1 The Discrete Case: Multinomial Bayesian Networks

In this chapter we will introduce the fundamental ideas behind Bayesian networks (BNs) and their basic interpretation, using a hypothetical survey on the usage of different means of transport. We will focus on modelling discrete data, leaving continuous data to Chapter 2 and more complex data types to Chapter 3.

1.1 Introductory Example: Train Use Survey

Consider a simple, hypothetical survey whose aim is to investigate the usage patterns of different means of transport, with a focus on cars and trains. Such surveys are used to assess customer satisfaction across different social groups, to evaluate public policies or for urban planning. Some real-world examples can be found, for example, in Kenett et al. (2012).

In our current example we will examine, for each individual, the following six discrete variables (labels used in computations and figures are reported in parenthesis):

- **Age** (`A`): the age, recorded as *young* (`young`) for individuals below 30 years old, *adult* (`adult`) for individuals between 30 and 60 years old, and *old* (`old`) for people older than 60.
- **Sex** (`S`): the biological sex of the individual, recorded as *male* (`M`) or *female* (`F`).
- **Education** (`E`): the highest level of education or training completed by the individual, recorded either as *up to high school* (`high`) or *university degree* (`uni`).
- **Occupation** (`O`): whether the individual is an *employee* (`emp`) or a *self-employed* (`self`) worker.
- **Residence** (`R`): the size of the city the individual lives in, recorded as either *small* (`small`) or *big* (`big`).

- **Travel** (T): the means of transport favoured by the individual, recorded either as *car* (car), *train* (train) or *other* (other).

In the scope of this survey, each variable falls into one of three groups. Age and Sex are *demographic indicators*. In other words, they are intrinsic characteristics of the individual; they may result in different patterns of behaviour, but are not influenced by the individual himself. On the other hand, the opposite is true for Education, Occupation and Residence. These variables are *socioeconomic indicators*, and describe the individual's position in society. Therefore, they provide a rough description of the individual's expected lifestyle; for example, they may characterise his spending habits or his work schedule. The last variable, Travel, is the *target* of the survey, the quantity of interest whose behaviour is under investigation.

1.2 Graphical Representation

The nature of the variables recorded in the survey, and more in general of the three categories they belong to, suggests how they may be related with each other. Some of those relationships will be *direct*, while others will be mediated by one or more variables (*indirect*).

Both kinds of relationships can be represented effectively and intuitively by means of a *directed graph*, which is one of the two fundamental entities characterising a BN. Each *node* in the graph corresponds to one of the variables in the survey. In fact, they are usually referred to interchangeably in literature. Therefore, the graph produced from this example will contain 6 nodes, labelled after the variables (A, S, E, O, R and T). Direct dependence relationships are represented as *arcs* between pairs of variables (*i.e.*, A → E means that E depends on A). The node at the tail of the arc is called the *parent*, while that at the head (where the arrow is) is called the *child*. Indirect dependence relationships are not explicitly represented. However, they can be read from the graph as sequences of arcs leading from one variable to the other through one or more mediating variables (*i.e.*, the combination of A → E and E → R means that R depends on A through E). Such sequences of arcs are said to form a *path* leading from one variable to the other; these two variables must be distinct. Paths of the form A → ... → A, which are known as *cycles*, are not allowed. For this reason, the graphs used in BNs are called *directed acyclic graphs* (DAGs).

Note, however, that some caution must be exercised in interpreting both direct and indirect dependencies. The presence of arrows or arcs seems to imply, at an intuitive level, that for each arc one variable should be interpreted as a *cause* and the other as an *effect* (*i.e.* A → E means that A causes E). This interpretation, which is called *causal*, is difficult to justify in most situations; for this reason, in general we speak about dependence relationships instead

of causal effects. The assumptions required for causal BN modelling will be discussed in Section 4.7.

To create and manipulate DAGs in the context of BNs, we will use mainly the **bnlearn** package (short for "**B**ayesian **n**etwork **learn**ing").

```
> library(bnlearn)
```

As a first step, we create a DAG with one node for each variable in the survey and no arcs.

```
> dag <- empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))
```

Such a DAG is usually called an *empty graph*, because it has an empty arc set. The DAG is stored in an object of class bn, which looks as follows when printed.

```
> dag
  Random/Generated Bayesian network

  model:
   [A][S][E][O][R][T]
  nodes:                                 6
  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
```

Now we can start adding the arcs that encode the direct dependencies between the variables in the survey. As we said in the previous section, Age and Sex are not influenced by any of the other variables. Therefore, there are no arcs pointing to either variable. On the other hand, both Age and Sex have a direct influence on Education. It is well known, for instance, that the number of people attending universities has increased over the years. As a consequence, younger people are more likely to have a university degree than older people.

```
> dag <- set.arc(dag, from = "A", to = "E")
```

Similarly, Sex also influences Education; the gender gap in university applications has been widening for many years, with women outnumbering and outperforming men.

```
> dag <- set.arc(dag, from = "S", to = "E")
```

In turn, Education strongly influences both Occupation and Residence. Clearly, higher education levels help in accessing more prestigious professions. In addition, people often move to attend a particular university or to find a job that matches the skills they acquired in their studies.

```
> dag <- set.arc(dag, from = "E", to = "O")
> dag <- set.arc(dag, from = "E", to = "R")
```

Finally, the preferred means of transport are directly influenced by both Occupation and Residence. For the former, the reason is that a few jobs require periodic long-distance trips, while others require more frequent trips but on shorter distances. For the latter, the reason is that both commute time and distance are deciding factors in choosing between travelling by car or by train.

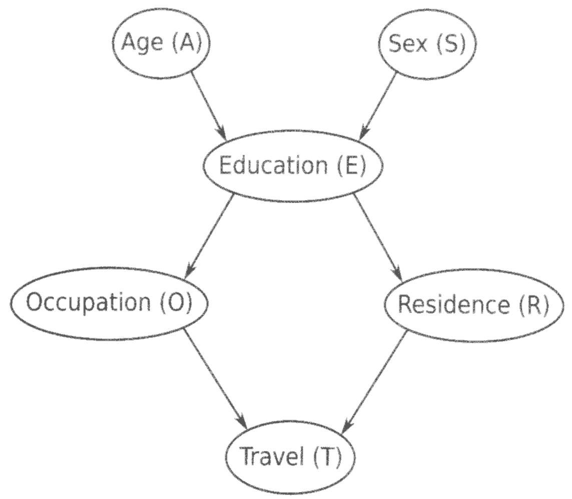
```
> dag <- set.arc(dag, from = "O", to = "T")
> dag <- set.arc(dag, from = "R", to = "T")
```

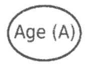
Now that we have added all the arcs, the DAG in the `dag` object encodes the desired direct dependencies. Its structure is shown in Figure 1.1, and can be read from the model formula generated from the `dag` object itself.

```
> dag
  Random/Generated Bayesian network

  model:
   [A][S][E|A:S][O|E][R|E][T|O:R]
  nodes:                                 6
  arcs:                                  6
    undirected arcs:                     0
    directed arcs:                       6
  average markov blanket size:           2.67
  average neighbourhood size:            2.00
  average branching factor:              1.00

  generation algorithm:                  Empty
```

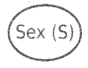
Direct dependencies are listed for each variable, denoted by a bar (`|`) and separated by semicolons (`:`). For example, `[E|A:S]` means that `A` → `E` and `S` → `E`; while `[A]` means that there is no arc pointing towards `A`. This representation of the graph structure is designed to recall a product of conditional probabilities, for reasons that will be clear in the next section, and can be produced with the `modelstring` function.

```
> modelstring(dag)
[1] "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

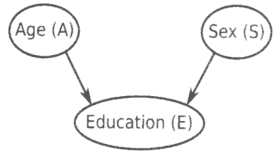




	young	adult	old
	0.30	0.50	0.20

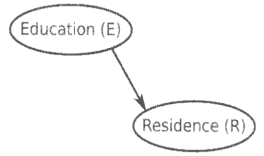


	M	F
	0.60	0.40

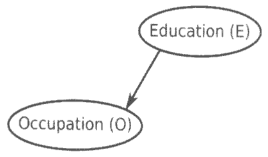


'ftable'

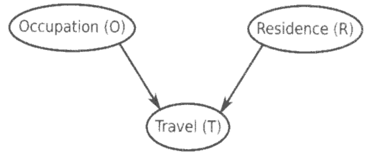
	high	uni
young & M	0.75	0.25
adult & M	0.72	0.28
old & M	0.88	0.12
young & F	0.64	0.36
adult & F	0.70	0.30
old & F	0.90	0.10



	small	big
high	0.25	0.75
uni	0.2	0.8



	emp	self
high	0.96	0.04
uni	0.92	0.08



	car	train	other
small & emp	0.48	0.42	0.10
small & self	0.56	0.36	0.08
big & emp	0.58	0.24	0.18
big & self	0.70	0.21	0.09

Figure 1.1
DAG representing the dependence relationships linking the variables recorded in the survey: Age (`A`), Sex (`S`), Education (`E`), Occupation (`O`), Residence (`R`) and Travel (`T`). The corresponding conditional probability tables are reported below.

bnlearn provides many other functions to investigate and manipulate `bn` objects. For a comprehensive overview, we refer the reader to the documentation included in the package. Two basic examples are `nodes` and `arcs`.

