy local search: starting from some initial graph it makes only score-improving, single-edge changes until it hits a local maximum. With the Asia data there are many near-optimal peaks because the sample is small and several edge orientations score almost the same; different starts, tie-breaking, or random initial structures can therefore lead the search to different peaks. When two peaks differ in the skeleton or in the set of v-structures, their CPDAGs are not equal, so the learned DAGs are non-equivalent.

By comparing the results In the example where differnet socres are used, it is shown that the network learned with iss= 1 is much simpler (sparser) than the one learned with iss = 10. This happens because the iss (imaginary sample size) parameter in the BDeu score acts as a regularizer, controlling the penalty for model complexity. A high iss weakens this penalty, allowing the algorithm to learn a more complex structure to better fit the data.

Classification with a Learned BN and True BN

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]")

Exact Inference with the Learned BN

The following code chunk performs the first part of the analysis. It splits the data, learns the structure and parameters of a Bayesian Network from the training set, and then uses exact inference with the gRain package to classify the test data. The bn1 object is first created by learning the BN structure using the hc (Hill-climbing) algorithm.

```
##      pred
##        no yes
##   no  359 141
##   yes 116 384
```

```
##       from to
##  [1,] "B"  "T"
##  [2,] "E"  "X"
##  [3,] "D"  "B"
##  [4,] "L"  "E"
##  [5,] "B"  "S"
##  [6,] "T"  "E"
##  [7,] "D"  "L"
##  [8,] "B"  "L"
##  [9,] "L"  "S"
## [10,] "D"  "T"
```

Exact Inference with the True BN

This second code chunk performs the same classification task, but using the known "true" BN structure for the Asia dataset. This allows for a direct comparison of the classification performance between our learned model and the actual model that generated the data. The parameters are still fit to the training data to ensure a fair comparison.

```
##      pred_true
##        no yes
##   no  359 141
##   yes 116 384
```

```
##      from to
## [1,] "A"  "T"
## [2,] "S"  "L"
## [3,] "S"  "B"
## [4,] "B"  "D"
## [5,] "E"  "D"
## [6,] "T"  "E"
## [7,] "L"  "E"
## [8,] "E"  "X"
```

Despite the learnt model learned a different structure (where S is a child of L and B) than the true model (where S is a parent), both BN fit on the same training data produces the same confusion matrix and accuracy on this test set.

Classification with the True BN's Markov Blanket

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.

We now perform the same classification task, but this time using the Markov blanket of S as defined by the true BN structure. This provides a direct comparison of the classification performance between the learned and true models when using only the most relevant variables for inference.

```
##      pred_mb
##        no yes
##   no  359 141
##   yes 116 384
```

```
## [1] 0.743
```

Classification with Naive Bayes

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.

The "naive" in Naive Bayes comes from the assumption that all predictive variables are independent given the class variable S. We will model this as a BN by hand, with S as the parent of all other variables.That is:

$$P(x_1, x_2, ..., x_n|y) = P(x_1|y) \cdot P(x_2|y) \cdots P(x_n|y)$$

Thus, Bayes' theorem becomes:

$$P(y|x_1, ..., x_n) = \frac{P(y) \cdot \prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

Since the denominator is constant for a given input, we can write:

$$P(y|x_1, ..., x_n) \propto P(y) \cdot \prod_{i=1}^{n} P(x_i|y)$$

```
##      pred_nb
##        no yes
##   no  390 110
##   yes 179 321
```

```
## [1] 0.711
```

With the naive Bayes BN, the accuracy is lower than the one achieved by the learned/true Asia models. Naive Bayes assumes all predictors are independent given S, but in the Asia network several features are still related.

Compare Results and Discussion

For a node S, the Markov blanket consists of:

- the **parents** of S,
- the **children** of S,
- and the **parents of the children** of S (i.e., co-parents).

Formally, we have:

$$P(S \mid \text{all other variables}) = P(S \mid \text{Markov blanket of } S).$$

And:

$$S \perp\!\!\!\perp \text{rest of network} \setminus MB(S) \mid MB(S).$$

In exercise 2, the learnt BN and true BN gave the same accuracy and confusion matrix despite of different networks. This happens because the posterior of S is determined by its Markov blanket, which means that to compute the posterior P(S evidence), only the states of L and B are needed. Any other nodes in the network are conditionally independent of S given L and B. Since both models has correctly captured the relationships that define this Markov blanket, so even if the rest of the structure differs, the decisions for S on this split end up identical.

Exercies 2 and 3 gave same results, despite that the prediction in exercise 3 is only based on the Markov blanket of S. This is because the Markov blanket contains all the information needed to predict S, making other variables redundant. It can also be claimed that: A is independent of all non-blanket nodes, given its Markov blanket.

While a different result was obtained in exercise 4, where a Naive Bayes classifier was built. Since this classifier assumes independence among predictors given S, which does not represent the real dependencies in the data. This has led to different results compared to the more complex models in exercises 2 and 3 that account for dependencies among variables.

Contritbutions

This report is a group effort by Xiaochen, Linn, Qinxia, and Huaide. The individual sections were drafted as follows: Xiaochen (Section 1), Linn (Section 2), Qinxia (Section 3), and Huaide (Section 4). The final discussion is the result of our collaborative work.

