Chapter 13

Bayesian Networks in R: Structure and Parameter Learning

13.1 Bayesian Networks with bnlearn

This tutorial is based predominantly on the bnlearn package, a package by Marco Scutari. It supports a wide variety of structure learning algorithms. These are found in the documentation. They are:

Constraint Based Algorithms

1. **Grow-Shrink gs**: based on the Grow-Shrink Markov Blanket, the first (and simplest) Markov blanket detection algorithm used in a structure learning algorithm.

2. **Incremental Association iamb**: based on the Markov blanket detection algorithm of the same name, which is based on a two-phase selection scheme (a forward selection followed by an attempt to remove false positives).

3. **Fast Incremental Association fast.iamb**: a variant of IAMB which uses speculative stepwise forward selection to reduce the number of conditional independence tests.

4. **Interleaved Incremental Association inter.iamb**: another variant of IAMB which uses forward stepwise selection to avoid false positives in the Markov blanket detection phase.

Search and Score Learning Algorithms

1. **Hill-Climbing hc**: a hill climbing greedy search on the space of the directed graphs. The optimised implementation uses score caching, score decomposability and score equivalence to reduce the number of duplicated tests.

2. **Tabu Search tabu**: a modified hill climbing able to escape local optima by selecting a network that minimally decreases the score function.
CHAPTER 13. BAYESIAN NETWORKS IN R: STRUCTURE AND PARAMETER LEARNING

Hybrid Learning Algorithms

1. **Max-Min Hill-Climbing mmhc**: a hybrid algorithm which combines the Max-Min Parents and Children algorithm (to restrict the search space) and the Hill-Climbing algorithm (to find the optimal network structure in the restricted space).

2. **Restricted Maximization rsm2**: a more general implementation of the Max-Min Hill-Climbing, which can use any combination of constraint-based and score-based algorithms.

Other (Constraint-Based) Learning Algorithms

These algorithms learn the structure of the undirected graph underlying the Bayesian network, which is known as the skeleton of the network or the (partial) correlation graph. Therefore all the arcs are undirected, and no attempt is made to detect their orientation. They are often used in hybrid learning algorithms.

1. **Max-Min Parents and Children mpc**: a forward selection technique for neighbourhood detection based on the maximization of the minimum association measure observed with any subset of the nodes selected in the previous iterations.

2. **Hiton Parents and Children si.hiton.pc**: a fast forward selection technique for neighbourhood detection designed to exclude nodes early based on the marginal association. The implementation follows the Semi-Interleaved variant of the algorithm.

3. **Chow-Liu chow.liu**: an application of the minimum-weight spanning tree and the information inequality. It learn the tree structure closest to the true one in the probability space.

4. **ARACNE aracne**: an improved version of the Chow-Liu algorithm that is able to learn poly-trees.

13.1.1 Creating and Manipulating Network Structures

The following illustrates how to create objects of class bn. We consider the marks data set, which gives the exam scores of 88 students across five different topics: mechanics, vectors, algebra, analysis and statistics. The original data set was investigated by Mardia et. al. (1979) [73] and subsequently became a benchmark for structure learning (e.g. Whittaker (1990) [118]. It is a data set within the bnlearn package under the name marks.

```r
> library(bnlearn)
> data(marks)
> str(marks)
'data.frame': 88 obs. of 5 variables:
$ MECH: num 77 63 75 55 63 53 51 59 62 64 ...
$ VECT: num 82 78 73 72 63 61 67 70 60 72 ...
```
First create an empty network with the nodes corresponding to the variables using the \texttt{empty.graph} function:

\begin{verbatim}
> ug<-empty.graph(names(marks))
\end{verbatim}

The arcs presented in Whittaker (1990) from Figure 13.1 may be added as follows:

\begin{verbatim}
> arcs(ug,ignore.cycles=TRUE)=matrix(
  + c("MECH","VECT","MECH","ALG","VECT","MECH",
  + "VECT","ALG","ALG","MECH","ALG","VECT",
  + "ALG","ANL","ALG","STAT","ANL","ALG",
  + "ANL","STAT","STAT","ALG","STAT","ANL"),
  + ncol=2, byrow=TRUE,
  + dimnames=list(c(),c("from","to")))
> plot(ug)
\end{verbatim}

![Figure 13.1: Marks network: undirected graph]

The resulting \texttt{ug} object belongs to graph \texttt{bn}. There are several arguments: \texttt{ug$learning}, \texttt{ug$nodes}, \texttt{ug$arcs}.