```
> nodes(dag)
[1] "A" "S" "E" "O" "R" "T"
> arcs(dag)
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
```

The latter function also provides a way to add arcs to a DAG that is faster than setting them one at a time. Obviously, the approach we used above is too cumbersome for large DAGs. Instead, we can create a matrix with the same structure as that returned by `arcs` and set the whole arc set at once.

```
> dag2 <- empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))
> arc.set <- matrix(c("A", "E",
+                     "S", "E",
+                     "E", "O",
+                     "E", "R",
+                     "O", "T",
+                     "R", "T"),
+              byrow = TRUE, ncol = 2,
+              dimnames = list(NULL, c("from", "to")))
> arcs(dag2) <- arc.set
```

The resulting DAG is identical to the previous one, `dag`.

```
> all.equal(dag, dag2)
[1] TRUE
```

Furthermore, both approaches guarantee that the DAG will indeed be acyclic; trying to introduce a cycle in the DAG returns an error.

```
> try(set.arc(dag, from = "T", to = "E"))
Error in arc.operations(x = x, from = from, to = to, op = "set",
check.cycles = check.cycles,  :
  the resulting graph contains cycles.
```

1.3 Probabilistic Representation

In the previous section we represented the interactions between Age, Sex, Education, Occupation, Residence and Travel using a DAG. To complete the BN modelling the survey, we will now specify a joint probability distribution over these variables. All of them are discrete and defined on a set of non-ordered states (called *levels* in R).

```
> A.lv <- c("young", "adult", "old")
> S.lv <- c("M", "F")
> E.lv <- c("high", "uni")
> O.lv <- c("emp", "self")
> R.lv <- c("small", "big")
> T.lv <- c("car", "train", "other")
```

Therefore, the natural choice for the joint probability distribution is a multinomial distribution, assigning a probability to each combination of states of the variables in the survey. In the context of BNs, this joint distribution is called the *global distribution*.

However, using the global distribution directly is difficult; even for small problems, such as that we are considering, the number of its parameters is very high. In the case of this survey, the parameter set includes the 143 probabilities corresponding to the combinations of the levels of all the variables. Fortunately, we can use the information encoded in the DAG to break down the global distribution into a set of smaller *local distributions*, one for each variable. Recall that arcs represent direct dependencies; if there is an arc from one variable to another, the latter depends on the former. In other words, variables that are not linked by an arc are *conditionally independent*. As a result, we can factorise the global distribution as follows:

$$\Pr(\mathtt{A},\mathtt{S},\mathtt{E},\mathtt{O},\mathtt{R},\mathtt{T}) = \Pr(\mathtt{A})\Pr(\mathtt{S})\Pr(\mathtt{E} \mid \mathtt{A},\mathtt{S})\Pr(\mathtt{O} \mid \mathtt{E})\Pr(\mathtt{R} \mid \mathtt{E})\Pr(\mathtt{T} \mid \mathtt{O},\mathtt{R}). \tag{1.1}$$

Equation (1.1) provides a formal definition of how the dependencies encoded in the DAG *map* into the probability space via conditional independence relationships. The absence of cycles in the DAG ensures that the factorisation is well defined. Each variable depends only on its parents; its distribution is univariate and has a (comparatively) small number of parameters. Even the set of all the local distributions has, overall, fewer parameters than the global distribution. The latter represents a more general model than the former, because it does not make any assumption on the dependencies between the variables. In other words, the factorisation in Equation (1.1) defines a *nested model* or a *submodel* of the global distribution.

In our survey, Age and Sex are modelled by simple, unidimensional probability tables (they have no parent).

```
> A.prob <- array(c(0.30, 0.50, 0.20), dim = 3,
+                 dimnames = list(A = A.lv))
> A.prob
A
young adult   old
  0.3   0.5   0.2
> S.prob <- array(c(0.60, 0.40), dim = 2,
+                 dimnames = list(S = S.lv))
> S.prob
S
  M   F
0.6 0.4
```

$P(A)$

$P(S)$

Occupation and Residence, which depend on Education, are modelled by two-dimensional conditional probability tables. Each column corresponds to one level of the parent, and holds the distribution of the variable conditional on that particular level. As a result, probabilities sum up to 1 within each column.

```
> O.prob <- array(c(0.96, 0.04, 0.92, 0.08), dim = c(2, 2),
+                 dimnames = list(O = O.lv, E = E.lv))
> O.prob
      E
O      high  uni
  emp  0.96 0.92
  self 0.04 0.08
> R.prob <- array(c(0.25, 0.75, 0.20, 0.80), dim = c(2, 2),
+                 dimnames = list(R = R.lv, E = E.lv))
> R.prob
       E
R        high uni
  small 0.25 0.2
  big   0.75 0.8
```

$P(O|E)$

$R(R|E)$

For these one- and two-dimensional distributions, we can also use the `matrix` function to create the (conditional) probability tables. The syntax is almost identical to that of `array`; the difference is that only one dimension (either the number of rows, `nrow`, or the number of columns, `ncol`) must be specified.

```
> R.prob <- matrix(c(0.25, 0.75, 0.20, 0.80), ncol = 2,
+                 dimnames = list(R = R.lv, E = E.lv))
```

```
> R.prob
       E
R        high uni
  small 0.25 0.2
  big   0.75 0.8
```

Finally, Education and Travel are modelled as three-dimensional tables, since they have two parents each (Age and Sex for Education, Occupation and Residence for Travel). Each column corresponds to one combination of the levels of the parents, and holds the distribution of the variable conditional on that particular combination.

$P(E|A,S)$

```
> E.prob <- array(c(0.75, 0.25, 0.72, 0.28, 0.88, 0.12, 0.64,
+                 0.36, 0.70, 0.30, 0.90, 0.10), dim = c(2, 3, 2),
+                 dimnames = list(E = E.lv, A = A.lv, S = S.lv))
> T.prob <- array(c(0.48, 0.42, 0.10, 0.56, 0.36, 0.08, 0.58,
+                 0.24, 0.18, 0.70, 0.21, 0.09), dim = c(3, 2, 2),
+                 dimnames = list(T = T.lv, O = O.lv, R = R.lv))
```

$P(T|O,R)$

Overall, the local distributions we defined above have just 21 parameters, compared to the 143 of the global distribution. Furthermore, local distributions can be handled independently from each other, and have at most 8 parameters each. This reduction in dimension is a fundamental property of BNs, and makes their application feasible for high-dimensional problems.

Now that we have defined both the DAG and the local distribution corresponding to each variable, we can combine them to form a fully-specified BN. For didactic purposes, we recreate the DAG using the model formula interface provided by `modelstring`, whose syntax is almost identical to Equation (1.1). The nodes and the parents of each node can be listed in any order, thus allowing us to follow the logical structure of the network in writing the formula.

```
> dag3 <- model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
```

The resulting DAG is identical to that we created in the previous section, as shown below.

```
> all.equal(dag, dag3)
[1] TRUE
```

Then we combine the DAG we stored in `dag` and a list containing the local distributions, which we will call `cpt`, into an object of class `bn.fit` called `bn`.

```
> cpt <- list(A = A.prob, S = S.prob, E = E.prob, O = O.prob,
+          R = R.prob, T = T.prob)
> bn <- custom.fit(dag, cpt)
```

The number of parameters of the BN can be computed with the `nparams` function and is indeed 21, as expected from the parameter sets of the local distributions.

```
> nparams(bn)
[1] 21
```

Objects of class `bn.fit` are used to describe BNs in **bnlearn**. They include information about both the DAG (such as the parents and the children of each node) and the local distributions (their parameters). For most practical purposes, they can be used as if they were objects of class `bn` when investigating graphical properties. So, for example,

```
> arcs(bn)
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
```

and the same holds for other functions such as `nodes`, `parents`, and `children`. Furthermore, the conditional probability tables can either be printed from the `bn.fit` object,

```
> bn$R
  Parameters of node R (multinomial distribution)

Conditional probability table:


       E
R       high uni
  small 0.25 0.20
  big   0.75 0.80
```

or extracted for later use with the `coef` function as follows.

```
> R.cpt <- coef(bn$R)
```

Just typing

```
> bn
```

causes all the conditional probability tables in the BN to be printed.

1.4 Estimating the Parameters: Conditional Probability Tables

For the hypothetical survey described in this chapter, we have assumed to know both the DAG and the parameters of the local distributions defining the BN. In this scenario, BNs are used as *expert systems*, because they formalise the knowledge possessed by one or more experts in the relevant fields. However, in most cases the parameters of the local distributions will be estimated (or *learned*) from an observed sample. Typically, the data will be stored in a text file we can import with `read.table`,

```
> survey <- read.table("survey.txt", header = TRUE)
```

with one variable per column (labelled in the first row) and one observation per line.

