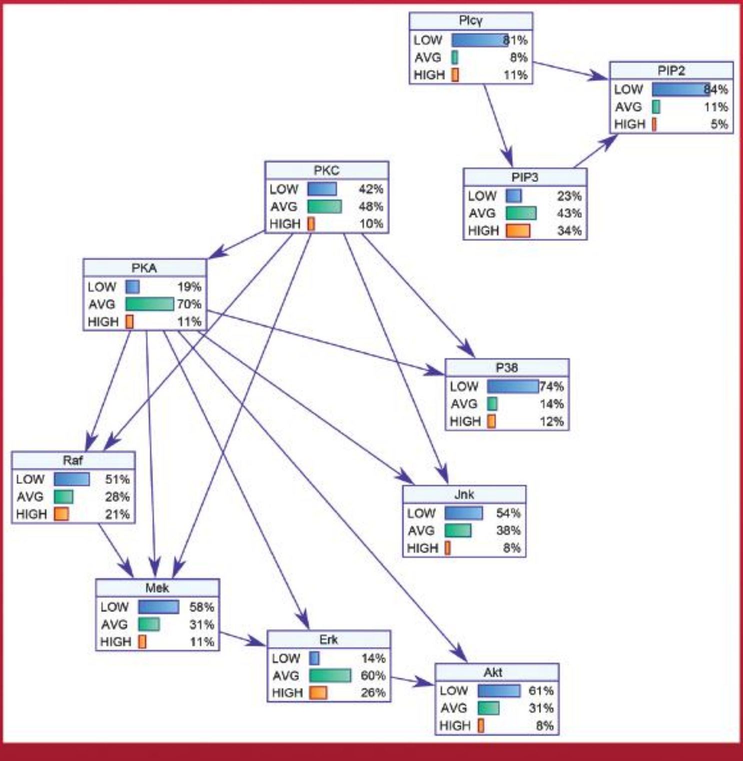
Texts in Statistical Science

Bayesian Networks

With Examples in R



Marco Scutari
Jean-Baptiste Denis

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Bayesian Networks

With Examples in R

CHAPMAN & HALL/CRC
Texts in Statistical Science Series

Statistical Theory: A Concise Introduction
F. Abramovich and Y. Ritov

Practical Multivariate Analysis, Fifth Edition
A. Afifi, S. May, and V.A. Clark

Practical Statistics for Medical Research
D.G. Altman

Interpreting Data: A First Course in Statistics
A.J.B. Anderson

Introduction to Probability with R
K. Baclawski

Linear Algebra and Matrix Analysis for Statistics
S. Banerjee and A. Roy

Statistical Methods for SPC and TQM
D. Bissell

Bayesian Methods for Data Analysis, Third Edition
B.P. Carlin and T.A. Louis

Second Edition
R. Caulcutt

The Analysis of Time Series: An Introduction, Sixth Edition
C. Chatfield

Introduction to Multivariate Analysis
C. Chatfield and A.J. Collins

Problem Solving: A Statistician's Guide, Second Edition
C. Chatfield

Statistics for Technology: A Course in Applied Statistics, Third Edition
C. Chatfield

Bayesian Ideas and Data Analysis: An Introduction for Scientists and Statisticians
R. Christensen, W. Johnson, A. Branscum, and T.E. Hanson

Modelling Binary Data, Second Edition
D. Collett

Modelling Survival Data in Medical Research, Second Edition
D. Collett

Introduction to Statistical Methods for Clinical Trials
T.D. Cook and D.L. DeMets

Applied Statistics: Principles and Examples
D.R. Cox and E.J. Snell

Multivariate Survival Analysis and Competing Risks
M. Crowder

Statistical Analysis of Reliability Data
M.J. Crowder, A.C. Kimber, T.J. Sweeting, and R.L. Smith

An Introduction to Generalized Linear Models, Third Edition
A.J. Dobson and A.G. Barnett

Nonlinear Time Series: Theory, Methods, and Applications with R Examples
R. Douc, E. Moulines, and D.S. Stoffer

Introduction to Optimization Methods and Their Applications in Statistics
B.S. Everitt

Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models
J.J. Faraway

Linear Models with R, Second Edition
J.J. Faraway

A Course in Large Sample Theory
T.S. Ferguson

Multivariate Statistics: A Practical Approach
B. Flury and H. Riedwyl

Readings in Decision Analysis
S. French

Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, Second Edition
D. Gamerman and H.F. Lopes

Bayesian Data Analysis, Third Edition
A. Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin

Multivariate Analysis of Variance and Repeated Measures: A Practical Approach for Behavioural Scientists
D.J. Hand and C.C. Taylor

Practical Data Analysis for Designed Practical Longitudinal Data Analysis
D.J. Hand and M. Crowder

Logistic Regression Models
J.M. Hilbe

Richly Parameterized Linear Models: Additive, Time Series, and Spatial Models Using Random Effects
J.S. Hodges

Statistics for Epidemiology
N.P. Jewell

Stochastic Processes: An Introduction, Second Edition
P.W. Jones and P. Smith

The Theory of Linear Models
B. Jørgensen

Principles of Uncertainty
J.B. Kadane

Graphics for Statistics and Data Analysis with R
K.J. Keen

Mathematical Statistics
K. Knight

Introduction to Multivariate Analysis: Linear and Nonlinear Modeling
S. Konishi

Nonparametric Methods in Statistics with SAS Applications
O. Korosteleva

Modeling and Analysis of Stochastic Systems, Second Edition
V.G. Kulkarni

Exercises and Solutions in Biostatistical Theory
L.L. Kupper, B.H. Neelon, and S.M. O'Brien

Exercises and Solutions in Statistical Theory
L.L. Kupper, B.H. Neelon, and S.M. O'Brien

Design and Analysis of Experiments with SAS
J. Lawson

A Course in Categorical Data Analysis
T. Leonard

Statistics for Accountants
S. Letchford

Introduction to the Theory of Statistical Inference
H. Liero and S. Zwanzig

Statistical Theory, Fourth Edition
B.W. Lindgren

Stationary Stochastic Processes: Theory and Applications
G. Lindgren

The BUGS Book: A Practical Introduction to Bayesian Analysis
D. Lunn, C. Jackson, N. Best, A. Thomas, and D. Spiegelhalter

Introduction to General and Generalized Linear Models
H. Madsen and P. Thyregod

Time Series Analysis
H. Madsen

Pólya Urn Models
H. Mahmoud

Randomization, Bootstrap and Monte Carlo Methods in Biology, Third Edition
B.F.J. Manly

Introduction to Randomized Controlled Clinical Trials, Second Edition
J.N.S. Matthews

Statistical Methods in Agriculture and Experimental Biology, Second Edition
R. Mead, R.N. Curnow, and A.M. Hasted

Statistics in Engineering: A Practical Approach
A.V. Metcalfe

Beyond ANOVA: Basics of Applied Statistics
R.G. Miller, Jr.

A Primer on Linear Models
J.F. Monahan

Applied Stochastic Modelling, Second Edition
B.J.T. Morgan

Elements of Simulation
B.J.T. Morgan

Probability: Methods and Measurement
A. O'Hagan

Introduction to Statistical Limit Theory
A.M. Polansky

Applied Bayesian Forecasting and Time Series Analysis
A. Pole, M. West, and J. Harrison

Statistics in Research and Development, Time Series: Modeling, Computation, and Inference
R. Prado and M. West

Introduction to Statistical Process Control
P. Qiu

Sampling Methodologies with Applications
P.S.R.S. Rao

A First Course in Linear Model Theory
N. Ravishanker and D.K. Dey

Essential Statistics, Fourth Edition
D.A.G. Rees

Stochastic Modeling and Mathematical Statistics: A Text for Statisticians and Quantitative
F.J. Samaniego

Statistical Methods for Spatial Data Analysis
O. Schabenberger and C.A. Gotway

Bayesian Networks: With Examples in R
M. Scutari and J.-B. Denis

Large Sample Methods in Statistics
P.K. Sen and J. da Motta Singer

Decision Analysis: A Bayesian Approach
J.Q. Smith

Analysis of Failure and Survival Data
P. J. Smith

Applied Statistics: Handbook of GENSTAT Analyses
E.J. Snell and H. Simpson

Applied Nonparametric Statistical Methods, Fourth Edition
P. Sprent and N.C. Smeeton

Data Driven Statistical Methods
P. Sprent

Generalized Linear Mixed Models: Modern Concepts, Methods and Applications
W. W. Stroup

Survival Analysis Using S: Analysis of Time-to-Event Data
M. Tableman and J.S. Kim

Applied Categorical and Count Data Analysis
W. Tang, H. He, and X.M. Tu

Elementary Applications of Probability Theory, Second Edition
H.C. Tuckwell

Introduction to Statistical Inference and Its Applications with R
M.W. Trosset

Understanding Advanced Statistical Methods
P.H. Westfall and K.S.S. Henning

Statistical Process Control: Theory and Practice, Third Edition
G.B. Wetherill and D.W. Brown

Generalized Additive Models: An Introduction with R
S. Wood

Epidemiology: Study Design and Data Analysis, Third Edition
M. Woodward

Experiments
B.S. Yandell

Texts in Statistical Science

Bayesian Networks

With Examples in R

Marco Scutari
UCL Genetics Institute (UGI)
London, United Kingdom

Jean-Baptiste Denis
Unité de Recherche Mathématiques et Informatique Appliquées, INRA,
Jouy-en-Josas, France

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20140514

International Standard Book Number-13: 978-1-4822-2559-4 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

To my family

To my wife, Jeanie Denis

Contents

Preface

Applications of Bayesian networks have multiplied in recent years, spanning such different topics as systems biology, economics, social sciences and medical informatics. Different aspects and properties of this class of models are crucial in each field: the possibility of learning causal effects from observational data in social sciences, where collecting experimental data is often not possible; the intuitive graphical representation, which provides a qualitative understanding of pathways in biological sciences; the ability to construct complex hierarchical models for phenomena that involve many interrelated components, using the most appropriate probability distribution for each of them. However, all these capabilities are built on the solid foundations provided by a small set of core definitions and properties, on which we will focus for most of the book. Handling high-dimensional data and missing values, the fine details of causal reasoning, learning under sets of additional assumptions specific to a particular field, and other advanced topics are beyond the scope of this book. They are thoroughly explored in monographs such as Nagarajan et al. (2013), Pourret et al. (2008) and Pearl (2009).

The choice of the R language is motivated, likewise, by its increasing popularity across different disciplines. Its main shortcoming is that R only provides a command-line interface, which comes with a fairly steep learning curve and is intimidating to practitioners of disciplines in which computer programming is not a core topic. However, once mastered, R provides a very versatile environment for both data analysis and the prototyping of new statistical methods. The availability of several contributed packages covering various aspects of Bayesian networks means that the reader can explore the contents of this book without reimplementing standard approaches from literature. Among these packages, we focus mainly on **bnlearn** (written by the first author, at version 3.5 at the time of this writing) to allow the reader to concentrate on studying Bayesian networks without having to first figure out the peculiarities of each package. A much better treatment of their capabilities is provided in Højsgaard et al. (2012) and in the respective documentation resources, such as vignettes and reference papers.

Bayesian Networks: With Examples in R aims to introduce the reader to Bayesian networks using a hands-on approach, through simple yet meaningful examples explored with the R software for statistical computing. Indeed, being *hands-on* is a key point of this book, in that the material strives to detail each modelling step in a simple way and with supporting R code. We know very well that a number of good books are available on this topic, and we referenced them in the "Further Reading" sections at the end of each chapter. However, we feel that the way we chose to present the material is different and that it makes this book suitable for a first introductory overview of Bayesian networks. At the same time, it may also provide a practical way to use, thanks to R, such a versatile class of models.

We hope that the book will also be useful to non-statisticians working in very different fields. Obviously, it is not possible to provide worked-out examples covering every field in which Bayesian networks are relevant. Instead, we prefer to give a clear understanding of the general approach and of the steps it involves. Therefore, we explore a limited number of examples in great depth, considering that experts will be able to reinterpret them in the respective fields. We start from the simplest notions, gradually increasing complexity in later chapters. We also distinguish the probabilistic models from their estimation with data sets: when the separation is not clear, confusion is apparent when performing inference.

Bayesian Networks: With Examples in R is suitable for teaching in a semester or half-semester course, possibly integrating other books. More advanced theoretical material and the analysis of two real-world data sets are included in the second half of the book for further understanding of Bayesian networks. The book is targeted at the level of a M.Sc. or Ph.D. course, depending on the background of the student. In the case of disciplines such as mathematics, statistics and computer science the book is suitable for M.Sc. courses, while for life and social sciences the lack of a strong grounding in probability theory may make the book more suitable for a Ph.D. course. In the former, the reader may prefer to first review the second half of the book, to grasp the theoretical aspects of Bayesian networks before applying them; while in the latter he can get a hang of what Bayesian networks are about before investing time in studying their underpinnings. Introductory material on probability, statistics and graph theory is included in the appendixes. Furthermore, the solutions to the exercises are included in the book for the convenience of the reader. The real-world examples in the last chapter will motivate students by showing current applications in the literature. Introductory examples in earlier chapters are more varied in topic, to present simple applications in different contexts.

The skills required to understand the material are mostly at the level of a B.Sc. graduate. Nevertheless, a few topics are based on more specialised concepts whose illustration is beyond the scope of this book. The basics of R programming are not covered in the book, either, because of the availability of accessible and thorough references such as Venables and Ripley (2002), Spector (2009) and Crawley (2013). Basic graph and probability theory are covered in the appendixes for easy reference. Pointers to literature are provided at the end of each chapter, and supporting material will be available online from `www.bnlearn.com`.