Appendix

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

# Compute overall accuracy
acc <- function(tab) sum(diag(tab)) / sum(tab)
# Classify S for many cases using a compiled gRain model
```

```r
classify_S <- function(compiled_fit, data, evidence_vars) {
pred <- character(nrow(data))
p_yes <- numeric(nrow(data))
for (i in seq_len(nrow(data))) {
# Build evidence list for this case, convert all factor values to character
ev <- as.list(data[i, evidence_vars, drop = FALSE])
ev <- lapply(ev, as.character)
# Set evidence and query posterior
q <- querygrain(setEvidence(compiled_fit, evidence = ev), nodes = "S")$S
p_yes[i] <- q["yes"]
pred[i] <- names(which.max(q)) # class with highest posterior
}
list(pred = pred, p_yes = p_yes)
}

library(bnlearn)
data('asia')
set.seed(123)
# initialize a graph with empty arcs
bn1 = empty.graph( colnames(asia))

# acr set to be included in the initial graph
# set graph as A -> D, B -> T, E -> X
arc.set = matrix(c("A", "D", "B", "T", "E", "X"),ncol = 2, byrow = TRUE,
dimnames = list(NULL, c("from", "to")))

arcs(bn1) = arc.set

# learn the structure with and without initial graph
bn1 <- hc(asia, start = bn1)
bn2 <- hc(asia)

# plot the two graphs
par(mfrow = c(1, 2))
graphviz.compare(bn1, bn2,layout = "circo", main = c("With initial start", "Without initial start"))

# check if the two graphs are equivalent
print(all.equal(cpdag(bn1), cpdag(bn2)))


data("asia")

set.seed(123)
bn3 = hc(asia) # score bic

set.seed(456)
bn4 = hc(asia, score = "bde", iss = 10)

par(mfrow = c(1, 2))

graphviz.compare(bn3, bn4,layout = "circo", main = c("With bic score", "with bde score"))
```

```r
#BDeu is one of the default scoring functions
#Run 1: Use a smaller iss (which favors simpler models)
bn_run1 <- hc(asia, score = "bde", iss = 1)
#Run 2: Use a larger iss (which allows for more complex models)
bn_run2 <- hc(asia, score = "bde", iss = 10)

par(mfrow = c(1, 2))

graphviz.compare(bn_run1, bn_run2,layout = "circo", main = c("iss as 1, bic score", "iss as 10, bic sco
```

```r
### 2
# data split
library(gRain)
set.seed(123)
train_id = sample(nrow(asia), 0.8*nrow(asia))
train = asia[train_id, ]
test = asia[-train_id, ]

# fit the model and convert to grain object
bn_fit <- bn.fit(bn1, train)

grain_fit <-as.grain(bn_fit)

# compilation, so inference uses the junction tree → fast repeated queries.
compiled_fit <- compile(grain_fit)

# character(nrow(test)) to pre-allocate a character vector of the right length.
pred <- character(nrow(test))

for (i in 1:nrow(test)) {

  ev <- as.list(test[i,colnames(test)!= 'S']) # use all variables except S
  ev <- lapply(ev, as.character)# convert to character strings
  res <- querygrain(setEvidence(compiled_fit, evidence= ev), nodes = 'S')
  pred[i] <- names(which.max(res$S))# pick class with max posterior P(S|evidence)
}

print(table(test$S,pred))

print(arcs(bn1))

# predict the test data
real_model = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
real_fit = bn.fit(real_model, train)
grain_real = as.grain(real_fit)

# compilation, so inference uses the junction tree → fast repeated queries.
compiled_true <- compile(grain_real)

pred_true <- character(nrow(test)) # preallocate vector


for (i in 1:nrow(test)) {
```

```
  ev <- as.list(test[i,colnames(test)!= 'S']) # use all variables except S
  ev <- lapply(ev, as.character)  # convert to character strings
  res <- querygrain(setEvidence(compiled_true, evidence= ev), nodes = 'S')
  pred_true[i] <- names(which.max(res$S)) # pick class with max posterior P(S|evidence)
}

print(table(test$S,pred_true))
print(arcs(real_model))
```

```
### 3
# evidence = only Markov blanket of S

# get Markov bla evidence = only Markov blanket of Snket of S
mbS <- mb(bn1, 'S')

# character(nrow(test)) to pre-allocate a character vector of the right length.
pred_mb <- character(nrow(test))

for (i in 1:nrow(test)) {

  ev <- as.list(test[i,mbS]) # use all variables except S
  ev <- lapply(ev, as.character)# convert to character strings
  res <- querygrain(setEvidence(compiled_fit, evidence= ev), nodes = 'S')
  pred_mb[i] <- names(which.max(res$S))# pick class with max posterior P(S|evidence)
}

print(table(test$S,pred_mb))
print(acc(table(test$S,pred_mb)))
```

```
### 4
#    The "naive" in Naive Bayes comes from the assumption that all features are
#    independent given the class.
#    Asia variables: A, S, T, L, B, D, E, X
#    Model string: S -> A, S -> T, S -> L, S -> B, S -> D, S -> E, S -> X

nb_model <- model2network("[S][A|S][T|S][L|S][B|S][D|S][E|S][X|S]")
nb_fit<- bn.fit(nb_model, train)
grain_nb <- as.grain(nb_fit)
compiled_nb <- compile(grain_nb)
pred_nb <- character(nrow(test)) # preallocate vector

for (i in 1:nrow(test)) {

  ev <- as.list(test[i,colnames(test)!= 'S']) # use all variables except S
  ev <- lapply(ev, as.character)  # convert to character strings
  res <- querygrain(setEvidence(compiled_nb, evidence= ev), nodes = 'S')
  pred_nb[i] <- names(which.max(res$S)) # pick class with max posterior P(S|evidence)
}
print(table(test$S,pred_nb))
print(acc(table(test$S,pred_nb)))
```