```
> head(survey)
      A     R    E   O S     T
1 adult   big high emp F   car
2 adult small  uni emp M   car
3 adult   big  uni emp F train
4 adult   big high emp M   car
5 adult   big high emp M   car
6 adult small high emp F train
```

In the case of this survey, and of discrete BNs in general, the parameters to estimate are the conditional probabilities in the local distributions. They can be estimated, for example, with the corresponding empirical frequencies in the data set, *e.g.*,

$$\begin{aligned}\widehat{\Pr}(\mathtt{O}=\mathtt{emp}\mid\mathtt{E}=\mathtt{high}) &= \frac{\widehat{\Pr}(\mathtt{O}=\mathtt{emp},\mathtt{E}=\mathtt{high})}{\widehat{\Pr}(\mathtt{E}=\mathtt{high})} = \\ &= \frac{\text{number of observations for which }\mathtt{O}=\mathtt{emp}\text{ and }\mathtt{E}=\mathtt{high}}{\text{number of observations for which }\mathtt{E}=\mathtt{high}}. \qquad (1.2)\end{aligned}$$

This yields the classic *frequentist* and *maximum likelihood* estimates. In **bnlearn**, we can compute them with the `bn.fit` function. `bn.fit` complements the `custom.fit` function we used in the previous section; the latter constructs a BN using a set of *custom* parameters specified by the user, while the former estimates the same from the data.

```
> bn.mle <- bn.fit(dag, data = survey, method = "mle")
```

Similarly to `custom.fit`, `bn.fit` returns an object of class `bn.fit`. The `method` argument determines which estimator will be used; in this case, `"mle"` for the maximum likelihood estimator. Again, the structure of the network is assumed to be known, and is passed to the function via the dag object. For didactic purposes, we can also compute the same estimates manually

```
> prop.table(table(survey[, c("O", "E")]), margin = 2)
      E
O          high    uni
  emp  0.9808 0.9259
  self 0.0192 0.0741
```

and verify that we get the same result as bn.fit.

```
> bn.mle$O
  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O          high    uni
  emp  0.9808 0.9259
  self 0.0192 0.0741
```

As an alternative, we can also estimate the same conditional probabilities in a Bayesian setting, using their posterior distributions. An overview of the underlying probability ~~theory and~~ the distributions relevant for BNs is provided in Appendixes B.3, B.4 and B.5. In this case, the method argument of bn.fit must be set to "bayes".

```
> bn.bayes <- bn.fit(dag, data = survey, method = "bayes",
+                    iss = 10)
```

The estimated posterior probabilities are computed from a uniform prior over each conditional probability table. The iss optional argument, whose name stands for *imaginary sample size* (also known as *equivalent sample size*), determines how much weight is assigned to the prior distribution compared to the data when computing the posterior. The weight is specified as the size of an imaginary sample supporting the prior distribution. Its value is divided by the number of cells in the conditional probability table (because the prior is flat) and used to compute the posterior estimate as a weighted mean with the empirical frequencies. So, for example, suppose we have a sample of size n, which we can compute as nrow(survey). If we let

$$\hat{p}_{\texttt{emp,high}} = \frac{\text{number of observations for which } \texttt{O} = \texttt{emp} \text{ and } \texttt{E} = \texttt{high}}{n} \qquad (1.3)$$

$$\hat{p}_{\texttt{high}} = \frac{\text{number of observations for which } \texttt{E} = \texttt{high}}{n} \qquad (1.4)$$

and we denote the corresponding prior probabilities as

$$\pi_{\texttt{emp,high}} = \frac{1}{\texttt{nO} \times \texttt{nE}} \quad \text{and} \quad \pi_{\texttt{high}} = \frac{\texttt{nO}}{\texttt{nO} \times \texttt{nE}} \qquad (1.5)$$

where nO = nlevels(bn.bayes$O) and nE = nlevels(bn.bayes$E), we have that

$$\widehat{\Pr}(\texttt{O} = \texttt{emp}, \texttt{E} = \texttt{high}) = \frac{\texttt{iss}}{n + \texttt{iss}}\pi_{\texttt{emp,high}} + \frac{n}{n + \texttt{iss}}\hat{p}_{\texttt{emp,high}} \qquad (1.6)$$

$$\widehat{\Pr}(\texttt{E} = \texttt{high}) = \frac{\texttt{iss}}{n + \texttt{iss}}\pi_{\texttt{high}} + \frac{n}{n + \texttt{iss}}\hat{p}_{\texttt{high}} \qquad (1.7)$$

and therefore that

$$\widehat{\Pr}(\texttt{O} = \texttt{emp} \mid \texttt{E} = \texttt{high}) = \frac{\widehat{\Pr}(\texttt{O} = \texttt{emp}, \texttt{E} = \texttt{high})}{\widehat{\Pr}(\texttt{E} = \texttt{high})}. \qquad (1.8)$$

The value of iss is typically chosen to be small, usually between 1 and 15, to allow the prior distribution to be easily dominated by the data. Such small values result in conditional probabilities that are smoother but still close to the empirical frequencies (*i.e.* $\hat{p}_{\texttt{emp,high}}$) they are computed from.

```
> bn.bayes$O
  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O          high    uni
  emp  0.9743 0.9107
  self 0.0257 0.0893
```

As we can see from the conditional probability table above, all the posterior estimates are farther from both 0 and 1 than the corresponding maximum likelihood estimates due to the influence of the prior distribution. This is desirable for several reasons. First of all, this ensures that the regularity conditions of model estimation and inference methods are fulfilled. In particular, it is not possible to obtain sparse conditional probability tables (with many zero cells) even from small data sets. Furthermore, posterior estimates are more robust than maximum likelihood estimates and result in BNs with better predictive power.

Increasing the value of iss makes the posterior distribution more and more flat, pushing it towards the uniform distribution used as the prior. As shown in Figure 1.2, for large values of iss the conditional posterior distributions for $\Pr(\texttt{O} \mid \texttt{E} = \texttt{high})$ and $\Pr(\texttt{O} \mid \texttt{E} = \texttt{uni})$ assign a probability of approximately 0.5 to both self and emp. This trend is already apparent if we compare the conditional probabilities obtained for iss = 10 with those for iss = 20, reported below.

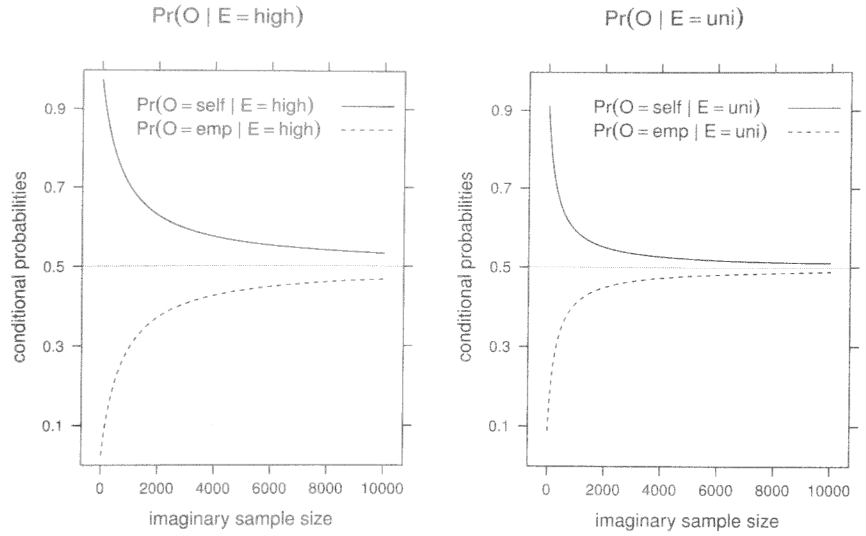


Figure 1.2
Conditional probability distributions for `O` given both possible values of `E`, that is, Pr(`O` | `E` = `high`) and Pr(`O` | `E` = `uni`), converge to uniform distributions as the imaginary sample size increases.