The book is organised as follows. Discrete Bayesian networks are described first (Chapter 1), followed by Gaussian Bayesian networks (Chapter 2). Hybrid networks (which include arbitrary random variables, and typically mix continuous and discrete ones) are covered in Chapter 3. These chapters explain the whole process of Bayesian network modelling, from structure learning to parameter learning to inference. All steps are illustrated with R code. A concise but rigorous treatment of the fundamentals of Bayesian networks is given in Chapter 4, and includes a brief introduction to causal Bayesian networks. For completeness, we also provide an overview of the available software in Chapter 5, both in R and other software packages. Subsequently, two real-world examples are analysed in Chapter 6. The first replicates the study in the landmark causal protein-signalling network paper published in *Science* by Sachs et al. (2005). The second investigates possible graphical modelling approaches in predicting the contributions of fat, lean and bone to the composition of different body parts.

Last but not least, we are immensely grateful to friends and colleagues who helped us in planning and writing this book, and its French version *Résaux Bayésiens avec R: élaboration, manipulation et utilisation en modélisation appliquée.* We are also grateful to John Kimmel of Taylor & Francis for his dedication in improving this book and organising draft reviews. We hope not to have unduly raised his stress levels, as we did our best to incorporate the reviewers' feedback and we even submitted the final manuscript on time. Likewise, we thank the people at EDP Sciences for their interest in publishing a book on this topic: they originally asked the second author to write a book in French. He was not confident enough to write a book alone and looked for a co-author, thus starting the collaboration with the first author and a wonderful exchange of ideas. The latter, not being very proficient in the French language, prepared the English draft from which this Chapman & Hall book originates. The French version is also planned to be in print by the end of this year.

London, United Kingdom *Marco Scutari*
Jouy-en-Josas, France *Jean-Baptiste Denis*
March 2014

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.

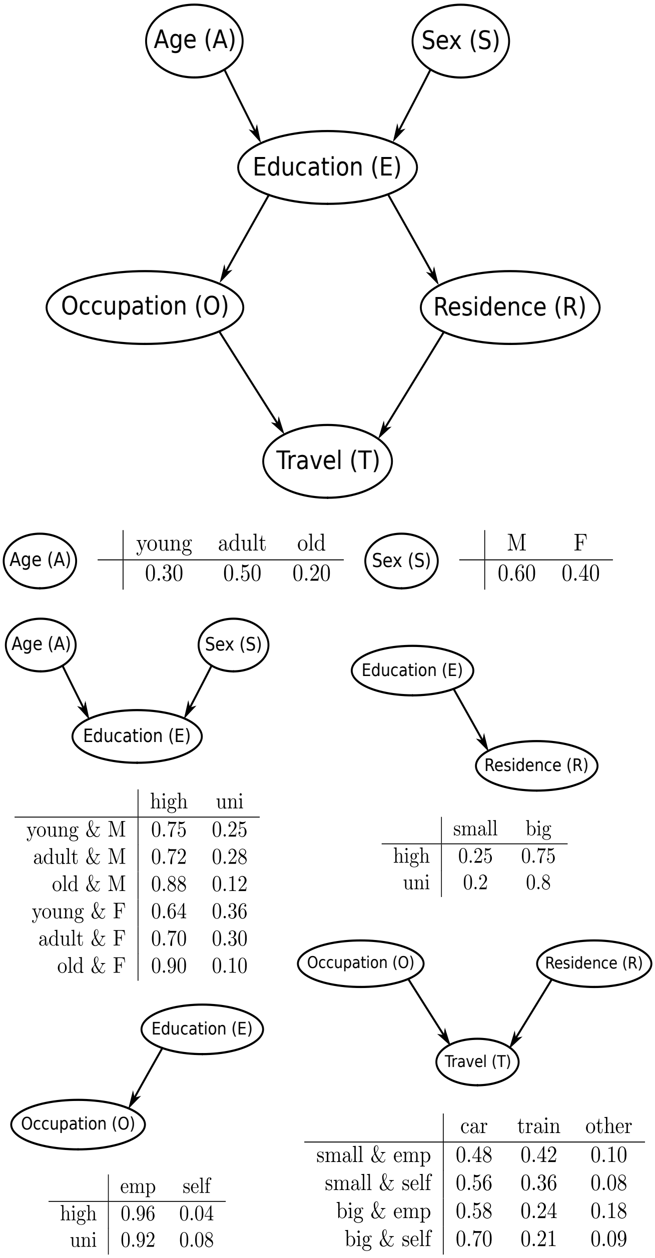
```
> dag
  Random/Generated Bayesian network

  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]"
```



Age (A)	young	adult	old
	0.30	0.50	0.20

Sex (S)	M	F
	0.60	0.40

	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
```

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
```

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.

```
> 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))
```

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.*,

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

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

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

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

and we denote the corresponding prior probabilities as

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

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

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

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

and therefore that

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

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.

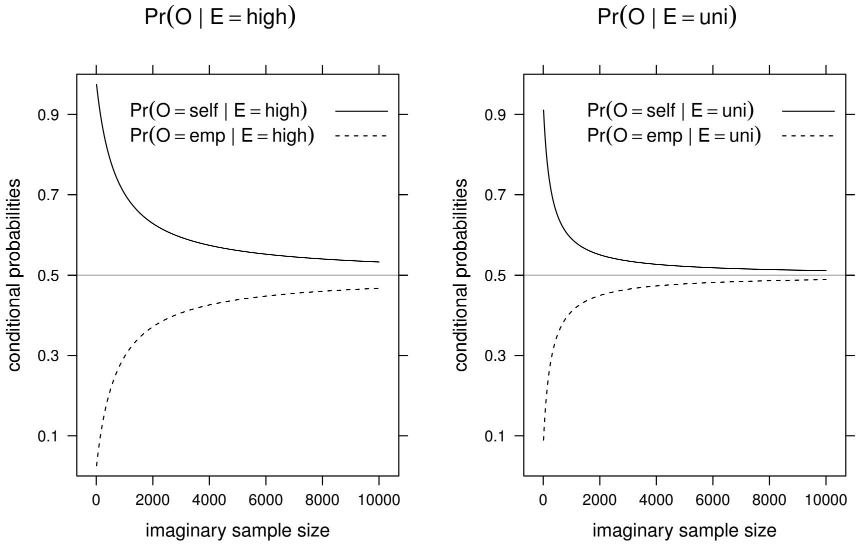


Figure 1.2
Conditional probability distributions for O given both possible values of E, that is, Pr(O | E = high) and Pr(O | E = uni), converge to uniform distributions as the imaginary sample size increases.

```
> 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

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 ($\mathtt{E} \rightarrow \mathtt{T}$) 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 E × 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 O → 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 O × 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 O → 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

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

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 $\log \mathrm{BDe}$ 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, $\log \mathrm{BDe}$ 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 `E` → `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 E → 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.

```
> 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

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 $\mathbf{X}$ and $\mathbf{Y}$ of variables are independent given a third set $\mathbf{Z}$ of variables if there is no set of arcs connecting them that is not *blocked* by the conditioning variables. Conditioning on $\mathbf{Z}$ is equivalent to *fixing* the values of its elements, so that they are known quantities. In other words, the $\mathbf{X}$ and $\mathbf{Y}$ are *separated* by $\mathbf{Z}$, 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 $\mathbf{X}$ and $\mathbf{Y}$ are blocked, $\mathbf{X}$ and $\mathbf{Y}$ 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 $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$; 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:

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

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

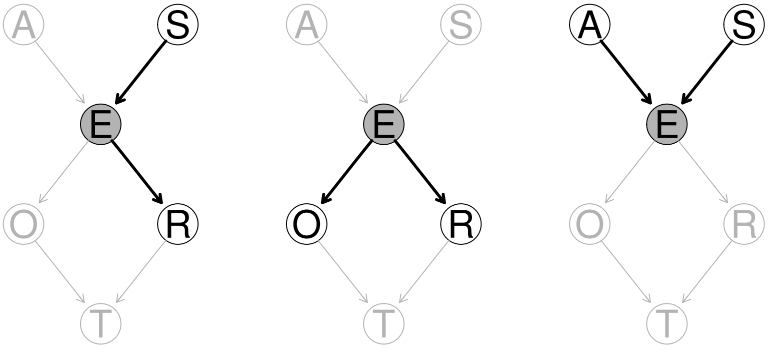


Figure 1.3
Some examples of d-separation covering the three fundamental connections: the *serial connection* (left), the *divergent connection* (centre) and the *convergent connection* (right). Nodes in the conditioning set are highlighted in grey.

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

Again, from Equation (1.1) we have

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

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 $\Pr(\texttt{E} \mid \texttt{A}, \texttt{S})$; then using Bayes' theorem we have

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

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 $\Pr(\texttt{A}, \texttt{S}) = \Pr(\texttt{A} \mid \texttt{S}) \Pr(\texttt{S}) = \Pr(\texttt{A}) \Pr(\texttt{S})$: 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:

- structures like $\texttt{S} \rightarrow \texttt{E} \rightarrow \texttt{R}$ (the first example) are known as *serial connections*, since both arcs have the same direction and follow one after the other;
- structures like $\texttt{R} \leftarrow \texttt{E} \rightarrow \texttt{O}$ (the second example) are known as *divergent connections*, because the two arcs have divergent directions from a central node;
- structures like $\texttt{A} \rightarrow \texttt{E} \leftarrow \texttt{S}$ (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 $\texttt{A} \rightarrow \texttt{S}$ nor $\texttt{A} \leftarrow \texttt{S}$) convergent connections are called *v-structures*. As we will see in Chapter 4, their properties are crucial in characterising and learning BNs.

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).

```
> 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.

```
> querygrain(junction, nodes = "T")$T
T
   car  train  other
0.5618 0.2809 0.1573
> jsex <- setEvidence(junction, nodes = "S", states = "F")
> querygrain(jsex, nodes = "T")$T
T
   car  train  other
0.5621 0.2806 0.1573
```

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
```

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.

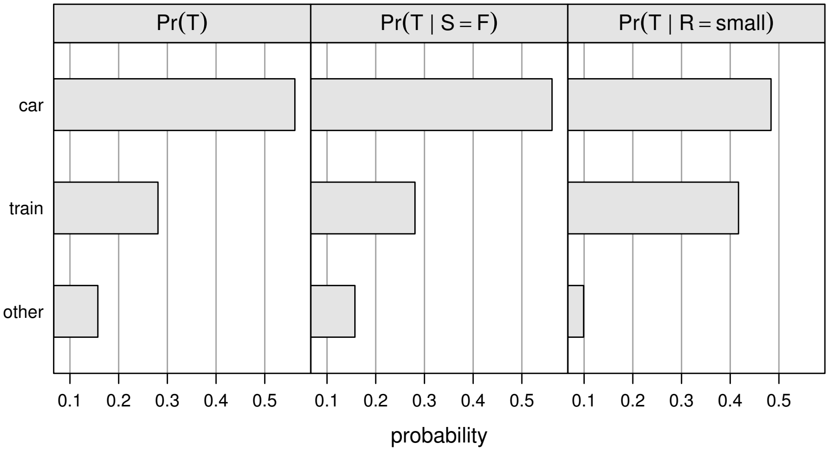


Figure 1.4
Probability distribution of Travel (`T`) given no evidence (left panel), given evidence that Sex (`S`) is equal to `F` (central panel) and given that Residence (`R`) is equal to `small` (right panel).

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`, $\Pr(\texttt{S}, \texttt{T} \mid \texttt{E} = \texttt{high})$, 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
```

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
```

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

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

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,

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

and

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

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.

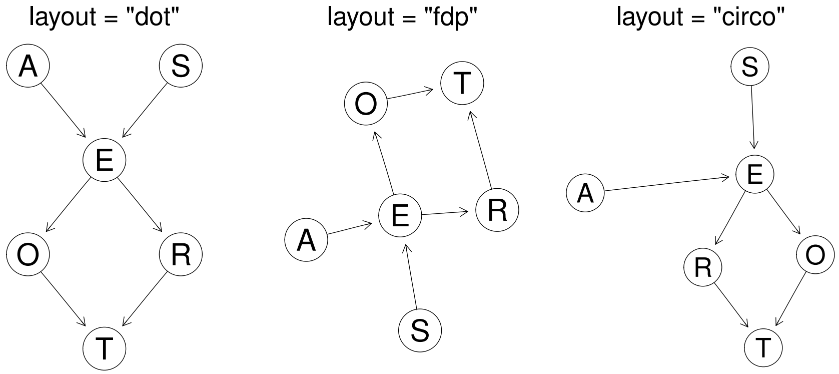


Figure 1.5
Some layouts implemented in **Rgraphviz** and available from `graphviz.plot`: `dot` (the default, on the left), `fdp` (centre) and `circo` (right).

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

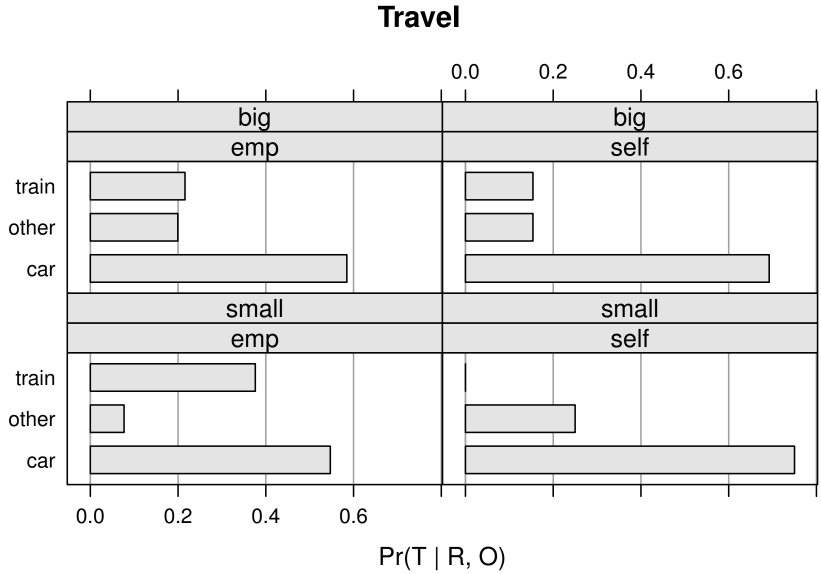


Figure 1.6
Barchart for the probability tables of Travel conditional on Residence and Occupation.