\texttt{learning} is not useful in this example, since this argument gives information about the results of the structure learning algorithm used to generate the network and its tuning parameters (which were not used here).

\texttt{$nodes} gives information about the Markov blanket of each node, while \texttt{$arcs} gives the arcs presented in the network.
> ug

Random/Generated Bayesian network

model:
  [undirected graph]

nodes: 5
arcs: 6
  undirected arcs: 6
directed arcs: 0
average markov blanket size: 2.40
average neighbourhood size: 2.40
average branching factor: 0.00

generation algorithm: Empty

> dag = empty.graph(names(marks))
> arcs(dag)=matrix(c("VECT","MECH","ALG","MECH","ALG","VECT",
+ "ANL","ALG","STAT","ALG","STAT","ANL"),
+ ncol=2,byrow=TRUE,
+ dimnames=list(c(),c("from","to")))
> dag

Random/Generated Bayesian network

model:
  [STAT] [ANL] [STAT] [ALG] [ANL:STAT] [VECT] [ALG] [MECH] [VECT:ALG]

nodes: 5
arcs: 6
  undirected arcs: 0
directed arcs: 6
average markov blanket size: 2.40
average neighbourhood size: 2.40
average branching factor: 1.20

generation algorithm: Empty

A dag can be specified by its adjacency matrix. The function all.equal() indicates whether two graphs are equal.

> mat=matrix(c(0,1,1,0,0,0,0,1,0,0,0,0,0),
+ 0,1,1,0,0,0,1,0,0,0,0,0),
+ nrow=5,
+ dimnames=list(nodes(dag),nodes(dag)))
> mat

<table>
<thead>
<tr>
<th></th>
<th>MECH</th>
<th>VECT</th>
<th>ALG</th>
<th>ANL</th>
<th>STAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MECH</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VECT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALG</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ANL</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STAT</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

> dag2=empty.graph(nodes(dag))
> amat(dag2)=mat
> all.equal(dag,dag2)
[1] TRUE

A new bn object may be created by adding (set.arc), dropping (drop.arc) or reversing rev.arc) arcs from the original. For example:

> dag3 = empty.graph(nodes(dag))
> dag3 = set.arc(dag3,"VECT","MECH")
> dag3 = set.arc(dag3,"ALG","MECH")

A topological ordering of the nodes (from ancestors to descendants) may be obtained by the function node.ordering(). The neighbours and Markov blanket may be found using nbr() and mb() respectively. The %in% command may be used to establish membership.

> node.ordering(dag)
[1] "STAT" "ANL" "ALG" "VECT" "MECH"
> nbr(dag,"ANL")
[1] "ALG" "STAT"
> mb(dag,"ANL")
[1] "ALG" "STAT"
> "ANL" %in% mb(dag,"ALG")
[1] TRUE

We can check that the Markov blanket of a variable consists of parents, children and children of parents:

> chld=children(dag,"VECT")
> par=parents(dag,"VECT")
> o.par=sapply(chld,parents,x=dag)
> unique(c(chld,par,o.par[o.par != "VECT"],"MECH")
[1] "MECH" "ALG"
> mb(dag,"VECT")
[1] "MECH" "ALG"
13.1.2 Visualising Graphical Models

The structures from \texttt{bnlearn} may be plotted using functions provided by \texttt{graph} and \texttt{Rgraphviz} packages (Gentry et. al. [45](2012)). The \texttt{graphviz.plot} function takes a \texttt{bn} object and returns the corresponding \texttt{graph} object.

In \texttt{bnlearn}, vee-structure refers to a collider connection.

\begin{verbatim}
> library(Rgraphviz)
Loading required package: grid
> h = list(arcs=vstructs(dag2,arcs=TRUE),lwd=4,col="black")
> graphviz.plot(dag2,h,ludge="fdp",main="dag2")
\end{verbatim}

The output is shown in Figure 13.2.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{dag2}
\caption{Plot obtained using \texttt{graphviz.plot}}
\end{figure}

The \textit{essential graph}, showing the Markov equivalence class is returned by \texttt{cpdag}. The function \texttt{moral} returns the moral graph.

\begin{verbatim}
> plot(cpdag(dag2))
\end{verbatim}

for example gives a plot of the essential graph corresponding to \texttt{dag2}.