```
> bn.bayes <- bn.fit(dag, data = survey, method = "bayes",
+                    iss = 20)
> bn.bayes$O

  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O       high   uni
  emp  0.968 0.897
  self 0.032 0.103
```

1.5 Learning the DAG Structure: Tests and Scores

In the previous sections we have assumed that the DAG underlying the BN is known. In other words, we rely on prior knowledge on the phenomenon we are

modelling to decide which arcs are present in the graph and which are not. However, this is not always possible or desired; the structure of the DAG itself may be the object of our investigation. It is common in genetics and systems biology, for instance, to reconstruct the molecular pathways and networks underlying complex diseases and metabolic processes. An outstanding example of this kind of study can be found in Sachs et al. (2005) and will be explored in Chapter 6. In the context of social sciences, the structure of the DAG may identify which nodes are directly related to the target of the analysis and may therefore be used to improve the process of policy making. For instance, the DAG of the survey we are using as an example suggests that train fares should be adjusted (to maximise profit) on the basis of Occupation and Residence alone.

Learning the DAG of a BN is a complex task, for two reasons. First, the space of the possible DAGs is very big; the number of DAGs increases super-exponentially as the number of nodes grows. As a result, only a small fraction of its elements can be investigated in a reasonable time. Furthermore, this space is very different from real spaces (*e.g.*, $\mathbb{R}$, $\mathbb{R}^2$, $\mathbb{R}^3$, etc.) in that it is not continuous and has a finite number of elements. Therefore, *ad-hoc* algorithms are required to explore it. We will investigate the algorithms proposed for this task and their theoretical foundations in Section 4.5. For the moment, we will limit our attention to the two classes of statistical criteria used by those algorithms to evaluate DAGs: *conditional independence tests* and *network scores*.

1.5.1 Conditional Independence Tests

? adding arc

Conditional independence tests focus on the presence of individual arcs. Since each arc encodes a probabilistic dependence, conditional independence tests can be used to assess whether that probabilistic dependence is supported by the data. If the null hypothesis (of conditional independence) is rejected, the arc can be considered for inclusion in the DAG. For instance, consider adding an arc from Education to Travel ($\mathtt{E} \to \mathtt{T}$) to the DAG shown in Figure 1.1. The null hypothesis is that Travel is probabilistically independent ($\perp\!\!\!\perp_P$) from Education conditional on its parents, *i.e.*,

$$H_0 : \mathtt{T} \perp\!\!\!\perp_P \mathtt{E} \mid \{\mathtt{O}, \mathtt{R}\}, \tag{1.9}$$

and the alternative hypothesis is that

$$H_1 : \mathtt{T} \not\!\perp\!\!\!\perp_P \mathtt{E} \mid \{\mathtt{O}, \mathtt{R}\}. \tag{1.10}$$

We can test this null hypothesis by adapting either the *log-likelihood ratio* G^2 or *Pearson's* X^2 to test for conditional independence instead of marginal independence. For G^2, the test statistic assumes the form

$$G^2(\mathtt{T}, \mathtt{E} \mid \mathtt{O}, \mathtt{R}) = \sum_{t \in \mathtt{T}} \sum_{e \in \mathtt{E}} \sum_{k \in \mathtt{O} \times \mathtt{R}} \frac{n_{tek}}{n} \log \frac{n_{tek} n_{++k}}{n_{t+k} n_{+ek}}, \tag{1.11}$$

where we denote the categories of Travel with $t \in \mathtt{T}$, the categories of Education with $e \in \mathtt{E}$, and the configurations of Occupation and Residence with $k \in \mathtt{O} \times \mathtt{R}$. Hence, n_{tek} is the number of observations for the combination of a category t of Travel, a category e of Education and a category k of $\mathtt{O} \times \mathtt{R}$. The use of a "+" subscript denotes the sum over an index, as in the classic book from Agresti (2013), and is used to indicate the marginal counts for the remaining variables. So, for example, n_{t+k} is the number of observations for t and k obtained by summing over all the categories of Education. For Pearson's X^2, using the same notation we have that

$$X^2(\mathtt{T}, \mathtt{E} \mid \mathtt{O}, \mathtt{R}) = \sum_{t \in \mathtt{T}} \sum_{e \in \mathtt{E}} \sum_{k \in \mathtt{O} \times \mathtt{R}} \frac{(n_{tek} - m_{tek})^2}{m_{tek}}, \quad \text{where} \quad m_{tek} = \frac{n_{t+k} n_{+ek}}{n_{++k}}. \tag{1.12}$$

Both tests have an asymptotic χ^2 distribution under the null hypothesis, in this case with

```
> (nlevels(survey[, "T"]) - 1) * (nlevels(survey[, "E"]) - 1) *
+   (nlevels(survey[, "O"]) * nlevels(survey[, "R"]))
[1] 8
```

degrees of freedom. Conditional independence results in small values of G^2 and X^2; conversely, the null hypothesis is rejected for large values of the test statistics, which increase with the strength of the conditional dependence between the variables.

The `ci.test` function from **bnlearn** implements both G^2 and X^2, in addition to other tests which will be covered in Section 4.5.1.1. The G^2 test, which is equivalent to the *mutual information* test from information theory, is used when `test = "mi"`.

```
> ci.test("T", "E", c("O", "R"), test = "mi", data = survey)
        Mutual Information (disc.)

data:  T ~ E | O + R
mi = 9.88, df = 8, p-value = 0.2733
alternative hypothesis: true value is greater than 0
```

Pearson's X^2 test is used when `test = "x2"`.

```
> ci.test("T", "E", c("O", "R"), test = "x2", data = survey)
        Pearson's X^2

data:  T ~ E | O + R
x2 = 5.74, df = 8, p-value = 0.6766
alternative hypothesis: true value is greater than 0
```

Both tests return very large p-values, indicating that the dependence relationship encoded by $\mathtt{E} \to \mathtt{T}$ is not significant given the current DAG structure.

We can test in a similar way whether one of the arcs in the DAG should be removed because the dependence relationship it encodes is not supported by the data. So, for example, we can remove $\mathtt{O} \to \mathtt{T}$ by testing

$$H_0 : \mathtt{T} \perp\!\!\!\perp_P \mathtt{O} \mid \mathtt{R} \qquad \text{versus} \qquad H_1 : \mathtt{T} \not\!\perp\!\!\!\perp_P \mathtt{O} \mid \mathtt{R} \tag{1.13}$$

as follows.

```
> ci.test("T", "O", "R", test = "x2", data = survey)
        Pearson's X^2

data:  T ~ O | R
x2 = 2.34, df = 4, p-value = 0.6727
alternative hypothesis: true value is greater than 0
```

Again, we find that $\mathtt{O} \to \mathtt{T}$ is not significant.

The task of testing each arc in turn for significance can be automated using the `arc.strength` function, and specifying the test label with the `criterion` argument.

```
> arc.strength(dag, data = survey, criterion = "x2")
  from to strength
1    A  E  0.00098
2    S  E  0.00125
3    E  O  0.00264
4    E  R  0.00056
5    O  T  0.67272
6    R  T  0.00168
```

`arc.strength` is designed to measure the strength of the probabilistic dependence corresponding to each arc by removing that particular arc from the graph and quantifying the change with some probabilistic `criterion`. Possible choices are a conditional independence test (in the example above) or a network score (in the next section). In the case of conditional independence tests, the value of the `criterion` argument is the same as that of the `test` argument in `ci.test`, and the test is for the `to` node to be independent from the `from` node conditional on the remaining parents of `to`. The reported `strength` is the resulting p-value. What we see from the output above is that all arcs with the exception of $\mathtt{O} \to \mathtt{T}$ have p-values smaller than 0.05 and are well supported by the data.

1.5.2 Network Scores

Unlike conditional independence tests, network scores focus on the DAG as a whole; they are goodness-of-fit statistics measuring how well the DAG mirrors the dependence structure of the data. Again, several scores are in common use. One of them is the *Bayesian Information criterion* (BIC), which for our survey BN takes the form

$$\begin{aligned}\text{BIC} &= \log \widehat{\Pr}(\mathtt{A},\mathtt{S},\mathtt{E},\mathtt{O},\mathtt{R},\mathtt{T}) - \frac{d}{2}\log n = \\ &= \left[\log \widehat{\Pr}(\mathtt{A}) - \frac{d_{\mathtt{A}}}{2}\log n\right] + \left[\log \widehat{\Pr}(\mathtt{S}) - \frac{d_{\mathtt{S}}}{2}\log n\right] + \\ &+ \left[\log \widehat{\Pr}(\mathtt{E} \mid \mathtt{A},\mathtt{S}) - \frac{d_{\mathtt{E}}}{2}\log n\right] + \left[\log \widehat{\Pr}(\mathtt{O} \mid \mathtt{E}) - \frac{d_{\mathtt{O}}}{2}\log n\right] + \\ &+ \left[\log \widehat{\Pr}(\mathtt{R} \mid \mathtt{E}) - \frac{d_{\mathtt{R}}}{2}\log n\right] + \left[\log \widehat{\Pr}(\mathtt{T} \mid \mathtt{O},\mathtt{R}) - \frac{d_{\mathtt{T}}}{2}\log n\right] \qquad (1.14)\end{aligned}$$

where n is the sample size, d is the number of parameters of the whole network (*i.e.*, 21) and $d_{\mathtt{A}}$, $d_{\mathtt{S}}$, $d_{\mathtt{E}}$, $d_{\mathtt{O}}$, $d_{\mathtt{R}}$ and $d_{\mathtt{T}}$ are the numbers of parameters associated with each node. The decomposition in Equation (1.1) makes it easy to compute BIC from the local distributions. Another score commonly used in literature is the *Bayesian Dirichlet equivalent uniform* (BDeu) posterior probability of the DAG associated with a uniform prior over both the space of the DAGs and of the parameters; its general form is given in Section 4.5. It is often denoted simply as BDe. Both BIC and BDe assign higher scores to DAGs that fit the data better.

Both scores can be computed in **bnlearn** using the `score` function; BIC is computed when `type = "bic"`, and log BDe when `type = "bde"`.