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.*

2

The Continuous Case: Gaussian Bayesian Networks

In this chapter we will continue our exploration of BNs, focusing on modelling continuous data under a multivariate Normal (Gaussian) assumption.

2.1 Introductory Example: Crop Analysis

Suppose that we are interested in the analysis of a particular plant, which we will model in a very simplistic way by considering:

- the potential of the plant and of the environment;
- the production of vegetative mass;
- and the harvested grain mass, which is called the *crop*.

To be more precise, we define two synthetic variables to describe the initial status of the plant: its *genetic potential*, which we will denote as G, and the *environmental potential* of the location and the season it is grown in, which we will denote as E. Both G and E are assumed to summarise in a single score all genotypic effects and all environmental effects, respectively; their composite nature justifies the use of continuous variables.

It is well known to farmers and plant breeders that the first step in evaluating a crop is analysing the vegetative organs. Root, stems and leaves grow and accumulate reserves which are later exploited for reproduction. In our model, we will consider a single vegetative variable summarising all the information available on constituted reserves. This variable will be denoted as V, and it will be modelled again as a continuous variable. Clearly, V is directly influenced by the values of G and E. The greater they are, the greater is the possibility of a large vegetative mass V.

As mentioned above, the crop is a function of the vegetative mass. They are related through various quantities, mainly the number of seeds and their mean weight. We will denote them with N and W, respectively. These two variables are not measured at the same time: N is determined at flowering time while W is obtained later in the plant's life. Finally, the crop, denoted by C, depends directly on N and W. All these three variables can again be naturally modelled as continuous variables.

As a result, the behaviour of a plant can be described by

$$\{\texttt{G},\texttt{E}\} \rightarrow \texttt{V}, \qquad \texttt{V} \rightarrow \texttt{N}, \qquad \texttt{V} \rightarrow \texttt{W}, \qquad \{\texttt{N},\texttt{W}\} \rightarrow \texttt{C}; \tag{2.1}$$

and we will use these relationships to model the crop with a BN.

2.2 Graphical Representation

A graphical representation of the relationships in Equation (2.1) is shown in Figure 2.1. The figure displays the six variables used in the BN (the nodes) and the six arcs corresponding to the direct dependencies linking them. Together they form a DAG that is very similar to that presented in Figure 1.1, differing only in the names of the variables. Indeed the DAG does not depend on the nature of the variables under consideration, because the same dependence structure can apply to many different situations.

To work on such a graph, we create an R object describing it. As shown in Section 1.7, we can use the `model2network` function in **bnlearn** and the model formula representation to express the relationships between the variables.

```
> library(bnlearn)
> dag.bnlearn <- model2network("[G][E][V|G:E][N|V][W|V][C|N:W]")
> dag.bnlearn
  Random/Generated Bayesian network

  model:
   [E][G][V|E:G][N|V][W|V][C|N:W]
  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
```

The structure of the DAG is defined by the string that is the argument of `model2network`; each node is reported in square brackets along with its parents. As noted in Section 1.2, the order in which the nodes and the parents of each node are given is not important.

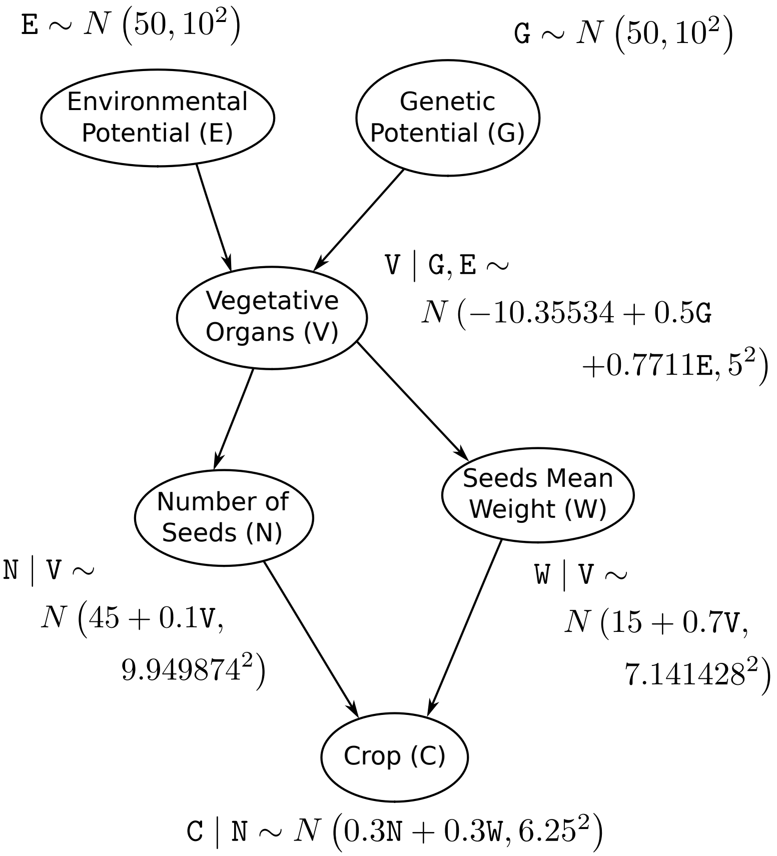

$\texttt{E} \sim N\left(50, 10^2\right)$
$\texttt{G} \sim N\left(50, 10^2\right)$
Environmental Potential (E)
Genetic Potential (G)
Vegetative Organs (V)
$\texttt{V} \mid \texttt{G}, \texttt{E} \sim N\left(-10.35534 + 0.5\texttt{G} + 0.7711\texttt{E}, 5^2\right)$
Number of Seeds (N)
Seeds Mean Weight (W)
$\texttt{N} \mid \texttt{V} \sim N\left(45 + 0.1\texttt{V}, 9.949874^2\right)$
$\texttt{W} \mid \texttt{V} \sim N\left(15 + 0.7\texttt{V}, 7.141428^2\right)$
Crop (C)
$\texttt{C} \mid \texttt{N} \sim N\left(0.3\texttt{N} + 0.3\texttt{W}, 6.25^2\right)$

Figure 2.1
DAG representing the crop (`C`) network, with variables environmental potential (`E`), genetic potential (`G`), vegetative organs (`V`), number of seeds (`N`) and their mean weight (`W`). The local probability distributions are shown for each node.

Some important properties of the BN can already be deduced from the DAG. For instance, in our example we assume that C depends on G because there are paths leading from G to C. From a probabilistic point of view, this means that the conditional distribution of C given $\mathtt{G} = g$ is a function of g, or equivalently

$$\mathrm{f}(\mathtt{C} \mid \mathtt{G} = g) \neq \mathrm{f}(\mathtt{C}), \tag{2.2}$$

where $\mathrm{f}(\mathtt{X})$ stands for the density function of the random variable X. Even though the relationship between C and G is motivated by causal reasoning, and therefore makes a distinction between cause G and effect C, this distinction is completely lost in probabilistic modelling. As a result, Equation (2.2) implies

$$\mathrm{f}(\mathtt{G} \mid \mathtt{C} = c) \neq \mathrm{f}(\mathtt{G}), \tag{2.3}$$

that is, the conditional distribution of G given C is a function of c, the value assumed by C.

More importantly, we have that C depends on G through V. This is a consequence of the fact that all paths going from G to C pass through V. In probabilistic terms, this is equivalent to

$$\mathrm{f}(\mathtt{C} \mid \mathtt{G} = g, \mathtt{V} = v) = \mathrm{f}(\mathtt{C} \mid \mathtt{V} = v). \tag{2.4}$$

In other words, knowing the value of G adds nothing to our knowledge about C when the value of V is known.

As was the case for discrete BNs, we can investigate conditional independencies. Consider, for example, the pair of variables N and W. Since they share a common parent, V, they are not independent. This is natural because they are influenced in similar ways by the state of vegetative organs: high values of V make high values of both N and W more likely. Nevertheless, when V is known to be equal to v, they become independent. From the decomposition into local distributions implied by the DAG, we can see the conditional independence:

$$\mathrm{f}(\mathtt{N}, \mathtt{W} \mid \mathtt{V} = v) = \mathrm{f}(\mathtt{N} \mid \mathtt{V} = v)\,\mathrm{f}(\mathtt{W} \mid \mathtt{V} = v). \tag{2.5}$$

This kind of reasoning generalises to arbitrary sets of variables, as we will see in Section 4.1. As underlined in Section 1.6.1, these properties do not depend on the nature of the random variables involved, nor on their probability distributions. We can investigate these independencies in a systematic way, whether they are conditional or not, using the dsep function from **bnlearn**.

First, we can find which pairs of variables are marginally independent.

```
> nano <- nodes(dag.bnlearn)
> for (n1 in nano) {
+   for (n2 in nano) {
+     if (dsep(dag.bnlearn, n1, n2))
+       cat(n1, "and", n2, "are independent.\n")
+   }#FOR
+ }#FOR
```

```
E and G are independent.
G and E are independent.
```

Of all possible pairs, only E and G are found to be independent, and as expected the independence is symmetric. Note that the two nested loops imply that n2 is identical to n1 in some tests and, quite logically, dsep returns FALSE in these cases.

```
> dsep(dag.bnlearn, "V", "V")
[1] FALSE
```

In addition, we can also find which pairs are conditionally independent given V, taking symmetry into account to avoid redundant checks.

```
> for (n1 in nano[nano != "V"]) {
+   for (n2 in nano[nano != "V"]) {
+     if (n1 < n2) {
+       if (dsep(dag.bnlearn, n1, n2, "V"))
+         cat(n1, "and", n2, "are independent given V.\n")
+     }#THEN
+   }#FOR
+ }#FOR
C and E are independent given V.
C and G are independent given V.
E and N are independent given V.
E and W are independent given V.
G and N are independent given V.
G and W are independent given V.
N and W are independent given V.
```

From the output above we can see that, conditional on the value of V, E and G are no longer independent. On the other hand, nodes belonging to pairs in which one node is *before* V in the graph and the other is *after* V are now conditionally independent. This is due to the d-separating property of V for such pairs of nodes: it descends from the fact that every path connecting two such variables passes through V. Furthermore, we can see that N and W are d-separated as stated in Equation (2.5).

Note that n1 or n2 are chosen so as to never be identical to V. However, dsep accepts such a configuration of nodes and returns TRUE in this case. Since V | V is a degenerate random variable that can assume a single value, it is independent from every random variable.

```
> dsep(dag.bnlearn, "E", "V", "V")
[1] TRUE
```

Many other operations involving the arcs and the nodes of a DAG may be of interest. One example is looking for a path going from one subset of nodes to another, which is one way of exploring how the latter depends on the former. For instance, we can use the `path` function to look for the existence of such a path between `E` and `C`.

```
> bnlearn::path(dag.bnlearn, from = "E", to = "C")
[1] TRUE
```

Note the use of the `bnlearn::` prefix to uniquely identify the `path` function. A function of the same name is provided by the **igraph** package, and it might be called by mistake depending on the order in which the two packages have been loaded. More operations will be considered in subsequent chapters as the relevant theory is introduced.

2.3 Probabilistic Representation

In order to make quantitative statements about the behaviour of the variables in the BN, we need to completely specify their joint probability distribution. This can be an extremely difficult task. In the framework of the multivariate normal distributions considered in this chapter, if we are modelling p variables we must specify p means, p variances and $\frac{1}{2}p\,(p-1)$ correlation coefficients. Furthermore, the correlation coefficients must be such that the resulting correlation matrix is non-negative definite. Fortunately, in the context of BNs we only need to specify the local distribution of each node conditional on the values of its parents, without worrying about the positive definiteness of the correlation matrix of the global distribution. In the case of the DAG shown in Figure 2.1, it means specifying 18 parameters instead of 27.

Suppose for the moment that the local distributions are known and reported in Table 2.1; the fundamental step of estimating the parameter values is postponed to Section 2.4. Of course the parents implicitly proposed by the conditional distributions are consistent with the DAG, and do not imply any cycle. We can than create a `bn` object for use with **bnlearn** in the same way we did in Section 1.3. First, we store the parameters of each local distribution in a list, here called `dis.list`.

```
> disE <- list(coef = c("(Intercept)" = 50), sd = 10)
> disG <- list(coef = c("(Intercept)" = 50), sd = 10)
> disV <- list(coef = c("(Intercept)" = -10.35534,
+                 E = 0.70711, G = 0.5), sd = 5)
> disN <- list(coef = c("(Intercept)" = 45, V = 0.1),
+                 sd = 9.949874)
```

$$\begin{aligned}
\texttt{G} &\sim N\left(50, 10^2\right)\\
\texttt{E} &\sim N\left(50, 10^2\right)\\
\texttt{V} \mid \texttt{G} = g, \texttt{E} = e &\sim N\left(-10.35534 + 0.5g + 0.70711e, 5^2\right)\\
\texttt{N} \mid \texttt{V} = v &\sim N\left(45 + 0.1v, 9.949874^2\right)\\
\texttt{W} \mid \texttt{V} = v &\sim N\left(15 + 07v, 7.141428^2\right)\\
\texttt{C} \mid \texttt{N} = n, \texttt{W} = w &\sim N\left(0.3n + 0.7w, 6.25^2\right)
\end{aligned}$$

Table 2.1
Probability distributions proposed for the DAG shown in Figure 2.1.

```
> disW <- list(coef = c("(Intercept)" = 15, V = 0.7),
+                 sd = 7.141428)
> disC <- list(coef = c("(Intercept)" = 0, N = 0.3, W = 0.7),
+                sd = 6.25)
> dis.list = list(E = disE, G = disG, V = disV, N = disN,
+                   W = disW, C = disC)
```

Comparing the R definition of each distribution and the corresponding mathematical expression in Table 2.1 elucidates the syntax used in code above.