13.1.3 Structure Learning

In \texttt{bnlearn}, the Maximum Minimum Parents Children (MMPC) algorithm is referred to as the \textit{grow-shrink} algorithm. The name is natural following the procedure; first the maximum parents / children set for each node is established and then unnecessary nodes are removed. This algorithm is implemented simply by the function \texttt{gs}.

\begin{verbatim}
> bn.gs <- gs(marks)
> bn.gs
\end{verbatim}
13.1. BAYESIAN NETWORKS WITH BNLEARN

Bayesian network learned via Constraint-based methods

```
model:
   [undirected graph]
nodes: 5
arcs: 6
   undirected arcs: 6
directed arcs: 0
average markov blanket size: 2.40
average neighbourhood size: 2.40
average branching factor: 0.00

learning algorithm: Grow-Shrink
conditional independence test: Pearson’s Correlation
alpha threshold: 0.05
tests used in the learning procedure: 44
optimized: TRUE
```

The parameter value $\alpha = 0.05$ is the nominal significance level for each $\chi^2$ test for independence. The `mmhc` algorithm learns a different network, but it is Markov equivalent to the network learned by the `gs` algorithm and has the same BIC score.

These structure learning algorithms often only direct an edge when a particular direction gives a better fit, leaving other edges undirected. The function `ceextend()` gets one graph out of the Markov equivalence class, which may be used for scoring purposes. The BIC score for the learned graph may be obtained as follows. The documentation lists other scoring criteria that are available (such as AIC).

```r
> bn.gsdirect <- ceextend(bn.gs)
> bn.gsdirect

Bayesian network learned via Constraint-based methods

model:
   [STAT] [ANL|STAT] [ALG|ANL:STAT] [VECT|ALG] [MECH|VECT:ALG]
nodes: 5
arcs: 6
   undirected arcs: 0
directed arcs: 6
average markov blanket size: 2.40
average neighbourhood size: 2.40
```
average branching factor: 1.20

learning algorithm: Grow-Shrink
conditional independence test: Pearson’s Correlation
alpha threshold: 0.05
tests used in the learning procedure: 44
optimized: TRUE

> score(bn.gsdirect, data=marks, type="bic-g")
[1] -1720.15

### 13.1.4 Parameter Learning

Having established the network, the next task is to learn the parameters. With `bnlearn`, this is performed by the `bn.fit` function.

```r
> fitted = bn.fit(bn.gsdirect, data=marks)
> fitted

Bayesian network parameters

Parameters of node MECH (Gaussian distribution)

Conditional density: MECH | VECT + ALG
Coefficients:
(Intercept) VECT ALG
-12.3647583 0.4658693 0.5484053
Standard deviation of the residuals: 13.97432

Parameters of node VECT (Gaussian distribution)

Conditional density: VECT | ALG
Coefficients:
(Intercept) ALG
12.4183094 0.7543653
Standard deviation of the residuals: 10.48167

Parameters of node ALG (Gaussian distribution)

Conditional density: ALG | ANL + STAT
Coefficients:
13.1. *BAYESIAN NETWORKS WITH BNLEARN*

\[
\begin{array}{ccc}
\text{(Intercept)} & \text{ANL} & \text{STAT} \\
24.7254768 & 0.3482454 & 0.2273881 \\
\end{array}
\]

Standard deviation of the residuals: 6.871428

Parameters of node ANL (Gaussian distribution)

Conditional density: ANL | STAT
Coefficients:
\[
\begin{array}{ccc}
\text{(Intercept)} & \text{STAT} \\
24.5824229 & 0.5223601 \\
\end{array}
\]

Standard deviation of the residuals: 11.86392

Parameters of node STAT (Gaussian distribution)

Conditional density: STAT
Coefficients:
\[
\begin{array}{c}
\text{(Intercept)} \\
42.30682 \\
\end{array}
\]

Standard deviation of the residuals: 17.25559

The type of estimator (maximum likelihood or Bayes) can be specified by either `mle` (maximum likelihood estimates) or `Bayes` the posterior Bayesian estimate arising from a flat, non-informative prior. Only `mle` is available with continuous (Gaussian) data; the `Bayes` considers Dirichlet densities over the parameter space.

The parameters of a fitted network can easily be replaced. For example, ALG has two parents, ANL and STAT. For the Gaussian network, the restriction is that the standard deviation for the residuals at each node is the same. We consider

\[\text{ALG} = \beta_0 + \text{ANL}\beta_1 + \text{STAT}\beta_2 + \epsilon\]

where \(\epsilon \sim \mathcal{N}(0,\sigma^2)\), independent identically distributed. This is carried out by:

```r
> fitted$ALG = list(coef=c("(Intercept)"=25, "ANL"=0.5, "STAT"=0.25), sd=6.5)
> fitted$ALG
```

Parameters of node ALG (Gaussian distribution)

Conditional density: ALG | ANL + STAT
Coefficients:
\[
\begin{array}{ccc}
\text{(Intercept)} & \text{ANL} & \text{STAT} \\
25.00 & 0.50 & 0.25 \\
\end{array}
\]

Standard deviation of the residuals: 6.5
A `bn.fit` object can be created from scratch using the `custom.fit` function. For example:

```r
> MECH.par = list(coef=c("(Intercept)"=-10, "VECT"=0.5, "ALG"=0.6),sd = 13)
> VECT.par = list(coef=c("(Intercept)"=10, "ALG"=1),sd=10)
> ALG.par=list(coef=c("(Intercept)"=25,"ANL"=0.5,"STAT"=0.25),sd=6.5)
> ANL.par=list(coef=c("(Intercept)"=25,"STAT"=0.5),sd=12)
> STAT.par=list(coef=c("(Intercept)"=43),sd=17)
> dist=list(MECH=MECH.par,VECT=VECT.par,ALG=ALG.par,ANL=ANL.par,STAT=STAT.par)
> fitted2 = custom.fit(bn.gsdirect,dist=dist)
```

### 13.1.5 Discretisation

The only continuous models that can be accommodated are Gaussian. When the data is manifestly not Gaussian, it is better to discretise it and to construct a Bayesian network over multinomial variables. There are several methods of discretisation available; look up the documentation for `discretize`. For example:

```r
> ?discretize
> dmarks = discretize(marks, breaks=2, method="quantile")
> bn.dgs=gs(dmarks)
> plot(bn.dgs)
> all.equal(cpdag(bn.dgs),cpdag(bn.gsdirect))
[1] "Different number of directed/undirected arcs"
```

The network learned from the discretised data is different; `MECH` is independent of the other variables.

The parameters may be fitted to the structure using the discretised data:

```r
> fitted3=bn.fit(cextend(bn.dgs),data=dmarks)
> fitted3$ALG
```

Parameters of node `ALG` (multinomial distribution)

Conditional probability table:

```latex
\begin{array}{c|c|c}
\text{ANL} & [9,49] & (49,70] \\
\text{ALG} & \{15,50] & 0.7777778 & 0.2558140 \\
& (50,80] & 0.2222222 & 0.7441860 \\
\end{array}
```

### 13.1.6 Latent Variables

Probability distributions often fail to have a faithful graphical representation because there are latent (or hidden) variables missing from the model.
For the marks data, Edwards (2000) \cite{edwards2000} assumed that the students fell into two distinct groups (which we call $A$ and $B$). He then used a classification technique involving the EM algorithm to assign the students to two different classes. The results were as follows: group $A$ contained students 1-44 and 46-52 while group $B$ contained students 45 and 53 - 88. We add this latent variable and we construct a network for group $A$ and another network for group $B$. We then discretize the variables and learn the network when the latent variable is included. The results are:

```r
> latent=factor(c(rep("A",44),"B",rep("A",7),rep("B",36)))
> bn.A = hc(marks[latent=="A",])
> bn.B = hc(marks[latent=="B",])
> modelstring(bn.A)
> modelstring(bn.B)
[1] "[MECH][ALG][ANL][STAT][VECT:MECH]"
> dmarks=discretize(marks,breaks=2,method="interval")
> dmarks2=cbind(dmarks,LAT=latent)
> bn.LAT=hc(dmarks2)
> bn.LAT
```

Bayesian network learned via Score-based methods