```
> score(dag, data = survey, type = "bic")
[1] -2012.69
> score(dag, data = survey, type = "bde", iss = 10)
[1] -1998.28
```

Note that the `iss` argument for BDe is the same imaginary sample size we introduced when computing posterior estimates of the BN's parameters in Section 1.4. As before, it can be interpreted as the weight assigned to the (flat) prior distribution in terms of the size of an imaginary sample. For small values of `iss` or large observed samples, log BDe and BIC scores yield similar values.

```
> score(dag, data = survey, type = "bde", iss = 1)
[1] -2015.65
```

Using either of these scores it is possible to compare different DAGs and investigate which fits the data better. For instance, we can consider once more whether the DAG from Figure 1.1 fits the `survey` data better before or after adding the arc $\mathtt{E} \rightarrow \mathtt{T}$.

```
> dag4 <- set.arc(dag, from = "E", to = "T")
> nparams(dag4, survey)
[1] 29
> score(dag4, data = survey, type = "bic")
[1] -2032.6
```

Again, adding $\mathtt{E} \rightarrow \mathtt{T}$ is not beneficial, as the increase in $\log \widehat{\Pr}(\mathtt{A},\mathtt{S},\mathtt{E},\mathtt{O},\mathtt{R},\mathtt{T})$ is not sufficient to offset the heavier penalty from the additional parameters. The score for `dag4` (−2032.6) is lower than that of `dag3` (−2012.69).

Scores can also be used to compare completely different networks, unlike conditional independence tests. We can even generate a DAG at random with `random.graph` and compare it to the previous DAGs through its score.

```
> rnd <- random.graph(nodes = c("A", "S", "E", "O", "R", "T"))
> modelstring(rnd)
[1] "[A][S|A][E|A:S][O|S:E][R|S:E][T|S:E]"
> score(rnd, data = survey, type = "bic")
[1] -2034.99
```

As expected, `rnd` is worse than `dag` and even `dag4`; after all, neither data nor common sense are used to select its structure! Learning the DAG from `survey` yields a much better network. There are several algorithms that tackle this problem by searching for the DAG that maximises a given network score; some will be illustrated in Section 4.5.1.2. A simple one is *hill-climbing*: starting from a DAG with no arcs, it adds, removes and reverses one arc at a time and picks the change that increases the network score the most. It is implemented in the `hc` function, which in its simplest form takes the data (`survey`) as the only argument and defaults to the BIC score.

```
> learned <- hc(survey)
> modelstring(learned)
[1] "[R][E|R][T|R][A|E][O|E][S|E]"
> score(learned, data = survey, type = "bic")
[1] -1998.43
```

Other scores can be specified with the `score` argument; for example, we can change the default `score = "bic"` to `score = "bde"`.

```
> learned2 <- hc(survey, score = "bde")
```

Unsurprisingly, removing any arc from `learned` decreases its BIC score. We can confirm this conveniently using `arc.strength`, which reports the change in the score caused by an arc removal as the arc's `strength` when `criterion` is a network score.

3. M.L. Explanation

```
> arc.strength(learned, data = survey, criterion = "bic")
  from to strength
1    R  E   -3.390
2    E  S   -2.726
3    R  T   -1.848
4    E  A   -1.720
5    E  O   -0.827
```

This is not true for dag, suggesting that not all the dependencies it encodes can be learned correctly from survey.

```
> arc.strength(dag, data = survey, criterion = "bic")
  from to strength
1    A  E    2.489
2    S  E    1.482
3    E  O   -0.827
4    E  R   -3.390
5    O  T   10.046
6    R  T    2.973
```

In particular, removing O → T causes a marked increase in the BIC score, which is consistent with the high p-value we observed for this arc when using arc.strength in the previous section.

1.6 Using Discrete BNs

A BN can be used for inference through either its DAG or the set of local distributions. The process of answering questions using either of these two approaches is known in computer science as *querying*. If we consider a BN as an expert system, we can imagine asking it questions (*i.e.*, querying it) as we would a human expert and getting answers out of it. They may take the form of probabilities associated with an event under specific conditions, leading to *conditional probability queries*; they may validate the association between two variables after the influence of other variables is removed, leading to *conditional independence queries*; or they may identify the most likely state of one or more variables, leading to *most likely explanation* queries.

1.6.1 Using the DAG Structure (2. C I Q)

Using the DAG we saved in dag, we can investigate whether a variable is associated to another, essentially asking a conditional independence query. Both direct and indirect associations between two variables can be read from the DAG by checking whether they are connected in some way. If the variables

depend directly on each other, there will be a single arc connecting the nodes corresponding to those two variables. If the dependence is indirect, there will be two or more arcs passing through the nodes that mediate the association. In general, two sets **X** and **Y** of variables are independent given a third set **Z** of variables if there is no set of arcs connecting them that is not *blocked* by the conditioning variables. Conditioning on **Z** is equivalent to *fixing* the values of its elements, so that they are known quantities. In other words, the **X** and **Y** are *separated* by **Z**, which we denote with $\mathbf{X} \perp\!\!\!\perp_G \mathbf{Y} \mid \mathbf{Z}$. Given that BNs are based on DAGs, we speak of *d-separation* (directed separation); a formal treatment of its definition and properties is provided in Section 4.1. For the moment, we will just say that graphical separation ($\perp\!\!\!\perp_G$) implies probabilistic independence ($\perp\!\!\!\perp_P$) in a BN; if all the paths between **X** and **Y** are blocked, **X** and **Y** are (conditionally) independent. The converse is not necessarily true: not every conditional independence relationship is reflected in the graph.

We can investigate whether two nodes in a bn object are d-separated using the dsep function. dsep takes three arguments, x, y and z, corresponding to **X**, **Y** and **Z**; the first two must be the names of two nodes being tested for d-separation, while the latter is an optional d-separating set. So, for example, we can see from dag that both S and O are associated with R.

```
> dsep(dag, x = "S", y = "R")
[1] FALSE
> dsep(dag, x = "O", y = "R")
[1] FALSE
```

Clearly, S is associated with R because E is influenced by S (S → E) and R is influenced by E (E → R). In fact, the path function shows that there is a path leading from S to R

```
> path(dag, from = "S", to = "R")
[1] TRUE
```

and, if we condition on E, that path is blocked and S and R become independent.

```
> dsep(dag, x = "S", y = "R", z = "E")
[1] TRUE
```

From Equation (1.1), we can see that indeed the global distribution decomposes cleanly in a part that depends only on S and in a part that depends only on R once E is known:

$$\Pr(\mathrm{S}, \mathrm{R} \mid \mathrm{E}) = \Pr(\mathrm{S} \mid \mathrm{E}) \Pr(\mathrm{R} \mid \mathrm{E}). \tag{1.15}$$

The same holds for R and O. They both depend on E, and therefore become independent if we condition on it.

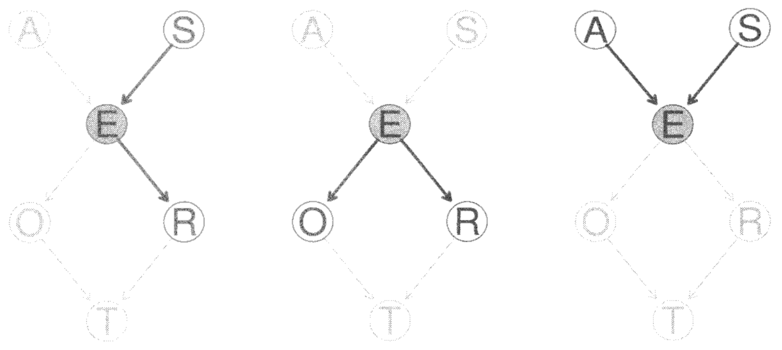


Figure 1.3
Some examples of d-separation covering the three fundamental connections: the *serial connection* (left), the *divergent connection* (centre) and the *convergent connection* (right). Nodes in the conditioning set are highlighted in grey.

```
> dsep(dag, x = "O", y = "R", z = "E")
[1] TRUE
```

Again, from Equation (1.1) we have

$$\Pr(\mathtt{O}, \mathtt{R} \mid \mathtt{E}) = \Pr(\mathtt{O} \mid \mathtt{E}) \Pr(\mathtt{R} \mid \mathtt{E}). \tag{1.16}$$

On the other hand, conditioning on a particular node can also make two other nodes dependent when they are marginally independent. Consider the following example involving A and S conditional on E.