Subsequently, we can call the `custom.fit` function to create an object of class `bn.fit` from the graph structure stored in `dag.bnlearn` and the local distributions in `dis.list`.

```
> gbn.bnlearn <- custom.fit(dag.bnlearn, dist = dis.list)
```

As was the case for multinomial BNs, printing `gbn.bnlearn` prints all local distributions and their parameters. For brevity, we show only the local distributions of `G` (a root node) and `C` (a node with two parents).

```
> gbn.bnlearn$G
  Parameters of node G (Gaussian distribution)

Conditional density: G
Coefficients:
(Intercept)
         50
Standard deviation of the residuals: 10
```

```
> gbn.bnlearn$C
  Parameters of node C (Gaussian distribution)

Conditional density: C | N + W
Coefficients:
(Intercept)            N            W
        0.0          0.3          0.7
Standard deviation of the residuals: 6.25
```

The network we just created rests upon the following assumptions, which characterise linear Gaussian Bayesian networks (GBNs):

- every node follows a normal distribution;
- nodes without any parent, known as *root* nodes, are described by the respective marginal distributions;
- the conditioning effect of the parent nodes is given by an additive linear term in the mean, and does not affect the variance. In other words, each node has a variance that is specific to that node and does not depend on the values of the parents;
- the local distribution of each node can be equivalently expressed as a Gaussian linear model which includes an intercept and the node's parents as explanatory variables, without any interaction term.

The distributions shown in Table 2.1 probably would not have been an obvious choice for an expert in crop production. Surely, a more complex specification would have been preferred. For instance, using multiplicative terms instead of additive ones in the expectation of $(\mathtt{C} \mid \mathtt{N} = n, \mathtt{W} = w)$ would have been more realistic. The use of linear dependencies is motivated by their good mathematical properties, most importantly their tractability and the availability of closed-form results for many inference procedures. More sophisticated models are certainly possible, and will be presented in Chapter 3. Nevertheless, it is important to note that for small variations any continuous function can be approximated by an additive function, *e.g.*, a first-order Taylor expansion. Furthermore, relatively simple models often perform better than more sophisticated ones when few observations are available.

In the following, we will supplement **bnlearn** with another R package, **rbmn** (short for "**r**éseaux **b**ayésiens **m**ulti**n**ormaux"), focusing specifically on GBNs. **rbmn** provides a function to convert `bn.fit` objects such as `gbn.bnlearn` in its own native format.

```
> library(rbmn)
> gbn.rbmn <- bnfit2nbn(gbn.bnlearn)
```

It can be shown that the properties assumed above for the local distributions imply that the joint distribution of all nodes (the global distribution) is multivariate normal. Using the decomposition introduced in Section 1.3, we can obtain it as the product of the local distributions:

$$\mathrm{f}(\mathtt{G},\mathtt{E},\mathtt{V},\mathtt{N},\mathtt{W},\mathtt{C}) = \mathrm{f}(\mathtt{G})\,\mathrm{f}(\mathtt{E})\,\mathrm{f}(\mathtt{V} \mid \mathtt{G},\mathtt{E})\,\mathrm{f}(\mathtt{N} \mid \mathtt{V})\,\mathrm{f}(\mathtt{W} \mid \mathtt{V})\,\mathrm{f}(\mathtt{C} \mid \mathtt{N},\mathtt{W}). \qquad (2.6)$$

The parameters of that multivariate normal distribution can be derived numerically as follows.

```
> gema.rbmn <- nbn2gema(gbn.rbmn)
> mn.rbmn <- gema2mn(gema.rbmn)
> print8mn(mn.rbmn)
  mu s.d.   C.E  C.G   C.V   C.W   C.N   C.C
E 50   10 1.000 0.00 0.707 0.495 0.071 0.368
G 50   10 0.000 1.00 0.500 0.350 0.050 0.260
V 50   10 0.707 0.50 1.000 0.700 0.100 0.520
W 50   10 0.495 0.35 0.700 1.000 0.070 0.721
N 50   10 0.071 0.05 0.100 0.070 1.000 0.349
C 50   10 0.368 0.26 0.520 0.721 0.349 1.000
```

The first column of the resulting matrix is the vector of the marginal expectations (here all equal to 50), the second contains the marginal standard deviations (here all equal to 10), and the remaining ones contain the correlation matrix. The reason behind the choice of these particular values for the parameters in Table 2.1 is now apparent: all marginal distributions have the same mean and variance. This will simplify the interpretation of the relationships between the variables in Section 2.6.

The `mn.rbmn` object is a simple list holding the expectation vector (`mu`) and the covariance matrix (`gamma`) resulting from the correlations and the variances printed above.

```
> str(mn.rbmn);
List of 2
 $ mu   : Named num [1:6] 50 50 50 50 50 ...
  ..- attr(*, "names")= chr [1:6] "E" "G" "V" "W" ...
 $ gamma: num [1:6, 1:6] 100 0 70.71 49.5 7.07 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:6] "E" "G" "V" "W" ...
  .. ..$ : chr [1:6] "E" "G" "V" "W" ...
```

Both `mu` and `gamma` have their dimensions names set to the node labels to facilitate the extraction and manipulation of the parameters of the GBN.

2.4 Estimating the Parameters: Correlation Coefficients

In this section we will tackle the estimation of the parameters of a GBN assuming its structure (*i.e.*, the DAG) is completely known. For this purpose, we generated a sample of 200 observations from the GBN and we saved it in a data frame called `cropdata1`.

```
> dim(cropdata1)
[1] 200   6
> round(head(cropdata1), 2)
      C     E     G     N     V     W
1 48.83 51.48 42.64 54.10 42.96 41.96
2 48.85 73.43 40.97 60.07 65.29 48.96
3 67.01 71.10 52.52 51.64 63.22 62.03
4 37.83 49.33 56.15 49.01 47.75 38.77
5 55.30 49.27 63.55 54.62 60.57 56.66
6 56.12 48.72 66.02 43.95 55.54 52.39
```

As we did in Section 1.3 for the discrete case, we can use the `bn.fit` function to produce parameter estimates. **bnlearn** will deduce that the BN is a GBN from the fact that the variables in `cropdata1` are not factors, and it will use the appropriate estimators for the parameters.

```
> est.para <- bn.fit(dag.bnlearn, data = cropdata1)
```

At the time of this writing, `bn.fit` implements only the maximum likelihood estimator. Other estimators can be used in a `bn.fit` object by fitting one model for each node with another R package, and either calling `custom.fit` as we did above or replacing the parameter estimates in an existing `bn.fit` object.

The latter is particularly easy for linear models fitted with the `lm` function; we can just assign the return value of `lm` directly to the corresponding node.

```
> est.para$C <- lm(C ~ N + W, data = cropdata1)
```

From the parametric assumptions in Section 2.3, we have that each local distribution can be expressed as a classic Gaussian linear regression model in which the node is the response variable (`C`) and its parents are the explanatory variables (`N`, `W`). The contributions of the parents are purely additive; the model does not contain any interaction term, just the main effect of each parent (`N + W`) and the intercept. The parameter estimates produced by `lm` are the same maximum likelihood estimates we obtained from `bn.fit`. However, it may still be convenient to use `lm` due to its flexibility in dealing with missing values and the possibility of including weights in the model.

Other common forms of regression can be handled in the same way through the **penalized** package (Goeman, 2012). In our example, all the models will have the form $\texttt{C} \sim \mu_{\texttt{C}} + \texttt{N}\beta_{\texttt{N}} + \texttt{W}\beta_{\texttt{W}}$, where $\mu_{\texttt{C}}$ is the intercept and $\beta_{\texttt{N}}, \beta_{\texttt{W}}$ are the regression coefficients for `N` and `W`. Denote each observation for `C`, `N` and `W` with $x_{i,\texttt{C}}$, $x_{i,\texttt{N}}$ and $x_{i,\texttt{W}}$, respectively, where $i = 1, \ldots, n$ and n is the sample size. **penalized** implements *ridge regression*, which estimates $\beta_{\texttt{N}}, \beta_{\texttt{W}}$ with

$$\{\widehat{\beta}_{\texttt{N}}^{\texttt{RIDGE}}, \widehat{\beta}_{\texttt{W}}^{\texttt{RIDGE}}\} = \operatorname*{argmin}_{\beta_{\texttt{N}},\beta_{\texttt{W}}} \left\{ \sum_{i=1}^{n} \left(x_{i,\texttt{C}} - \mu_{\texttt{C}} - x_{i,\texttt{N}}\beta_{\texttt{N}} - x_{i,\texttt{W}}\beta_{\texttt{W}}\right)^2 + \lambda_2(\beta_{\texttt{N}}^2 + \beta_{\texttt{W}}^2) \right\}; \tag{2.7}$$

the *lasso*, with

$$\{\widehat{\beta}_{\texttt{N}}^{\texttt{LASSO}}, \widehat{\beta}_{\texttt{W}}^{\texttt{LASSO}}\} = \\ = \operatorname*{argmin}_{\beta_{\texttt{N}},\beta_{\texttt{W}}} \left\{ \sum_{i=1}^{n} \left(x_{i,\texttt{C}} - \mu_{\texttt{C}} - x_{i,\texttt{N}}\beta_{\texttt{N}} - x_{i,\texttt{W}}\beta_{\texttt{W}}\right)^2 + \lambda_1(|\beta_{\texttt{N}}| + |\beta_{\texttt{W}}|) \right\}; \tag{2.8}$$

and the *elastic net*, with

$$\{\widehat{\beta}_{\texttt{N}}^{\texttt{ENET}}, \widehat{\beta}_{\texttt{W}}^{\texttt{ENET}}\} = \operatorname*{argmin}_{\beta_{\texttt{N}},\beta_{\texttt{W}}} \left\{ \sum_{i=1}^{n} \left(x_{i,\texttt{C}} - \mu_{\texttt{C}} - x_{i,\texttt{N}}\beta_{\texttt{N}} - x_{i,\texttt{W}}\beta_{\texttt{W}}\right)^2 + \right. \\ \left. + \lambda_1(|\beta_{\texttt{N}}| + |\beta_{\texttt{W}}|) + \lambda_2(\beta_{\texttt{N}}^2 + \beta_{\texttt{W}}^2) \right\}, \tag{2.9}$$

which includes ridge regression and the lasso as special cases when $\lambda_1 = 0$ and $\lambda_2 = 0$, respectively. The parameters λ_1 and λ_2 penalise large values of $\beta_{\texttt{N}}, \beta_{\texttt{W}}$ in different ways and shrink them towards zero, resulting in smoother estimates and better predictive power. For instance, we can fit ridge regression for `C` as follows.

```
> library(penalized)
> est.para$C <- penalized(C ~ N + W, lambda1 = 0, lambda2 = 1.5,
+                      data = cropdata1)
```

The parameter set for each node of the GBN is stored in one element of the object returned by `bn.fit` function. Here is the result for the root node `E`, *i.e.*, a node without any parent: it comprises the expectation and the standard deviation of the node. For reference, the estimated values can be compared with the true values (50, 10) from Table 2.1.

```
> est.para$E
  Parameters of node E (Gaussian distribution)

Conditional density: E
Coefficients:
(Intercept)
       50.8
Standard deviation of the residuals: 10.7
```

When a node has one or more parents, the corresponding regression coefficients are also printed.

```
> est.para$C
  Parameters of node C (Gaussian distribution)

Conditional density: C | N + W
Coefficients:
(Intercept)            N            W
      2.403        0.273        0.686
Standard deviation of the residuals: 6.31
```

The estimated regression coefficients are close to their true values from Table 2.1, which are $\beta_{\mathtt{N}} = 0.3$ and $\beta_{\mathtt{W}} = 0.7$. The residual standard deviation is also close to its true value, $\sigma_{\mathtt{C}} = 6.25$. The estimated intercept, however, is $\widehat{\mu}_{\mathtt{C}} = 2.4026$ and is markedly different from $\mu_{\mathtt{C}} = 0$. We can correct this using `lm` as above and fitting a model with a null intercept.

```
> est.para$C <- lm(C ~ N + W - 1, data = cropdata1)
> est.para$C
  Parameters of node C (Gaussian distribution)

Conditional density: C | N + W
Coefficients:
(Intercept)            N            W
      0.000        0.296        0.711
Standard deviation of the residuals: 6.29
```

Now all the parameters in `est.para$C` are reasonably close to their true values.

As is the case for discrete BNs, parameter estimates are based only on the subset of the original data frame spanning the considered node and its parents, following the factorisation in Equation (2.6).

```
> lmC <- lm(C ~ N + W, data = cropdata1[, c("N", "W", "C")])
> coef(lmC)
(Intercept)           N           W
      2.403       0.273       0.686
```

Clearly, the quality of the estimates depends strongly on the sample size.

```
> confint(lmC)
             2.5 % 97.5 %
(Intercept) -4.381  9.186
N            0.181  0.366
W            0.589  0.782
```

As we can see, in the case of `cropdata1` all the confidence intervals for the parameters of `C` from Table 2.1 include the corresponding true values.