```
model:
  [MECH][ANL][LAT:MECH:ANL][VECT:LAT][ALG:LAT][STAT:LAT]
nodes: 6
arcs: 5
  undirected arcs: 0
directed arcs: 5
average markov blanket size: 2.00
average neighbourhood size: 1.67
average branching factor: 0.83

learning algorithm: Hill-Climbing
score: BIC (disc.)
penalization coefficient: 2.238668
tests used in the learning procedure: 40
optimized: TRUE
```

Note that for the learned network, variable LAT has two parents; MECH and ANL. If MECH, VECT, ALG, ANL, STAT were continuous, this distribution would therefore not fall into the CG framework.
13.1.7 Application to Gene Expression Data

The analysis for large arrays of gene expression data is dealt with in the following steps:

1. Outliers are removed. This is because, for continuous data, Bayesian Networks only supports multivariate Gaussian distributions; outliers make the Gaussian modelling assumptions less likely to hold.

2. Structure learning is repeated several times, so that there is more chance of finding a global maximiser for the score function.

3. The networks discovered in the previous step are averaged. This is a technique from Claskens and Hjort (2008) [27]. The averaged network uses arcs present in (say) 85% of the networks.

We try this on the sachs.data.txt data set, found in the data directory of the course home page:

```r
> library(bnlearn)
> sachs.data <- read.delim("~/data/sachs.data.txt")
> sachs<-sachs.data
> dsachs=discretize(sachs,method="hartemink",breaks=3,ibreaks=60,idisc="quantile")
```

Each variable in the dsachs data frame is a factor with three levels, corresponding approximately to low, normal and high expression. Now apply bootstrap resampling to learn a set of 500 networks to be used for model averaging:

```r
> boot=boot.strength(data=dsachs,R=500,algorithm="hc",algorithm.args=list(score="bde",iss=10))
> boot[[boot$strength>0.85)&(boot$direction>=0.5),]

   from     to strength direction
 1  praf  pmek  1.000  0.5180000
23  plcg  PIP2  1.000  0.5100000
24  plcg  PIP3  1.000  0.5220000
34  PIP2  PIP3  1.000  0.5120000
56  p44.42 pakts473  1.000  0.5620000
57  p44.42  PKA  0.992  0.5665323
67 pakts473  PKA  1.000  0.5690000
89       PKC  P38  1.000  0.5100000
90       PKC  pJnk  1.000  0.5100000
100      P38  pJnk  0.954  0.5062893
```

The virtual sample size is 10, which is very low. Arcs are significant if they appear in at least 85% of the networks and in the direction that appears most frequently. The averaged network is formed quite simply using the averaged.network function:

```r
> avg.boot = averaged.network(boot,threshold=0.85)
```
An alternative approach is to average the results of several hill climbing searches, each starting from a different network. The initial condition can be generated using a distribution over the space of connected graphs. An algorithm to do this was proposed by Ide and Cozman [56](2002). It is implemented by the function `random.graph()`. It is carried out as follows:

```r
> library("bnlearn", lib.loc="~/R/x86_64-redhat-linux-gnu-library/3.1")
> nodes=names(dsachs)
> start=random.graph(nodes=nodes,method="ic-dag",num=500)
> netlist=lapply(start,function(net){
  + hc(dsachs,score="bde",iss=10,start=net))
> rnd=custom.strength(netlist, nodes=nodes)
> rnd[(rnd$strength>0.85) & (rnd$direction>=0.5),]
  from  to    strength
def praf  pme  1  0.500
11 pme  praf  1  0.500
23 plc  PIP2  1  0.500
24 plc  PIP3  1  0.620
33 PIP2 plc  1  0.500
34 PIP2 PIP3  1  0.620
56 p44.42 pakts473 1  0.500
57 p44.42 PKA  1  0.507
66 pakts473 p44.42 1  0.500
67 pakts473 PKA  1  0.507
89 PKC  P38  1  0.500
90 PKC pjnk  1  0.500
99 P38  PKC  1  0.500
100 P38  pjnk 1  0.500
109 pjnk PKC  1  0.500
110 pjnk P38  1  0.500
> avg.start=averaged.network(rnd, threshold=0.85)
Warning messages:
1: In averaged.network.backend(strength = strength, nodes = nodes, :
   arc pjnk -> PKC would introduce cycles in the graph, ignoring.
2: In averaged.network.backend(strength = strength, nodes = nodes, :
   arc pjnk -> P38 would introduce cycles in the graph, ignoring.
> all.equal(cpdag(avg.boot), cpdag(avg.start))
[1] TRUE
```

The networks have the same skeleton, although some of the directions are different.

The score is computed first by taking `cpdag` to get an essential graph and then by taking `cextend` to form a dag.
The \texttt{bnlearn} package contains a default level for the threshold, which is found in \texttt{averaged.network}

\begin{verbatim}
> averaged.network(boot)

Random/Generated Bayesian network

model:
[praf] [plcg] [p44.42] [PKC] [psek] [praf] [PIP2] [plcg] [pakts473] [p44.42] [P38] [PKC]
[pjnk] [PKC] [PIP3] [plcg:PIP2] [PKA] [p44.42:pakts473]

nodes: 11
arcs: 9
undirected arcs: 0
directed arcs: 9
average markov blanket size: 1.64
average neighbourhood size: 1.64
average branching factor: 0.82

generation algorithm: Model Averaging
significance threshold: 0.954
\end{verbatim}

The default threshold is computed as follows: Let

\[ \hat{F}_{(k)} = \{ 0 \leq \hat{f}_{(1)} \leq \ldots \leq \hat{f}_{(k)} \leq 1 \} \]

denote the order statistics for the arc strengths stored in \texttt{boot}. Now, let \( \hat{t} \) denote a threshold and set

\[ \hat{F}_{(k)}(t) = \begin{cases} 
1 & \hat{f}_{(k)} \geq t \\
0 & \hat{f}_{(k)} < t. 
\end{cases} \]

This denotes the ‘empirical’ probability function for arc strengths for the graph where arcs are present if and only if \( \hat{f}_{(k)} \geq t \). Let \( \hat{\mathbf{P}}_{(\cdot)} \) denote the resulting vector.

Now choose \( \hat{t} \) to minimise

\[ L_{1}(t, \hat{\mathbf{P}}_{(\cdot)}) := \int |F_{\hat{\mathbf{P}}_{(\cdot)}} - F_{\mathbf{P}_{(\cdot)}}| dx \]

where \( F_{\hat{\mathbf{P}}_{(\cdot)}} \) and \( F_{\mathbf{P}_{(\cdot)}} \) are the empirical distribution functions of \( \hat{\mathbf{P}}_{(\cdot)} \) and \( \mathbf{P}_{(\cdot)} \) respectively. Then \( \hat{t} \), the threshold is chosen to minimise this.
13.1. **BAYESIAN NETWORKS WITH BNLEARN**

13.1.8 Interventional Data

The data set in `sachs.interventional.txt` gives data from different experiments, where the interventions to force the levels of certain variables, differ from experiment to experiment.

```r
> isachs <- read.table("~/data/sachs.interventional.txt", header=TRUE, colClasses="factor")
```

It is important that `colClasses = "factor"`.

One (less useful) way of dealing with the situation is to include the intervention INT in the network and make all the variables depend on it. This is done using the `whitelist` command, which contains all possible arcs from INT to the other nodes. These arcs are then forced to be present in the learned network structure.

```r
> wh = matrix(c(rep("INT",11),names(isachs)[1:11]),ncol=2)
> bn.wh = tabu(isachs, whitelist=wh, score="bde", iss=10, tabu=50)
```

The `tabu` learning algorithm gives more stable results here. Not all the arcs in `wh` are necessary. The `tiers2blacklist` function may be used to blacklist all arcs going towards INT, thus ensuring that only outgoing arcs are present.

```r
> tiers=list("INT",names(isachs)[1:11])
> bl = tiers2blacklist(nodes=tiers)
> bn.tiers = tabu(isachs, blacklist=bl, score="bde", iss=10, tabu=50)
```

While the two methods given above, producing `bn.wh` and `bn.tiers` show how to force certain arrows into a network, they do not involve the structure of the intervention.

The way to model an intervention is described as follows: the value of `INT` identifies which node is subject to an intervention. Therefore, we start by constructing a named list of which observations are manipulated for each node.