```
> dsep(dag, x = "A", y = "S")
[1] TRUE
> dsep(dag, x = "A", y = "S", z = "E")
[1] FALSE
```

From Figure 1.3, we can see that the state of E is influenced by A and S at the same time; intuitively, if we know what kind of Education one individual has, some combinations of his Age and Sex become more likely than others and, in turn, these two variables become dependent. Equivalently, we can see from Equation (1.1) that E depends on the joint distribution of A and S, as Pr(E | A, S); then using Bayes' theorem we have

$$\Pr(\mathtt{E} \mid \mathtt{A}, \mathtt{S}) = \frac{\Pr(\mathtt{A}, \mathtt{S}, \mathtt{E})}{\Pr(\mathtt{A}, \mathtt{S})} = \frac{\Pr(\mathtt{A}, \mathtt{S} \mid \mathtt{E}) \Pr(\mathtt{E})}{\Pr(\mathtt{A}) \Pr(\mathtt{S})} \propto \Pr(\mathtt{A}, \mathtt{S} \mid \mathtt{E}). \tag{1.17}$$

Therefore, when E is known we cannot decompose the joint distribution of A

and S in a part that depends only on A and in a part that depends only on S. However, note that Pr(A, S) = Pr(A | S) Pr(S) = Pr(A) Pr(S): as we have seen above using **dsep**, A and S are d-separated if we are not conditioning on E.

The three examples we have examined above and in Figure 1.3 cover all the possible configurations of three nodes and two arcs. These simple structures are known in literature as *fundamental connections* and are the building blocks of the graphical and probabilistic properties of BNs.

In particular: "chain"

- structures like S → E → R (the first example) are known as *serial connections*, since both arcs have the same direction and follow one after the other; "fork" "common cause"
- structures like R ← E → O (the second example) are known as *divergent connections*, because the two arcs have divergent directions from a central node; "colliders" "inverted fork"
- structures like A → E ← S (the third example) are known as *convergent connections*, because the two arcs converge to a central node. When there is no arc linking the two parents (*i.e.*, neither A → S nor A ← S) convergent connections are called *v-structures*. As we will see in Chapter 4, their properties are crucial in characterising and learning BNs. "common effect"

1.6.2 Using the Conditional Probability Tables

In the previous section we have seen how we can answer conditional independence queries using only the information encoded in the DAG. More complex queries, however, require the use of the local distributions. The DAG is still used indirectly, as it determines the composition of the local distributions and reduces the effective dimension of inference problems.

The two most common types of inference are *conditional probability queries*, which investigate the distribution of one or more variables under non-trivial conditioning, and *most likely explanation* queries, which look for the most likely outcome of one or more variables (again under non-trivial conditioning). In both contexts, the variables being conditioned on are the new *evidence* or *findings* which force the probability of an *event* of interest to be re-evaluated. These queries can be answered in two ways, using either *exact* or *approximate inference*; we will describe the theoretical properties of both approaches in more detail in Section 4.6.

1.6.2.1 Exact Inference

Exact inference, which is implemented in package **gRain** (short for "**gRa**phical model **in**ference"), relies on transforming the BN into a specially crafted tree to speed up the computation of conditional probabilities.

```
> library(gRain)
```

Such a tree is called a *junction tree*, and can be constructed as follows from the bn object we created in the previous section (see also Algorithm 4.4, Section 4.6.2 for a description of the required steps). from custom.fit on p 9 (bottom)

```
> junction <- compile(as.grain(bn))
```

Once the junction tree has been built (by as.grain) and its probability tables have been computed (by compile), we can input the evidence into junction using the setEvidence function. The local distributions of the nodes the evidence refers to are then updated, and the changes are propagated through the junction tree. The actual query is performed by the querygrain function, which extracts the distribution of the nodes of interest from junction.

We may be interested, for example, in the attitudes of women towards car and train use compared to the whole survey sample.

P(T)

```
> querygrain(junction, nodes = "T")$T
T
   car  train  other
0.5618 0.2809 0.1573
```

variable levels

```
> jsex <- setEvidence(junction, nodes = "S", states = "F")
> querygrain(jsex, nodes = "T")$T
T
   car  train  other
0.5621 0.2806 0.1573
```

$P(T \mid S=F)$

There are no marked differences in the probabilities derived from junction before and after calling setEvidence. The former correspond to Pr(T), the latter to Pr(T | S = F). This suggests that women show about the same preferences towards car and train use as the interviewees as a whole.

Another interesting problem is how living in a small city affects car and train use, that is, Pr(T | R = small). People working in big cities often live in neighbouring towns and commute to their workplaces, because house prices are lower as you move out into the countryside. This, however, forces them to travel mostly by car or train because other means of transport (bicycles, tube, bus lines, etc.) are either unavailable or impractical.

```
> jres <- setEvidence(junction, nodes = "R", states = "small")
> querygrain(jres, nodes = "T")$T
T
    car    train   other
0.48389 0.41708 0.09903
```

$P(T \mid R=S)$

As shown in Figure 1.4, this reasoning is supported by the BN we saved in the bn object. The probability associated with other drops from 0.1573 to 0.099, while the probability associated with train increases from 0.2808 to 0.4170.

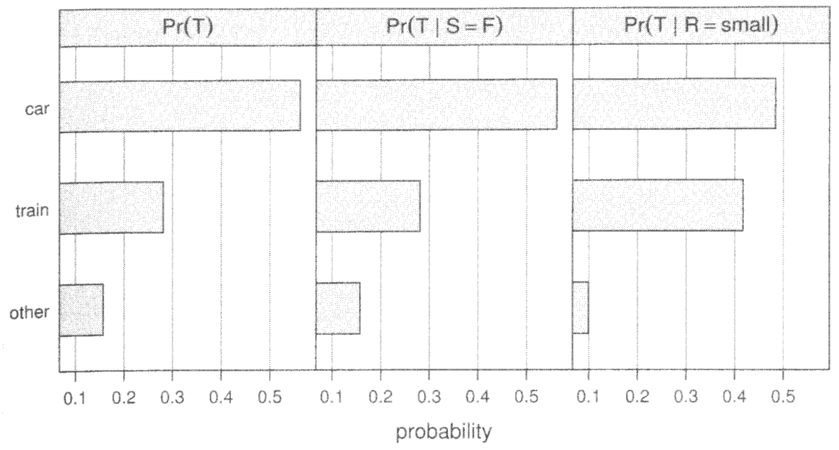


Figure 1.4
Probability distribution of Travel (T) given no evidence (left panel), given evidence that Sex (S) is equal to F (central panel) and given that Residence (R) is equal to small (right panel).

Overall, the combined probability of car and train increases from 0.8426 (for the whole survey sample) to 0.9009 (for people living in small cities). Extending this query to provide the most likely explanation, we conclude that for people living in small cities the car is the preferred means of transport.

Conditional probability queries can also be used to assess conditional independence, as we previously did with graphical separation and the dsep function. Consider again the relationship between S and T, this time conditioning on the evidence that E is equal to high. The joint probability distribution of S and T given E, Pr(S, T | E = high), can be computed using setEvidence and querygrain as follows.

```
> jedu <- setEvidence(junction, nodes = "E", states = "high")
> SxT.cpt <- querygrain(jedu, nodes = c("S", "T"),
+               type = "joint")
> SxT.cpt
  T
S     car  train   other
  M 0.3427 0.1737 0.09623
  F 0.2167 0.1098 0.06087
```

$P(S,T \mid E=high)$

The argument type in querygrain specifies which of the possible distributions involving the nodes is returned. The default value is "marginal", for the marginal distribution of each node.

```
> querygrain(jedu, nodes = c("S", "T"), type = "marginal")
$S
S
     M      F
0.6126 0.3874

$T
T
   car  train  other
0.5594 0.2835 0.1571
```

As we have seen above, another possible choice is "joint", for the joint distribution of the nodes. The last valid value is "conditional". In this case querygrain returns the distribution of the first node in nodes conditional on the other nodes in nodes (and, of course, on the evidence we specified with setEvidence).

```
> querygrain(jedu, nodes = c("S", "T"), type = "conditional")
  T
S       car  train  other
  M 0.6126 0.6126 0.6126
  F 0.3874 0.3874 0.3874
```

distrib of S

conditional on T

Note how the probabilities in each column sum up to 1, as they are computed conditional on the value T assumes in that particular column.

Furthermore, we can also see that all the conditional probabilities

$$\Pr(\mathtt{S} = \mathtt{M} \mid \mathtt{T} = t, \mathtt{E} = \mathtt{high}), \qquad t \in \{\mathtt{car}, \mathtt{train}, \mathtt{other}\} \tag{1.18}$$

are identical, regardless of the value of T we are conditioning on, and the same holds when S is equal to F. In other words,

$$\Pr(\mathtt{S} = \mathtt{M} \mid \mathtt{T} = t, \mathtt{E} = \mathtt{high}) = \Pr(\mathtt{S} = \mathtt{M} \mid \mathtt{E} = \mathtt{high}) \tag{1.19}$$

and

$$\Pr(\mathtt{S} = \mathtt{F} \mid \mathtt{T} = t, \mathtt{E} = \mathtt{high}) = \Pr(\mathtt{S} = \mathtt{F} \mid \mathtt{E} = \mathtt{high}) \tag{1.20}$$

This suggests that S is independent from T conditional on E; knowing the Sex of a person is not informative of his preferences if we know his Education. This is also implied by graphical separation, since S and T are d-separated by E.