2.5 Learning the DAG Structure: Tests and Scores

Often expert knowledge on the data is not detailed enough to completely specify the structure of the DAG. In such cases, if sufficient data are available, we can hope that a statistical procedure may help us in determining a small set of conditional dependencies to translate into a sparse BN. In this section, rather than revisiting all the considerations we made at the beginning of Section 1.5, we will concentrate on the tests and scores specific to GBNs.

2.5.1 Conditional Independence Tests

As was the case for discrete BNs, the two classes of criteria used to learn the structure of the DAG are conditional independence tests and network scores. Both are based on statistics derived within the framework of multivariate normality. As far as tests are concerned, the most common is the exact test for *partial correlations.* The usual empirical correlation coefficient, *e.g.*,

$$\rho_{\mathtt{C},\mathtt{W}} = \mathrm{COR}(\mathtt{C},\mathtt{W}) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_{i,\mathtt{C}} - \bar{x}_{\mathtt{C}})(x_{i,\mathtt{W}} - \bar{x}_{\mathtt{W}})}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{i,\mathtt{C}} - \bar{x}_{\mathtt{C}})^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{i,\mathtt{W}} - \bar{x}_{\mathtt{W}})^2}} \tag{2.10}$$

can only express marginal linear dependencies between two variables, in this case `C` and `W`. However, in GBNs we are usually interested in conditional dependencies. Consider the hypothesis that `C` may be independent from `W` given `N`,

$$H_0 : \mathtt{C} \perp\!\!\!\perp_P \mathtt{W} \mid \mathtt{N} \qquad \text{versus} \qquad H_1 : \mathtt{C} \not\perp\!\!\!\perp_P \mathtt{W} \mid \mathtt{N}, \tag{2.11}$$

which is equivalent to setting $\beta_W = 0$ in the regression model we considered in the previous section. The correlation we need to test is the partial correlation between `C` and `W` given `N`, say $\rho_{\mathtt{C},\mathtt{W}|\mathtt{N}}$, and $\mathtt{C} \perp\!\!\!\perp_P \mathtt{W} \mid \mathtt{N}$ if and only if $\rho_{\mathtt{C},\mathtt{W}|\mathtt{N}}$ is not significantly different from zero; it can be shown that $\beta_W = 0$ if and only if $\rho_{\mathtt{C},\mathtt{W}|\mathtt{N}} = 0$, and that holds in general for all regression coefficients in a Gaussian linear model. Unfortunately, there is no closed form expression for partial correlations, but they can be estimated numerically. First, we need to compute the correlation matrix for `C`, `W` and `N`.

```
> cormat <- cor(cropdata1[, c("C", "W", "N")])
```

Then we compute the inverse `invcor` of `cormat` with the `cor2pcor` function from package **corpcor** (Schäfer et al., 2013), which works even if the input matrix is not full rank.

```
> library(corpcor)
> invcor <- cor2pcor(cormat)
> dimnames(invcor) <- dimnames(cormat)
> invcor
       C      W      N
C 1.000  0.707  0.383
W 0.707  1.000 -0.288
N 0.383 -0.288  1.000
```

We can find the partial correlation $\rho_{\texttt{C,W|N}}$ in `invcor["C", "W"]`; similarly, $\rho_{\texttt{C,N|W}}$ is in `invcor["C", "N"]` and $\rho_{\texttt{W,N|C}}$ is in `invcor["W", "N"]`. More in general, the (X, Y) element of a partial correlation matrix contains the partial correlation between X and Y given all the other variables.

The same estimate for $\rho_{\texttt{C,W|N}}$ is produced by `ci.test` in **bnlearn**, and used to test the hypothesis in Equation (2.11).

```
> ci.test("C", "W", "N", test = "cor", data = cropdata1)
        Pearson's Correlation

data:  C ~ W | N
cor = 0.707, df = 197, p-value < 2.2e-16
alternative hypothesis: true value is not equal to 0
```

The distribution for the test under the null hypothesis is a Student's t distribution with $n - 3 = 197$ degrees of freedom for the transformation

$$\mathrm{t}(\rho_{\texttt{C,W|N}}) = \rho_{\texttt{C,W|N}}\sqrt{\frac{n-3}{1-\rho^2_{\texttt{C,W|N}}}}. \tag{2.12}$$

The degrees of freedom are computed by subtracting the number of variables involved in the test (three in this case) from the sample size. If the tested variables are conditionally independent, t is close to zero; large values, either positive or negative, are indicative of the presence and of the direction of the conditional dependence. So, in the case of $\rho_{\texttt{C,W|N}}$ above, we can say that `C` has a significant positive correlation with `W` given `N` and reject the null hypothesis of independence with an extremely small p-value.

The `cropdata1` data set is not very large, and therefore it is not likely to contain enough information to learn the true structure of the DAG. However, if we perform a naive attempt with one of the algorithms presented in Section 4.5.1.1,

```
> stru1 <- iamb(cropdata1, test = "cor")
```

and compare `stru1` (shown in Figure 2.2) with Figure 2.1, the result is encouraging: only the arc `V` → `N` is missing. All other arcs are present in `stru1` and their directions are correctly identified.

In order to reduce the number of candidate DAGs and help the structure learning algorithm, **bnlearn** gives the possibility to impose some arcs placed in a *whitelist* and to forbid other ones placed in a *blacklist*. These lists are two-column matrices, similar to those returned by the function `arcs`.

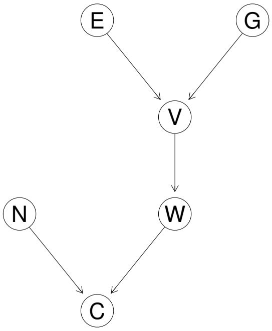


Figure 2.2
The DAG learned from the `cropdata1` data set.

So if we try the following, we obtain the DAG in Figure 2.1.

```
> wl <- matrix(c("V", "N"), ncol = 2)
> wl
     [,1] [,2]
[1,] "V"  "N"
> stru2 <- iamb(cropdata1, test = "cor", whitelist = wl)
> all.equal(dag.bnlearn, stru2)
[1] TRUE
```

Another way to learn a better DAG is to use a bigger sample. Suppose that a sample containing 2×10^4 observations is available in a data frame called `cropdata2`.

```
> dim(cropdata2)
[1] 20000     6
> stru3 <- iamb(cropdata2, test = "cor")
> all.equal(dag.bnlearn, stru3)
[1] TRUE
```

Unsurprisingly, we can see from the output above that the DAG has been correctly learned; all arcs are present and in the correct direction. Note that in general some arcs may be undirected regardless of the sample size, because both their directions are equivalent (see Chapter 4 for more details about equivalent DAGs). As a result, a non-causal approach is unable to conclude which one is relevant.

2.5.2 Network Scores

Network scores for `GBNs` have much in common with the scores for discrete BNs we introduced in Section 1.5.2. For instance, BIC takes the form

$$\begin{aligned}
\mathrm{BIC} &= \log \widehat{\mathtt{f}}(\mathtt{E}, \mathtt{G}, \mathtt{V}, \mathtt{N}, \mathtt{W}, \mathtt{C}) - \frac{d}{2}\log n = \\
&= \left[\log \widehat{\mathtt{f}}(\mathtt{E}) - \frac{d_{\mathtt{E}}}{2}\log n\right] + \left[\log \widehat{\mathtt{f}}(\mathtt{G}) - \frac{d_{\mathtt{G}}}{2}\log n\right] + \\
&+ \left[\log \widehat{\mathtt{f}}(\mathtt{V} \mid \mathtt{E}, \mathtt{G}) - \frac{d_{\mathtt{V}}}{2}\log n\right] + \left[\log \widehat{\mathtt{f}}(\mathtt{N} \mid \mathtt{V}) - \frac{d_{\mathtt{N}}}{2}\log n\right] + \\
&+ \left[\log \widehat{\mathtt{f}}(\mathtt{W} \mid \mathtt{V}) - \frac{d_{\mathtt{W}}}{2}\log n\right] + \left[\log \widehat{\mathtt{f}}(\mathtt{C} \mid \mathtt{N}, \mathtt{W}) - \frac{d_{\mathtt{C}}}{2}\log n\right]
\end{aligned} \tag{2.13}$$

where each local distribution is normally distributed with the parameters we estimated in Section 2.4. So, for instance,

$$\widehat{\mathtt{f}}(\mathtt{C} \mid \mathtt{N}, \mathtt{W}) = N(\widehat{\mu}_{\mathtt{C}} + \mathtt{N}\widehat{\beta}_{\mathtt{N}} + \mathtt{W}\widehat{\beta}_{\mathtt{W}}, \widehat{\sigma}^2_{\mathtt{C}}) \tag{2.14}$$

where $\widehat{\mu}_{\mathtt{C}} = 0$, $\widehat{\beta}_{\mathtt{N}} = 0.3221$, $\widehat{\beta}_{\mathtt{W}} = 0.6737$ and the residual variance $\widehat{\sigma}^2_{\mathtt{C}} = 5.8983$. Likewise, the posterior probability score in common use is that arising from a uniform prior over the space of DAGs and of the parameters; it is called the *Bayesian Gaussian equivalent* score (BGe, see Section 4.5 for details). Both scores can be computed by calling the `score` function from **bnlearn**; the first is obtained with `type = "bic-g"`, the second with `type = "bge"`.

```
> score(dag.bnlearn, data = cropdata2, type = "bic-g")
[1] -416391.45
> score(dag.bnlearn, data = cropdata2, type = "bge")
[1] -416426.13
```

BGe, similarly to BDe, accepts the imaginary sample size `iss` as an additional parameter controlling the relative weight of the uniform prior relative to the observed sample.

2.6 Using Gaussian Bayesian Networks

As explained in Section 1.6, BNs can be investigated from two different points of view: either limiting ourselves to the DAG or using the associated local distributions. DAG properties are independent from the distributional assumptions of the BN. Therefore, nothing has to be added to Section 1.6.1 since the two examples in Figures 1.1 and 2.1 are based on DAGs with the same structure. In this section, we will work only on the local distributions, assuming that the GBN is perfectly known and defined by the `gbn.bnlearn` object created in Section 2.3. We are interested in the probability of an event or in the distribution of some random variables, usually conditional on the values of other variables. Again, such probabilities can be computed either exactly or approximately, as was the case for discrete BNs.

2.6.1 Exact Inference

Some exact inference procedures can be performed with the use of the package **rbmn**, which we used to compute the global distribution of `gbn.bnlearn` in Section 2.3. It relies on three mathematically equivalent classes: `nbn`, `gema` and `mn`. In an `nbn` object, the GBN is described by the local distributions, which is natural considering how BNs are defined.

```
> print8nbn(gbn.rbmn)
=====Nodes===[parents]   = Exp. (sd.dev)
-----------------------------------------------
---------E---[-]  = 50  (10)
---------G---[-]  = 50  (10)
---------V---[E,G]  = -10.355 + 0.707*E + 0.5*G  (5)
---------W---[V]  = 15 + 0.7*V  (7.141)
---------N---[V]  = 45 + 0.1*V  (9.95)
---------C---[N,W]  = 0.3*N + 0.7*W  (6.25)
```

The interpretation of the output is straightforward; the structure of the object can easily be discovered by typing `str(gbn.rbmn)`.

In a `gema` object, the GBN is described by two generating matrices: a vector of expectations and a matrix to be multiplied by a $N(0, 1)$ white noise.

```
> print8gema(gema.rbmn)
  mu     E1   E2  E3   E4   E5   E6
E 50 10.000  0.0 0.0 0.00 0.00 0.00
G 50  0.000 10.0 0.0 0.00 0.00 0.00
V 50  7.071  5.0 5.0 0.00 0.00 0.00
W 50  4.950  3.5 3.5 7.14 0.00 0.00
N 50  0.707  0.5 0.5 0.00 9.95 0.00
C 50  3.677  2.6 2.6 5.00 2.98 6.25
```

So for example, if we consider the row for `V` in the output above, we can read that

$$\texttt{V} = 50 + 7.071\texttt{E1} + 5\texttt{E2} + 5\texttt{E3} \tag{2.15}$$

where $\texttt{E1}, \texttt{E2}, \ldots, \texttt{E6}$ are independent and identically distributed $N(0, 1)$ Gaussian variables. We already used the `mn` form in Section 2.3. Note that for `gema` and `mn` objects the order of the nodes is assumed to be topological, *e.g.*, parent nodes are listed before their children. Note also that no `print` functions are associated to these classes and that the `print8nbn`, `print8gema` and `print8mn` must be used instead.

The function `condi4joint` in **rbmn** can be used to obtain the conditional joint distribution of one or more nodes when the values of the others are fixed. For instance, we can compute the distribution of `C` when `V` is fixed to 80 and that of `V` when `C` is fixed to 80.

```
> print8mn(condi4joint(mn.rbmn, par = "C", pour = "V", x2 = 80))
    mu s.d.
C 65.6 8.54
> print8mn(condi4joint(mn.rbmn, par = "V", pour = "C", x2 = 80))
    mu s.d.
V 65.6 8.54
```

The results are symmetric because of the normalised distributions we chose. In addition, we can use `condi4joint` to obtain the conditional distribution of `C` given an arbitrary value of `V` simply not fixing `V`.

```
> unlist(condi4joint(mn.rbmn, par = "C", pour = "V", x2 = NULL))
 mu.C   rho gamma
24.00  0.52 72.96
```

This means that

$$\texttt{C} \mid \texttt{V} \sim N(24 + 0.52\texttt{V}, 72.9625). \tag{2.16}$$

2.6.2 Approximate Inference

Due to the increasing power of computers, investigating the properties of a given system by simulating it under different conditions and observing its behaviour is becoming increasingly common. When working with probability distributions, simulation means the production of a sample of realisations of random variables. The size of the sample and the approach used to generate it must be in accordance with the magnitude of the probability of the events we want to consider. Clearly, it is much easier to investigate the mean or the median than extreme quantiles in the tails of the distribution; we will see an example of this problem in Section 4.6.2. As shown in the previous section, much can be said about a GBN without any simulation. However, for difficult queries simulation is sometimes the only possible approach.