```r
> INT2 = sapply(1:11, function(x){which(isachs$INT==x)})
> nodes = names(isachs)[1:11]
> names(INT2) = nodes
```

Now pass the list to `tabu` as an additional argument for `mbde` (the *modified* BDe score function).

```r
> start = random.graph(nodes=nodes, method="melancon", num=500, burn.in=10^5, every=100)
> netlist = lapply(start, function(net){
  + tabu(isachs[,1:11], score="mbde", exp=INT2, iss=10, start=net, tabu=50))
```

```r
> bn.mbde = averaged.network(arcs, threshold=0.85)
```

**Warning messages:**

```r
1: In averaged.network.backend(strength = strength, nodes = nodes, : arc pjn -> PKA would introduce cycles in the graph, ignoring.
```
2: In averaged.network.backend(strength = strength, nodes = nodes, :
   arc PKC -> PKA would introduce cycles in the graph, ignoring.
3: In averaged.network.backend(strength = strength, nodes = nodes, :
   arc PKC -> P38 would introduce cycles in the graph, ignoring.
4: In averaged.network.backend(strength = strength, nodes = nodes, :
   arc pjnk -> P38 would introduce cycles in the graph, ignoring.
> bn.mbde2 <- cextend(cpdag(bn.mbde))
> graphviz.plot(bn.mbde2)
13.2 Exercises

1. This exercise uses the `asia` data set found in the `bnlearn` package.

   (a) Create a `bn` object with the network structure shown in Figure 13.3.

   ![Figure 13.3: Asia Network](image)

   (b) Derive the skeleton, the moral graph, and the essential graph representing the Markov equivalence class. Plot them using `graphviz.plot`.

   (c) Identify the parents, the children, the neighbours and the Markov blanket of each node.

   (d) For the network in Figure 13.3, estimate the CPPs.

   (e) Using the data `asia`, use the MMPC algorithm (called ‘grow-shrink’ in `bnlearn`) to learn the skeleton followed by hill climbing to learn the direction of the arrows. Is the output DAG Markov equivalent to the graph in Figure 13.3?

2. The `marks` data set is found in the `bnlearn` package.

   (a) Discretise the data using a quantile transform and different numbers of intervals (say 2 to 5). Learn the network structure. How does the structure change with the discretisation?

   (b) Repeat the discretisation using `interval` discretisation, using up to five intervals. Compare the resulting networks with those obtained previously using `quantile` discretisation.

   (c) Does Hartemink’s discretisation algorithm perform better than either quantile or interval discretisation? How does its behaviour depend on the number of initial breaks?

3. The ALARM network is a standard network used to test new algorithms. A synthetic data set `alarm` is found in the `bnlearn` package. Type:

   ```r
   > library(bnlearn)
   > ?alarm
   ```
On the bottom right quadrant of Rstudio, click on ALARM Monitoring System (synthetic) data set. This gives a description. Go to the bottom under Examples. You will find the structure of the ‘true’ network.

(a) Create a bn object for the true network using the model string provided in the documentation.

(b) Compare the networks learned from the data using different constraint based algorithms with the true network, both in terms of structural differences and also using either BIC or BDe.

(c) How are these constraint based strategies affected by different choices of $\alpha$ (the nominal significance level of each test)?

(d) Now learn the structure with hill-climbing and tabu search, using the posterior density BDe as a score function. How does the network change with the hyper parameters $\text{iss}$ (imaginary sample size)?

(e) Does the length of the tabu list have a significant impact on the network structures learned using tabu?

(f) Does the learned network depend on whether BDe or BIC is being used as a score criterion?

4. Now consider the data from Sachs et. al., found in sachs.data.txt on the course home page. Use the original data set; not the discretised data set.

(a) Evaluate the networks learned by hill-climbing with BIC and BGe, using cross-validation and the log-likelihood loss function.

(b) Use bootstrap resampling to evaluate the distribution of the number of arcs present in each of the networks learned. Do they differ significantly?

(c) Compute the averaged network structure for sachs using hill-climbing with BGe and different hyperparameters (imaginary sample sizes). How does the value of the significance threshold change as $\text{iss}$ increases?