```
> dsep(bn, x = "S", y = "T", z = "E")
[1] TRUE
```

Another way of confirming this conditional independence is to use the joint distribution of S and T we stored in SxT.cpt and perform a Pearson's X^2 test for independence. First, we multiply each entry of SxT.cpt by the sample size to convert the conditional probability table into a contingency table.

```
> SxT.ct = SxT.cpt * nrow(survey)
```

Each row in survey corresponds to one observation, so nrow(survey) is effectively the size of the sample. Pearson's X^2 test is implemented in the function chisq.test from package **stats**, which is included in the base R distribution.

```
> chisq.test(SxT.ct)
        Pearson's Chi-squared test

data:  SxT.ct
X-squared = 0, df = 2, p-value = 1
```

As expected, we accept the null hypothesis of independence, since the p-value of the test is exactly 1.

1.6.2.2 Approximate Inference

An alternative approach to inference is to use Monte Carlo simulations to randomly generate observations from the BN. In turn, we use these observations to compute approximate estimates of the conditional probabilities we are interested in. While this approach is computationally expensive, it allows for complex specifications of the evidence and scales better to BNs including a large number of nodes.

For discrete BNs, a simple way to implement approximate inference is to use *rejection sampling*. In rejection sampling, we generate random independent observations from the BN. Then we count how many match the evidence we are conditioning on and how many of those observations also match the event whose probability we are computing; the estimated conditional probability is the ratio between the latter and the former.

This approach is implemented in **bnlearn** in the cpquery and cpdist functions. cpquery returns the probability of a specific event given some evidence; so, for example, we can recompute the value of the first cell of the SxT table as follows.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+             evidence = (E == "high"))
[1] 0.3448
```

Note that the estimated conditional probability differs slightly from the exact value computed by querygrain, which is $\Pr(\mathtt{S} = \mathtt{M}, \mathtt{T} = \mathtt{car} \mid \mathtt{E} = \mathtt{high}) = 0.3427$. The quality of the approximation can be improved using the argument n to increase the number of random observations from the default 5000 * nparams(bn) to one million.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+               evidence = (E == "high"), n = 10^6)
[1] 0.343
```

The estimated probability is closer to its true value. However, increasing precision in this way has several drawbacks: answering the query takes much longer, and the precision may still be low if `evidence` has a low probability.

A better approach is *likelihood weighting*, which will be explained in detail in Section 4.6.2. Likelihood weighting generates random observations in such a way that all of them match the `evidence`, and re-weights them appropriately when computing the conditional probability for the query. It can be accessed from `cpquery` by setting `method = "lw"`.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+             evidence = list(E = "high"), method = "lw")
[1] 0.3421
```

As we can see, `cpquery` returned a conditional probability (0.3421) that is very close to the exact value (0.3427) without generating 10^6 random observations in the process. Unlike rejection sampling, which is the default for both `cpdist` and `cpquery`, `evidence` for likelihood weighting is provided by a list of values, one for each conditioning variable.

As an example of a more complex query, we can also compute

$$\Pr(\mathtt{S} = \mathtt{M}, \mathtt{T} = \mathtt{car} \mid \{\mathtt{A} = \mathtt{young}, \mathtt{E} = \mathtt{uni}\} \cup \{\mathtt{A} = \mathtt{adult}\}), \tag{1.21}$$

the probability of a man travelling by car given that his Age is `young` and his Education is `uni` or that he is an `adult`, regardless of his Education.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+   evidence = ((A == "young") & (E == "uni")) | (A == "adult"))
[1] 0.3337
```

The implementation of likelihood weighting in `cpquery` is not flexible enough to compute a query with composite evidence like the above; in that respect it shares the same limitations as the functions in the **gRain** package.

`cpdist`, which has a syntax similar to `cpquery`, returns a data frame containing the random observations for the variables in `nodes` that match `evidence`.

```
> SxT <- cpdist(bn, nodes = c("S", "T"),
+          evidence = (E == "high"))
> head(SxT)
  S     T
1 F   car
2 M   car
3 F train
4 M other
5 M other
6 M other
```

The observations contained in the data frame can then be used for any kind of inference, making this approach extremely versatile. For example, we can produce the probability table of `S` and `T` and compare it with that produced by `querygrain` on page 25. To do that, we first use `table` to produce a contingency table from the `SxT` data frame, and then `prop.table` to transform the counts into probabilities.

```
> prop.table(table(SxT))
   T
S      car  train  other
  M 0.3413 0.1764 0.0963
  F 0.2161 0.1089 0.0611
```

Again, we can extend conditional probability queries to produce the most likely explanation for `S` and `T` just by looking for the combination of their states that has the highest probability. As before, the answer is that among people whose Education is `high`, the most common Sex and Travel combination is male car drivers.

1.7 Plotting BNs

A key strength of BNs, and of graphical models in general, is the possibility of studying them through their graphical representations. Therefore, the ability of plotting a BN effectively is a key tool in BN inference.

1.7.1 Plotting DAGs

bnlearn uses the functionality implemented in the **Rgraphviz** package to plot graph structures, through the `graphviz.plot` function. If we call `graphviz.plot` without any argument other than the graph we want to plot, we obtain a DAG representation similar to that in Figure 1.1.

```
> graphviz.plot(dag)
```

`graphviz.plot` takes care of laying out nodes and arcs so as to minimise their overlap. By default, nodes are positioned so that parents are plotted above their children and that most arcs point downward. This layout is called `dot`. Other layouts can be specified with the `layout` argument; some examples are shown in Figure 1.5.

Highlighting particular nodes and arcs in a DAG, for instance to mark a path or the nodes involved in a particular query, can be achieved either with the `highlight` argument of `graphviz.plot` or using **Rgraphviz** directly. The former is easier to use, while the latter is more versatile.

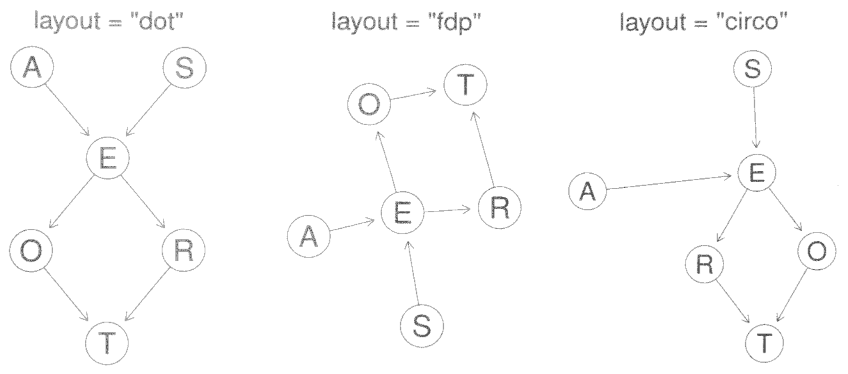


Figure 1.5
Some layouts implemented in **Rgraphviz** and available from `graphviz.plot`: `dot` (the default, on the left), `fdp` (centre) and `circo` (right).

Consider, for example, the left panel of Figure 1.3. All nodes and arcs with the exception of S → E → R are plotted in grey, to make the serial connection stand out. The node E, which d-separates S and R, is filled with a grey background to emphasise its role. To create such a plot, we need first to change the colour of all the nodes (including their labels) and the arcs to grey. To this end, we list all the nodes and all the arcs in a list called `hlight`, and we set their `col` and `textCol` to `grey`.

```
> hlight <- list(nodes = nodes(dag), arcs = arcs(dag),
+                    col = "grey", textCol = "grey")
```

Subsequently, we pass `hlight` to `graphviz.plot` via the `highlight` argument, and save the return value to make further changes to the plot.

```
> pp <- graphviz.plot(dag, highlight = hlight)
```

The `pp` object is an object of class `graph`, and it can be manipulated with the functions provided by the **graph** and **Rgraphviz** packages. The look of the arcs can be customised as follows using the `edgeRenderInfo` function from **Rgraphviz**.

```
> edgeRenderInfo(pp) <-
+   list(col = c("S~E" = "black", "E~R" = "black"),
+        lwd = c("S~E" = 3, "E~R" = 3))
```

Attributes being modified (*i.e.*, `col` for the colour and `lwd` for the line width) are specified again as the elements of a list. For each attribute, we specify a list containing the arcs we want to modify and the value to use for each of them. Arcs are identified by labels of the form *parent∼child*, *e.g.*, S → E is `S~E`.

Similarly, we can highlight nodes with `nodeRenderInfo`. We set their colour and the colour of the node labels to `black` and their background to `grey`.