Depending on the query, the simulation can be either direct or constrained. Function `rbn` implements the former, while `cpquery` and `cpdist` provide an easy access to both options. All three functions are in **bnlearn**.

Simulating from a BN, that is, getting a sample of random values from the joint distribution of the nodes, can always be done by sampling from one node at a time using its local distribution, following the order implied by the arcs of the DAG so that we sample from parent nodes before their children. For instance, we can simulate from the nodes following the order of Table 2.1, using the values simulated in previous steps for conditioning. In such a simulation, the values generated for each node are a sample from its marginal distribution. This is true regardless of the natures and distributions of the nodes. In addition, the same global simulation can be used for a pair of nodes, such as (V, N), as well as for any other subset of nodes. For instance, we can generate nbs $= 4$ observations from (V, N) using our crop GBN as follows,

```
> nbs <- 4
> VG <- rnorm(nbs, mean = 50, sd = 10)
> VE <- rnorm(nbs, mean = 50, sd = 10)
> VV <- rnorm(nbs, mean = -10.355 + 0.5 * VG + 0.707 * VE,
+          sd = 5)
> VN <- rnorm(nbs, mean = 45 + 0.1 * VV, sd = 9.95)
> cbind(VV, VN)
       VV   VN
[1,] 44.2 41.7
[2,] 50.5 50.7
[3,] 49.0 59.4
[4,] 29.4 46.8
```

Of course in the case of GBNs a quicker and easier way is to use **bnlearn**:

```
> sim <- rbn(gbn.bnlearn, n = 4)
> sim[, c("V", "N")]
     V    N
1 44.2 41.7
2 50.5 50.7
3 49.0 59.4
4 29.4 46.8
```

In fact the two data sets we used to introduce the statistical procedures in Section 2.4 and Section 2.5 were obtained in this way, more precisely with the following R code:

```
> set.seed(4567)
> cropdata1 <- rbn(gbn.bnlearn, n = 200)
> set.seed(1234)
> cropdata2 <- rbn(gbn.bnlearn, n = 20000)
```

For the moment, we have not introduced any restriction on the simulation, even though this is common in practice. For instance, we may discuss with an agronomist about our choices in modelling the vegetative mass V. Therefore, we may be interested in simulating V in such extreme scenarios as $\texttt{V} \mid \texttt{G} = 10, \texttt{E} = 90$ to see what happens for particularly bad genotypes grown in very favourable environments. In this case, by construction, we know the conditional distribution we need to answer this question: it is given in the third line of Table 2.1. It is also possible to get the distribution of a pair of conditioned variables, *e.g.*, $\mathtt{N}, \mathtt{W} \mid \mathtt{V} = 35$. This can be achieved by transforming (or building) the corresponding GBN. In addition, when the conditioning is not given directly but we can write it in closed form as in Equation (2.16), it is possible to simulate directly from the conditional distribution.

But this is not always the case. For instance, suppose that we are interested in the conditional distribution $\mathtt{N}, \mathtt{W} \mid \mathtt{C} > 80$ associated with the following question: *what are the values of* N *and* W *associated with a very good crop?* To answer such a question we need to condition not on a single value, but on an interval.

A naive but correct approach is: *just make a simulation with a high number of draws, and retain only those satisfying the condition*, here $\mathtt{C} > 80$. The most important limitation of this approach is that it is not feasible when the probability of generating observations satisfying the condition is very small. Anyway, we can try that way with `cpquery` and `cpdist`.

```
> head(cpdist(gbn.bnlearn, nodes = c("C", "N", "W"),
+          evidence = (C > 80)))
     C    N    W
1 84.2 56.6 80.7
2 80.1 53.5 80.2
3 83.5 64.0 67.2
4 81.2 52.5 73.0
5 85.1 63.9 76.1
6 85.6 70.0 76.6
```

Such an approach is clearly not possible when we are conditioning on a single value for one or more variables, as in the case of $\mathtt{V} \mid \mathtt{G} = 10, \mathtt{E} = 90$. In continuous distributions, a single value always has probability 0, and only intervals may have a non-zero probability. As a result, we would discard all the samples we generated! More advanced simulation approaches are needed to handle this case. A simple one is likelihood weighting from Section 1.6.2.2 (detailed in Section 4.6.2), which can be accessed from `cpdist` by setting `method = "lw"`.

```
> head(cpdist(gbn.bnlearn, nodes = c("V"),
+          evidence = list(G = 10, E = 90), method = "lw"))
     V
1 54.1
2 65.6
3 55.5
4 63.1
5 57.5
6 56.1
```

As we can see from the code above, the `evidence` for this method is provided by a list of values, one for each conditioning variable. Similarly, the probability of a specific `event` can be computed using likelihood weighting via `cpquery`. So, for example, we may be interested in the probability of having a vegetative mass above 70 in the conditions specified by `G` and `E`.

```
> cpquery(gbn.bnlearn, event = (V > 70),
+   evidence = list(G = 10, E = 90), method = "lw")
[1] 0.00935
```

The probability we obtain is very low despite the favourable environments, as expected from such bad genotypes.

2.7 Plotting Gaussian Bayesian Networks

2.7.1 Plotting DAGs

Plots displaying DAGs can be easily produced using several different R packages such as **bnlearn**, **Rgraphviz**, **igraph** and others. In this chapter, we will concentrate on **igraph** since **bnlearn** and **Rgraphviz** have already been presented in Chapter 1. Nevertheless we think that the most convenient way for **bnlearn** users is the function `graphviz.plot` which provides an interface with the R package **Rgraphviz** as shown in Section 1.7.1.

First, we introduce how to define a DAG via its arc set with **igraph**.

```
> library(igraph)
> igraph.options(print.full = TRUE)
> dag0.igraph <- graph.formula(G-+V, E-+V, V-+N, V-+W,
+                                 N-+C, W-+C)
> dag0.igraph
IGRAPH DN-- 6 6 --
+ attr: name (v/c)
+ edges (vertex names):
[1] G->V V->N V->W E->V N->C W->C
```

The arguments provided to the `graph.formula` function identify the nodes at the tail and at the head of each arc in the graph. For instance, `E-+V` indicates there is an arc going from node `E` to node `V`. The "`-`" sign means that `E` is at the tail of the arc, while the "`+`" means that `V` is at the head. Therefore, `E-+V` and `V+-E` identify the same arc.

Starting from the objects we generated with **bnlearn**, it is convenient to convert a `bn` or `bn.fit` object into an **igraph** `graph` object.

```
> dag.igraph <- igraph.from.graphNEL(as.graphNEL(dag.bnlearn))
```

Even though **igraph** implements a large number of functions for handling graphs (not only DAGs), only a very small part will be shown here. We refer the interested reader to the extensive documentation included in the package. Among these functions, V and E display the *vertices* and *edges* of a graph, which are synonymous with nodes and arcs when working with DAGs.

```
> V(dag.igraph)
Vertex sequence:
[1] "C" "E" "G" "N" "V" "W"
> E(dag.igraph)
Edge sequence:

[1] E -> V
[2] G -> V
[3] N -> C
[4] V -> N
[5] V -> W
[6] W -> C
```

We can now plot our DAG in different ways just applying some of the layout algorithms available in **igraph**. Here is the R code which produces Figure 2.3.

```
> par(mfrow = c(2, 2), mar = rep(3, 4), cex.main = 2)
> plot(dag.igraph, main = "\n1: defaults")
> dag2 <- dag.igraph
> V(dag2)$label <- V(dag2)$name
> plot(dag2, main = "\n2: with labels")
> ly <- matrix(c(2, 3, 1, 1, 2, 3,
+                1, 4, 4, 2, 3, 2), 6)
> plot(dag2, layout = ly, main = "\n3: positioning")
> colo <- c("black", "darkgrey", "darkgrey", rep(NA, 3))
> lcolo <- c(rep("white", 3), rep(NA, 3))
> par(mar = rep(0, 4), lwd = 1.5)
> plot(dag2, layout = ly, frame = TRUE,
+      main = "\n4: final",
+      vertex.color = colo, vertex.label.color = lcolo,
+      vertex.label.cex = 3, vertex.size = 50,
+      edge.arrow.size = 0.8, edge.color = "black")
```

Indeed, a difficult task in drawing DAGs is to position the nodes to highlight the structure, the logic and the purpose of the BN. Packages such as **graph** and **Rgraphviz** provide several automatic graph drawing algorithms, but for small graphs the best solution is for the user to place each node himself. With **igraph**, a convenient way to plot graphs is to call tkplot first, hand-tune

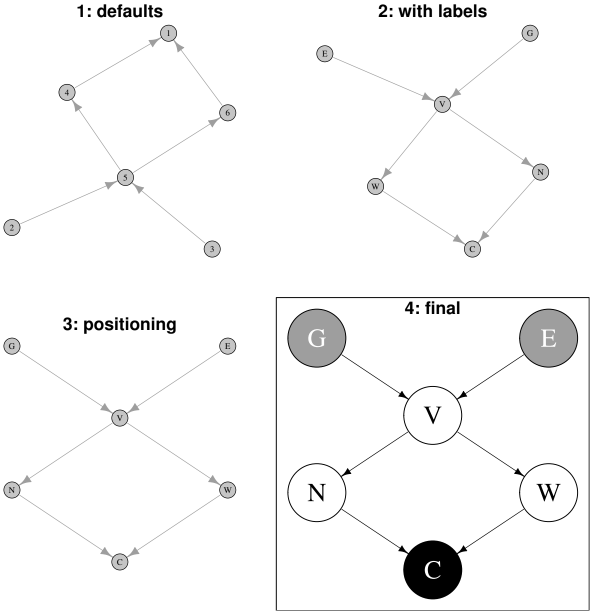


Figure 2.3
Four plots of the crop DAG obtained by specifying more and more arguments with the **igraph** package.

the placement of the nodes, query the coordinates with the `tkplot.getcoords` function and use them to `plot` the graph.

We illustrate how to customise a plot step by step in Figure 2.3. In the first panel, plotting is performed with the default arguments: the position is interesting but arcs are directed upward and the labels identifying the nodes are numbers, not their names. In the second panel, we introduced the correct node labels. In the third panel, the position of the nodes has been fixed with a two-columns matrix. Finally, we added some formatting in the fourth panel. Examining the code used above shows how arguments can be specified globally, as the colours of the arcs, or individually, as the colour of the nodes (`NA` means *no colour*).

2.7.2 Plotting Conditional Probability Distributions

bnlearn does not provide any function to plot local probability distributions in GBNs, unlike the `bn.fit.barchart` function available for discrete BNs. The reason is that while the parents of a node in a discrete BN have a finite number of configurations, which are trivial to enumerate and plot, the parents of a node in a GBN are defined over $\mathbb{R}$ and the corresponding local distribution is very difficult to plot effectively. However, some of common diagnostic plots for linear regression models are available:

- `bn.fit.qqplot`: a quantile-quantile plot of the residuals;
- `bn.fit.histogram`: a histogram of the residuals, with their theoretical normal density superimposed;
- `bn.fit.xyplot`: a plot of the residuals against the fitted values.

All these functions are based on the **lattice** functions of the same name and can be applied to a `bn.fit` object, thus producing a plot with one panel for each node,

```
> gbn.fit <- bn.fit(dag.bnlearn, cropdata2)
> bn.fit.qqplot(gbn.fit)
```

or to a single node in a `bn.fit` object.

```
> bn.fit.qqplot(gbn.fit$V)
```

It is important to note that such plots require the residuals and the fitted values to be stored in the `bn.fit` object. Therefore, GBNs created with `custom.fit` produce an error unless both quantities have been provided by the user.

```
> try(bn.fit.qqplot(gbn.bnlearn))
Loading required package: lattice
Error in lattice.gaussian.backend(fitted = fitted,:
  no residuals present in the bn.fit object.
```

As far as other conditional distributions are concerned, it is not difficult to produce some interesting plots. Here we will give an example with **rbmn**. Suppose that we are interested in how `C` changes in response to variations in `E` and `V`, that is, in $\texttt{C} \mid \texttt{E}, \texttt{V}$. In fact, due to the good properties of GBNs, we can derive the closed form of the associated distribution using:

```
> C.EV <- condi4joint(mn.rbmn, par = "C", pour = c("E", "V"),
+              x2 = NULL)
> C.EV$rho
  E    V
C 0 0.52
```

The zero regression coefficient obtained for `E` when `V` is introduced underscores how no additional information is added by `E` once `V` is already known. This is because `V` d-separates `E` and `C`:

```
> dsep(gbn.bnlearn, "E", "C", "V")
[1] TRUE
```

But imagine that we are in a more complicated case (see Chapter 3 for some examples) in which such an approach is not possible. Our aim is to produce a plot providing insight on the distribution of `C` when both `E` and `V` vary. Since creating a three-dimensional plot is difficult and error-prone, we will replace the third dimension with the size of the points representing each simulated observation. This can be done with the R commands below.

```
> set.seed(5678)
> cropdata3 <- cpdist(gbn.bnlearn, nodes = c("E", "V", "C"),
+                     evidence = TRUE, n = 1000)
> plot(cropdata3$V, cropdata3$C, type = "n",
+      main = "C | V, E; E is the point size")
> cexlim <- c(0.1, 2.4)
> cexE <- cexlim[1] + diff(cexlim) / diff(range(cropdata3$E)) *
+                     (cropdata3$E - min(cropdata3$E))
> points(cropdata3$V, cropdata3$C, cex = cexE)
> cqa <- quantile(cropdata3$C, seq(0, 1, 0.1))
> abline(h = cqa, lty = 3)
```

The result is shown in Figure 2.4. We can see a strong relationship between `V` and `C`, the variables in the x and y axes. No additional effect from `E` to `V` is apparent: for any given level of `C`, the variation of both variables is about the same. Changing their roles (Figure 2.5) highlights the additional effect of `V` with respect to `E`.

2.8 More Properties

Much more could be said about the interesting properties of GBNs. Those that arise from graphical modelling theory can be found in Chapter 4. General properties of multivariate normal distributions, and thus of the global and local distributions in a GBN, can be found in classic multivariate statistics books such as Mardia et al. (1979) and Anderson (2003).

In this last section, we would like to recall some of the most remarkable.

1. The precision matrix is the inverse of the covariance matrix of the global distribution. When the DAG is moralised (see Section 4.4), the absence of an arc between two nodes implies a zero entry in the precision matrix and vice versa. This is because the (i, j) entry of the precision matrix is the partial correlation between the ith and the jth variables given the rest, *e.g.*,

$$\rho_{X_i,X_j|\mathbf{X}\setminus\{X_i,X_j\}} = \texttt{COR}(X_i, X_j \mid \mathbf{X} \setminus \{X_i, X_j\}),$$

and $\rho_{X_i,X_j|\mathbf{X}\setminus\{X_i,X_j\}}$ is equal to zero if and only if the regression coefficient of X_j against X_i is zero as well.

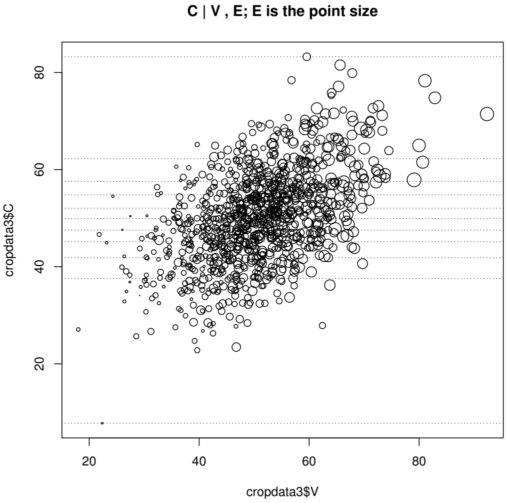


Figure 2.4
Simulated distribution of `C` given `E` and `V`. Horizontal lines correspond to the deciles of `C` (*e.g.*, the 0%, 10%, 20%, . . ., 100% quantiles), so there is the same number of points in each horizontal slice.

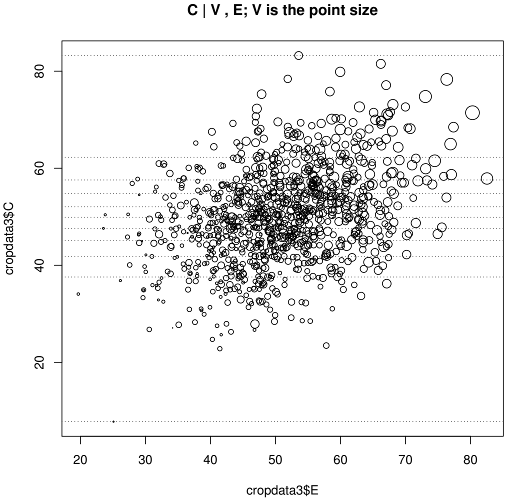


Figure 2.5
Simulated distribution of `C` given `E` and `V`. Horizontal lines correspond to the deciles of `C` (*e.g.*, the 0%, 10%, 20%, . . ., 100% quantiles), so there is the same number of points in each horizontal slice.

2. If some nodes are just linear combinations of their parents, their conditional standard deviation is zero, and the global distribution is degenerate because its covariance matrix is not full rank. In that case, such nodes must be removed before computing the precision matrix; their children can be made to depend on the removed nodes' parents to preserve the original dependencies in a transitive way. In fact, these deterministic nodes do not have any influence on the GBN's behaviour and they can be safely disregarded.
3. Furthermore, the global distribution can be singular or numerically ill-behaved if the GBN is learned from a sample whose size is smaller than the number of parameters ($n \ll p$) or not large enough ($n \approx p$). Since all the optimality properties of the covariance matrix are asymptotic, they hold only approximately even for very large ($n \gg p$) samples. The need for $n \gg p$ can be obviated by the use of *penalised*, *shrinkage* or *Bayesian* estimation techniques that supplement the lack of information in the data and enforce sparsity and regularity in the GBN. More on this topic will be covered in Chapter 4.

2.9 Further Reading

For a broader overview of GBNs, basic definitions and properties are in Koller and Friedman (2009, Chapter 7). Constraint-based structure learning is explored in Korb and Nicholson (2004, Chapter 8), parameter learning in Neapolitan (2003, Section 7.2) and inference in Neapolitan (2003, Section 4.1).

Exercises

Exercise 2.1 *Prove that Equation (2.2) implies Equation (2.3).*

Exercise 2.2 *Within the context of the DAG shown in Figure 2.1, prove that Equation (2.5) is true using Equation (2.6).*

Exercise 2.3 *Compute the marginal variance of the two nodes with two parents from the local distributions proposed in Table 2.1. Why is it much more complicated for* `C` *than for* `V`*?*

Exercise 2.4 *Write an R script using only the* `rnorm` *and* `cbind` *functions to create a* 100×6 *matrix of* 100 *observations simulated from the BN defined in Table 2.1. Compare the result with those produced by a call to* `cpdist` *function.*

Exercise 2.5 *Imagine two ways other than changing the size of the points (as in Section 2.7.2) to introduce a third variable in the plot.*

Exercise 2.6 *Can GBNs be extended to log-normal distributions? If so how, if not, why?*

Exercise 2.7 *How can we generalise GBNs as defined in Section 2.3 in order to make each node's variance depend on the node's parents?*

Exercise 2.8 *From the first three lines of Table 2.1, prove that the joint distribution of* `E`*,* `G` *and* `V` *is trivariate normal.*

3

More Complex Cases: Hybrid Bayesian Networks

In Chapters 1 and 2 we considered BNs with either discrete or continuous variables. Moreover, in each BN all variables followed probability distributions belonging to the same family: multinomial or multivariate normal. In the following, we would like to show that there are no theoretical reasons for such restrictions and that, according to the phenomenon under investigation:

1. we can mix discrete and continuous variables and
2. we can use any kind of distribution.

Unfortunately, this increase in flexibility means BNs become more complicated, and no dedicated R package exists to handle them. Therefore, both learning and inference require some programming effort from the user. One option is to use rejection sampling to perform inference, but as previously underlined (Sections 1.6.2.2 and 2.6.2) it is very inefficient when the conditional probabilities we are interested in are small. As an alternative, we will introduce the use of Markov chain Monte Carlo (MCMC) techniques through different implementations of BUGS, especially JAGS (Just Another Gibbs Sampler; Plummer, 2003). These software packages were developed to perform Bayesian inference in a more general setting, but we will see that they are very convenient to describe complicated BNs. Furthermore, they provide a way to sample from the empirical marginal and conditional distributions of BN nodes.

To illustrate and examine in depth the use of JAGS in the context of BNs, we will first consider one very small and simple example. Afterwards, we will work with a second, more complex example based on the same DAG we used in the previous two chapters. Further considerations about BUGS software packages will be developed in Section 5.2.

3.1 Introductory Example: Reinforcing Steel Rods

A building contractor is planning the construction of a business centre. In the process of securing the building materials required for reinforced concrete,

$$\mathtt{S} \sim Mu\,(1, p = (0.5, 0.5))$$
$$\mathtt{D} \mid \mathtt{S} = s \sim N\left(\mu_s, 0.05^2\right)$$

Table 3.1
Probability distributions proposed for the supplier (S) and the diameter (D) of the steel rods. S can assume two values, s1 and s2. $\mu_{\mathtt{s1}} = 6.1$mm and $\mu_{\mathtt{s2}} = 6.25$mm.

he is now considering which of two suppliers he should buy reinforcing steel rods from to satisfy the project's specifications. The steel rods from the two suppliers are produced with different industrial processes, and therefore have different physical characteristics. For example, while the nominal diameter of the rods (6mm) is the same for both suppliers, there are differences in the positions and the size of the ribs that bind them mechanically to the concrete.

In order to investigate which supplier would be the best choice, we need to combine a continuous variable (the actual diameter of the rods including the ribs, D) with a discrete one (the supplier, S, with values s1 and s2). The most natural way to build such a model is to start with the supplier and make the distribution of the diameter depend on it. A simple formulation is proposed in Table 3.1. The probabilistic model is just a mixture of two normal distributions having a common variance.

An interesting question is whether we can identify which supplier provided a particular steel rod knowing only its diameter, that is, getting the conditional distribution of S | D. In other words, are the rods so different we can identify them in a blind test? The density of D | S and its relationship with S | D are shown in Figure 3.1: D | S is the combination of the two densities for the two suppliers s1 and s2. This kind of problem can be handled numerically by JAGS even in much more complex settings.

3.1.1 Mixing Discrete and Continuous Variables

First we have to define the model. JAGS follows the BUGS syntax, which is similar to that of R. However, it has a very different logic and this is, at first, quite confusing. When programming in R, the variables and the values they store can be assigned, removed and manipulated at run-time. This is not true in BUGS; the assignment operators `<-` and `~` define the values and the distributions of each node in the BN, which cannot be modified afterwards. In other words, BUGS is a declarative language and not a programming language.

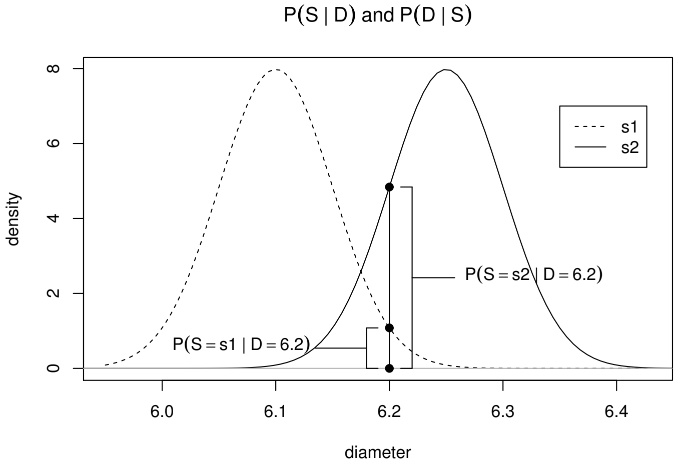


Figure 3.1
Diameter of the reinforcing steel rods according to their supplier, Pr(`D` | `S`), as specified in Table 3.1. The conditional probability Pr(`S` | `D`) can be read from the figure as the ratio of the two densities for any particular value of `D`. For instance, Pr(`S` = `s1` | `D` = 6.2) = Pr(`S` = `s1` | `D` = 6.2)/[Pr(`S` = `s1` | `D` = 6.2) + Pr(`S` = `s2` | `D` = 6.2)] since Pr(`S` = `s1`) = Pr(`S` = `s2`).

Here is the BUGS code for the model proposed in Table 3.1.

```
model {
  csup ~ dcat(sp);
  cdiam ~ dnorm(mu[csup], 1/sigma^2);
}
```

As we can see, the syntax is very concise: one line of code for each node. Two nodes are defined: `csup` and `cdiam`. The first node follows a categorical distribution (`dcat`) assuming values `s1` (coded as 1) and `s2` (coded as 2). The `sp` argument is a vector of length two providing the probabilities of the two categories. The second node follows a normal distribution (`dnorm`); the first parameter specifies the mean (here `mu[csup]`, which depends on the first node), the second one the precision, *i.e.*, the inverse of the variance. The dependence between the two nodes is introduced by the presence of `csup` in the arguments defining the distribution of `cdiam`, exactly as it would be in a mathematical formula. This similarity is one of the main features of BUGS coding.

While there is no automatic translator from BUGS code to R, we can use this model in R by accessing JAGS through the **rjags** package. We can generate random observations from the global distribution and from the conditional distribution of a subset of nodes for fixed values of the others. MCMC is performed transparently to the user with Gibbs sampling, which approximates an arbitrary distribution by sampling from each variable in turn conditional on the rest. Note that each generated sample is used to initialise the following one, so the random observations returned by JAGS are not independent. This is not important when computing conditional means, but the estimates of other summary statistics may be biased unless some care is taken as we will do below.

We can illustrate how this is done in practice by computing the probability that a steel rod with a diameter of 6.2mm has been produced by supplier `s1`. First, we create the R objects for the parameters of the distributions; we call them `sp`, `mu` and `sigma` to match the names of the corresponding arguments in the JAGS code. Subsequently, we store all these objects in a list and we call `jags.model` to load the model we specified above from a text file named `inclu.sc.jam` into JAGS.

```
> library(rjags)
> sp <- c(0.5, 0.5)
> mu <- c(6.1, 6.25)
> sigma <- 0.05
> jags.data <- list(sp = sp, mu = mu, sigma = sigma,
+                   cdiam = 6.20)
> model1 <- jags.model(file = "inclu.sc.jam", data = jags.data)
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 13

Initializing model
```

Constants like `sigma` and quantities that are assumed to be known (such as `cdiam`) are handled by JAGS in the same way, and must be specified in a list whose elements are named the same as in the model file.