```
> nodeRenderInfo(pp) <-
+   list(col = c("S" = "black", "E" = "black", "R" = "black"),
+     textCol = c("S" = "black", "E" = "black", "R" = "black"),
+     fill = c("E" = "grey"))
```

Once we have made all the desired modifications, we can plot the DAG again with the `renderGraph` function from **Rgraphviz**.

```
> renderGraph(pp)
```

More complicated plots can be created by repeated calls to `edgeRenderInfo` and `nodeRenderInfo`. These functions can be used to set several graphical parameters for each arc or node, and provide a fine-grained control on the appearance of the plot. Several (possibly overlapping) groups of nodes and arcs can be highlighted using different combinations of `lwd` (line width), `lty` (line type) and `col` (colour); or they can be hidden by setting `col` and `textCol` to a lighter colour or to `transparent`.

1.7.2 Plotting Conditional Probability Distributions

Plotting the conditional probabilities associated with a conditional probability table or a query is also useful for diagnostic and exploratory purposes. Such plots can be difficult to read when a large number of conditioning variables is involved, but nevertheless they provide useful insights for most synthetic and real-world data sets.

As far as conditional probability tables are concerned, **bnlearn** provides functions to plot barcharts (`bn.fit.barchart`) and dot plots (`bn.fit.dotplot`) from `bn.fit` objects. Both functions are based on the **lattice** package. So, for example, we can produce the plot in Figure 1.6 with

```
> bn.fit.barchart(bn.mle$T, main = "Travel",
+   xlab = "Pr(T | R,O)", ylab = "")
```

and the corresponding dot plot can be produced by calling `bn.fit.dotplot` with the same arguments. Each panel in the plot corresponds to one configuration of the levels of the parents of Travel: Occupation and Residence. Therefore, the plot is divided in four panels: {`O = self`, `R = big`}, {`O = self`, `R = small`}, {`O = emp`, `R = big`} and {`O = emp`, `R = small`}. The bars in each panel represent the probabilities for `car`, `train` and `other` conditional on the particular configuration of Occupation and Residence associated with the panel.

Both `bn.fit.barchart` and `bn.fit.dotplot` use the functionality provided by the **lattice** package, which implements a powerful and versatile

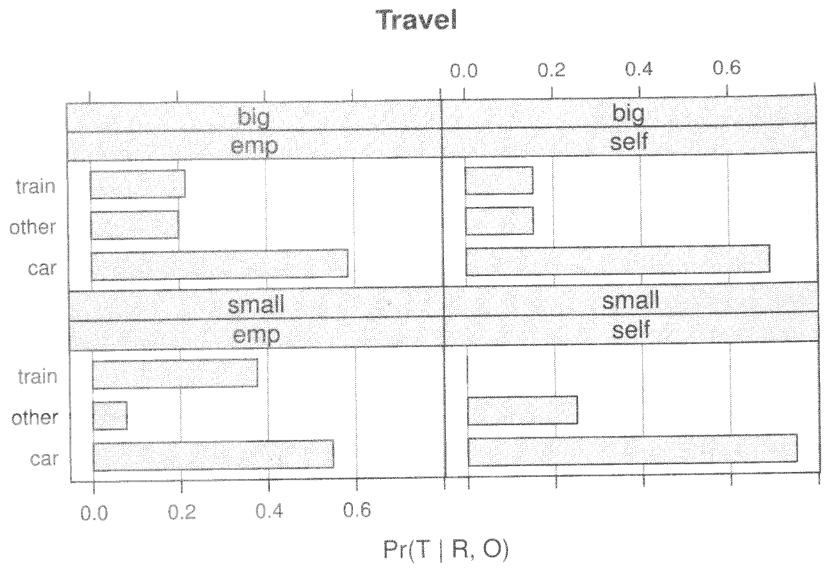


Figure 1.6
Barchart for the probability tables of Travel conditional on Residence and Occupation.

set of functions for multivariate data visualisation. As was the case for `graphviz.plot` and **Rgraphviz**, we can use the **lattice** functions directly to produce complex plots that are beyond the capabilities of **bnlearn**.

Consider, for example, the comparison between the marginal distribution of Travel and the results of the two conditional probability queries shown in Figure 1.4. That plot can be created using the `barchart` function from **lattice** in two simple steps. First, we need to create a data frame containing the three probability distributions.

```
> Evidence <-
+   factor(c(rep("Unconditional",3), rep("Female", 3),
+            rep("Small City",3)),
+          levels = c("Unconditional", "Female", "Small City"))
> Travel <- factor(rep(c("car", "train", "other"), 3),
+            levels = c("other", "train", "car"))
> distr <- data.frame(Evidence = Evidence, Travel = Travel,
+            Prob = c(0.5618, 0.2808, 0.15730, 0.5620, 0.2806,
+                     0.1573, 0.4838, 0.4170, 0.0990))
```

Each row of `distr` contains one probability (`Prob`), the level of Travel it refers to (`Travel`) and the evidence the query is conditioned on (`Evidence`).

```
> head(distr)
       Evidence Travel  Prob
1 Unconditional    car 0.562
2 Unconditional  train 0.281
3 Unconditional  other 0.157
4        Female    car 0.562
5        Female  train 0.281
6        Female  other 0.157
```

Once the probabilities have been organised in this way, the barchart in Figure 1.4 can be created as follows.

```
> barchart(Travel ~ Prob | Evidence, data = distr,
+    layout = c(3, 1), xlab = "probability",
+    scales = list(alternating = 1, tck = c(1, 0)),
+    strip = strip.custom(factor.levels =
+      c(expression(Pr(T)),
+        expression(Pr({T} * " | " * {S == F})),
+        expression(Pr({T} * " | " * {R == small})))),
+    panel = function(...) {
+      panel.barchart(...)
+      panel.grid(h = 0, v = -1)
+    })
```

As can be seen from the formula, we are plotting `Prob` for each level of `Travel` given the `Evidence`. For readability, we substitute the label of each panel with an `expression` describing the probability distribution corresponding to that panel. Furthermore, we lay the panels in a single row (with `layout`), move all the axes ticks at the bottom of the plot (with `scales`) and draw a grid over each panel (with the call to `panel.grid` in the function passed to the `panel` argument).

1.8 Further Reading

Discrete BNs are the most common type of BN studied in literature; all the books mentioned in the "Further Reading" sections of this book cover them in detail. Pearl (1988) and Castillo et al. (1997) both explore d-separation in depth. Koller and Friedman (2009, Chapter 17), Korb and Nicholson (2004, Chapter 6) and Neapolitan (2003, Section 7.1) cover parameter learning; Korb and Nicholson (2004, Chapter 9) and Murphy (2012, Section 16.4) cover structure learning.

Exercises

Exercise 1.1 *Consider the DAG for the survey studied in this chapter and shown in Figure 1.1.*

1. *List the parents and the children of each node.*
2. *List all the fundamental connections present in the DAG, and classify them as either serial, divergent or convergent.*
3. *Add an arc from Age to Occupation, and another arc from Travel to Education. Is the resulting graph still a valid BN? If not, why?*

Exercise 1.2 *Consider the probability distribution from the survey in Section 1.3.*

1. *Compute the number of configurations of the parents of each node.*
2. *Compute the number of parameters of the local distributions.*
3. *Compute the number of parameters of the global distribution.*
4. *Add an arc from Education to Travel. Recompute the factorisation into local distributions shown in Equation (1.1). How does the number of parameters of each local distribution change?*

Exercise 1.3 *Consider again the DAG for the survey.*

1. *Create an object of class* `bn` *for the DAG.*
2. *Use the functions in* **bnlearn** *and the R object created in the previous point to extract the nodes and the arcs of the DAG. Also extract the parents and the children of each node.*
3. *Print the model formula from* `bn`.
4. *Fit the parameters of the network from the data stored in* `survey.txt` *using their Bayesian estimators and save the result into an object of class* `bn.fit`.
5. *Remove the arc from Education to Occupation.*
6. *Fit the parameters of the modified network. Which local distributions change, and how?*

Exercise 1.4 *Re-create the* `bn.mle` *object used in Section 1.4.*

1. *Compare the distribution of Occupation conditional on Age with the corresponding marginal distribution using* `querygrain`.
2. *How many random observations are needed for* `cpquery` *to produce estimates of the parameters of these two distributions with a precision of* ± 0.01*?*

3. *Use the functions in* **bnlearn** *to extract the DAG from* `bn.mle`.
4. *Which nodes d-separate Age and Occupation?*

Exercise 1.5 *Implement an R function for BN inference via rejection sampling using the description provided in Section 1.4 as a reference.*

Exercise 1.6 *Using the* `dag` *and* `bn` *objects from Sections 1.2 and 1.3:*

1. *Plot the DAG using* `graphviz.plot`.
2. *Plot the DAG again, highlighting the nodes and the arcs that are part of one or more v-structures.*
3. *Plot the DAG one more time, highlighting the path leading from Age to Occupation.*
4. *Plot the conditional probability table of Education.*
5. *Compare graphically the distributions of Education for male and female interviewees.*