After defining the model, we can generate the random samples. Note that JAGS requires a *burn-in* phase for the MCMC to reach its *stationary distribution*. Intuitively, the iterative updating in Gibbs sampling must be initialised using a starting value for each variable. The first random samples will have a distribution that is strongly influenced by the choice of that starting value. This influence becomes weaker with each new sample; therefore, we discard a number of samples at the beginning (*i.e.*, during the burn-in) and assume that the Gibbs sampler has converged to the probability distribution we want to sample from.

```
> update(model1, n.iter = 10000)
> simu1 <- coda.samples(model = model1, variable.names = "csup",
+                       n.iter = 20000, thin = 20)
> sim1 <- simu1[[1]]
```

The function update in **rjags** performs the burn-in, while coda.samples generates random observations from the stationary distribution. coda.samples requires the following arguments: the model (model), the variables whose values will be returned (variable.names), the desired number of simulations (n.iter) and the thinning interval (thin) that specifies the fraction of simulations that will be returned. We set the latter to 1 out of each successive 20 simulations, to minimise the correlation between successive observations. The function coda.samples returns the observations generated in parallel by one or more chains, each stored as an element of a list. In the example above we used a single chain and we saved it in the variable sim1. We can use it to estimate the probability that a steel rod of diameter 6.2mm has been produced by supplier s1 with the frequency of s1 in the chain.

```
> sum(sim1 == 1) / length(sim1)
[1] 0.172
```

The estimated probability is close to 0.1824, its exact theoretical value computed by the ratio of the densities of the two suppliers shown in Figure 3.1.

```
> d.s1 <- dnorm(6.2, mean = mu[1], sd = sigma)
> d.s2 <- dnorm(6.2, mean = mu[2], sd = sigma)
> d.s1 / (d.s1 + d.s2)
[1] 0.1824
```

In conclusion, if a steel rod has a diameter of 6.2mm, it is apparent which supplier produced it: s2, with a probability of 0.8176.

3.1.2 Discretising Continuous Variables

When it is not feasible to include continuous nodes in the BN, a common trick is to discretise them and revert to the multinomial BNs described in Chapter 1. A real-world example of this approach is illustrated in Section 6.1.2.

Consider this possibility for the steel rods example. Figure 3.1 shows that the critical zone to discriminate between the two suppliers is around 6.175. Therefore, we define three categories of diameter with boundaries 6.16 and 6.19: *thin* (less than 6.16), *average* (between 6.16 and 6.19) and *thick* (greater than 6.19). We can then compute the conditional probability table of D given S as well as the probability table of S given D. First, we create a matrix with the probability of each diameter category conditional on the supplier, giving appropriate names to its rows and columns.

```
> limits <- c(6.16, 6.19)
> dsd <- matrix(c(diff(c(0, pnorm(limits, mu[1], sigma), 1)),
+                 diff(c(0, pnorm(limits, mu[2], sigma), 1))),
+               3, 2)
```

```
> dimnames(dsd) <- list(D = c("thin", "average", "thick"),
+                       S = c("s1", "s2"))
> dsd
         S
D               s1      s2
  thin    0.88493 0.03593
  average 0.07914 0.07914
  thick   0.03593 0.88493
```

Subsequently, we compute the joint distribution of D and S by multiplying dsd by the probability of each S, which we stored in sp when we loaded the JAGS model.

```
> jointd <- dsd * sp
```

Finally, we compute the conditional probability of S given D by dividing each row by its total, which we compute using rowSums, and by transposing the resulting matrix with the t function.

```
> dds <- t(jointd / rowSums(jointd))
> dds
    D
S       thin average   thick
  s1 0.96098     0.5 0.03902
  s2 0.03902     0.5 0.96098
```

Using **rjags**, we can assess the probability of each supplier using either the continuous or the discretised distribution, for any diameter. The result is shown in Figure 3.2. The reason for the discrepancies we observe is intuitively obvious: as expected, discretisation may imply losing accuracy when using the BN. And we have used convenient limits to do it: how different would the BNs be if we used, say, $(6.17, 6.18)$ instead of $(6.16, 6.19)$?

It is important to note, however, that discretisation may actually result in better BNs when it is not clear which distributions should be chosen for the nodes or when the decomposition into local distributions is computationally intractable. In such situations, the information that is lost compared to the original data has a much smaller impact than using misspecified distributions or coarse approximations of the conditional probabilities.

3.1.3 Using Different Probability Distributions

A further extension of the BNs proposed in Chapters 1 and 2 is not to limit ourselves to multinomial and multivariate normal distributions. Indeed, the choice of probability distributions for either discrete or continuous variables is quite large; a comprehensive reference is the series of books by Johnson, Kotz and others (Kotz et al., 2000; Johnson et al., 1994, 1995, 1997, 2005).

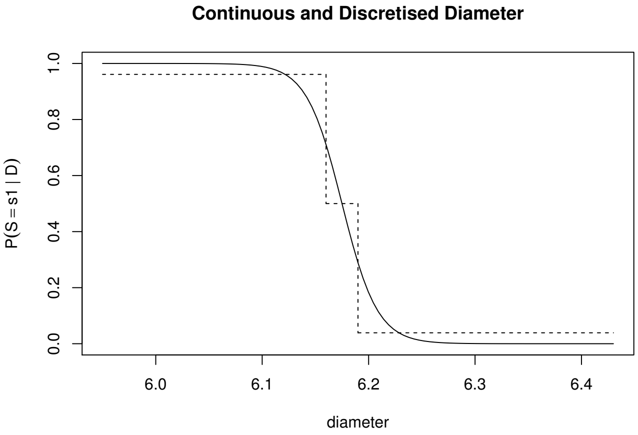


Figure 3.2
Comparison of the continuous (solid line) and discretised (dashed line) approaches to estimate the probability of a steel rod to have been produced by supplier `s1` knowing its diameter.

For instance, BUGS implements 27 different distributions (see Section 5.2.1). Here, we will just sketch how to use them by re-analysing the reinforcing steel rods example with the following BUGS model.

```
model {
  lambda ~ dchisqr(kd);
  xx ~ dpois(lambda);
  sup <- 1 + step(xx - kt);
  diam ~ dgamma(kr[sup], kl[sup]);
}
```

As was the case in the previous section, the model is defined by the marginal distribution of the supplier (`sup`) and by the conditional distribution of the diameter (`diam`) when the supplier is fixed. But instead of a multinomial distribution, the probability that a particular steel rod is produced by `s1` or `s2` is based on a Poisson distribution, whose parameter has a χ^2 distribution. The diameter follows a gamma distribution whose parameters depend on `sup`. Such a model, given the values below for the arguments `kd`, `kt`, `kr` and `kl`, can be processed in the same way as in Section 3.1.1.

```
kd <- 10
kt <- 10
kr <- c(12200, 31000)
kl <- c(2000, 5000)
```

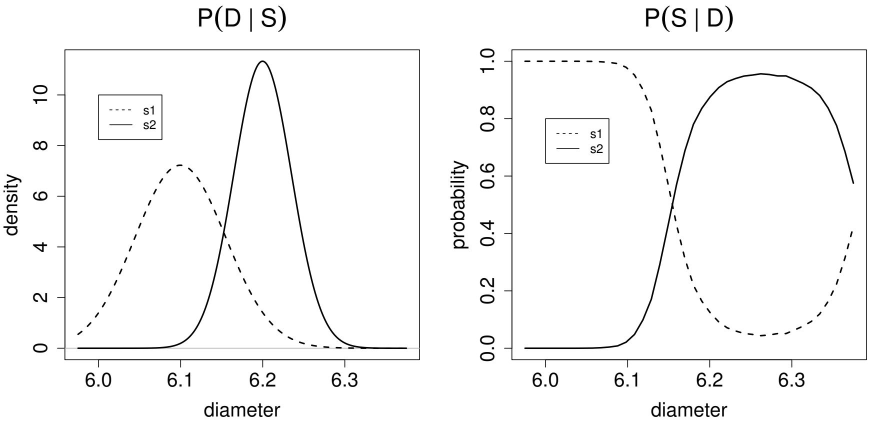


Figure 3.3
Conditional distributions for the steel rod BN based on the Poisson and gamma distributions. The left panel is similar to Figure 3.1. The right panel shows the probability for a steel rod of a given diameter to have been produced by a particular supplier.

Obtaining the marginal and conditional distributions from the model is straightforward, as shown above with `coda.samples`. The conditional probabilities that a steel rod from a given supplier has a certain diameter, Pr(`D` | `S`), and that a steel rod of a given diameter has been produced by supplier `s1`, Pr(`S` = `s1` | `D`), are shown in Figure 3.3; the corresponding plots for the previous model are in Figures 3.1 and 3.2.

It is important to note that Pr(`S` | `D`) does not necessarily take all the values between 0 and 1 any longer, because any particular diameter can be found in the rods from both supplier. For diameter values less than 6.1, we are convinced that the steel rod was produced by supplier `s1`; between 6.1 and 6.2, both are possible; between 6.2 and 6.3, supplier `s2` is the most probable but `s1` cannot be discarded; above 6.3, both are possible again. This is the consequence of the different variances of the suppliers. In fact, the standard deviation depends on the supplier, which is not possible for GBNs as defined in Section 2.3.

3.2 Pest Example with JAGS

Suppose that we are interested in the weight loss caused by some pest in a crop. We would like to explain the behaviour of this target variable with the help of a BN modelling crop plots across a range of conditions for the climate and the soil. Pests can damage the crop in two occasions: when the first generation awakes from the soil at the end of winter, and then when their direct descendants are born at a later time. Denote with `G1` and `G2` the size of the pest population at the first and second generations, respectively. `G1` is influenced mainly by the preceding crop in the plot, `PR`, if only because of the number of larvae of the pest surviving in the soil. The climate conditions during winter, `CL`, are also important because intense frost can kill some of the larvae. The magnitude of the loss, `LO`, is related only to `G2` because the second generation population is larger than the first and eats the reproductive organs of the plants. Finally, another factor has to be considered, `TR`: the farmer can treat his plot against the pest if he believes on the basis of `G1` that the losses due to the pest would be higher than the cost of the treatment. In conclusion, we have the relationships

$$\{\texttt{CL},\texttt{PR}\} \to \texttt{G1}, \qquad \texttt{G1} \to \texttt{G2}, \qquad \texttt{G1} \to \texttt{TR}, \qquad \{\texttt{G2},\texttt{TR}\} \to \texttt{LO}, \tag{3.1}$$

which result in a DAG similar to those in Chapters 1 and 2.

3.2.1 Modelling

The most important difference from the previous BNs is that the probability distributions used in the following are more sophisticated than multinomial and multivariate normal, as can be observed in Figure 3.4 and in Table 3.2.

Nevertheless, this model is very simple to code in BUGS.

```
model {
  PR ~ dcat(p.PR);
  CL ~ dbeta(a.CL, b.CL);
  G1 ~ dpois(CL * g.G1[PR]);
  G2 ~ dpois(G1 * k.G2);
  TR ~ dbern(ilogit((G1 - m.TR)/s.TR));
  x.LO <- G2 * (1 - (1 - r.LO) * TR);
  LO ~ dnchisqr(d.LO, x.LO);
}
```

As in the steel rods example, one node is described in each line, except for `LO` where an intermediate deterministic variable (`x.LO`) has been introduced for the sake of the example (we could have included the formula into the function call as we do for `TR`). To make the model more general, the values of the

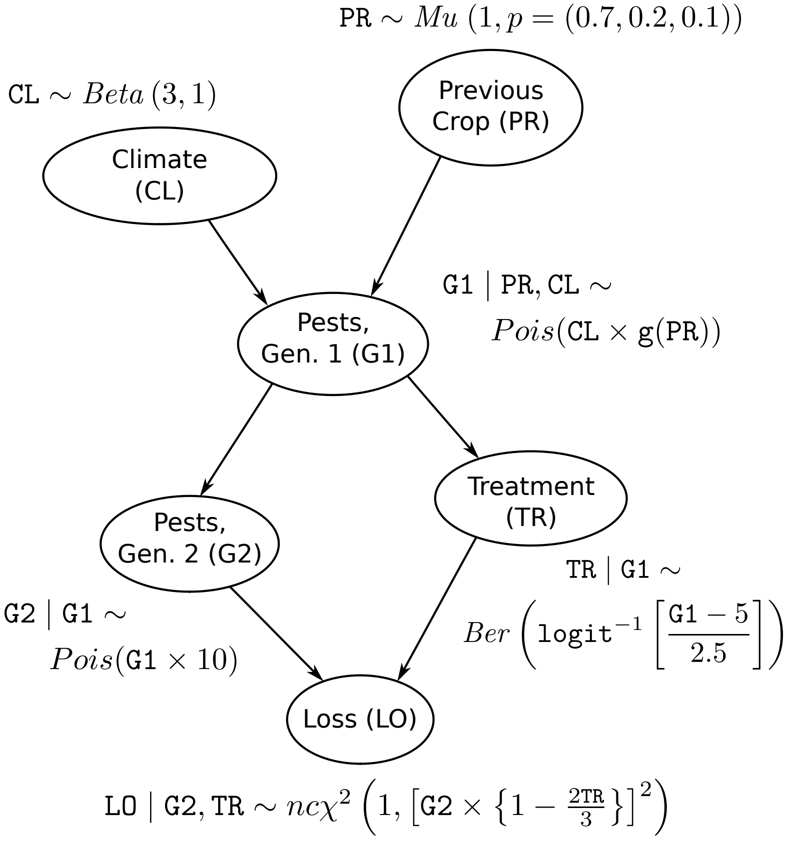

$\texttt{PR} \sim Mu\,(1, p = (0.7, 0.2, 0.1))$
Previous Crop (PR)
$\texttt{CL} \sim Beta\,(3, 1)$
Climate (CL)
$\texttt{G1} \mid \texttt{PR}, \texttt{CL} \sim Pois(\texttt{CL} \times \texttt{g}(\texttt{PR}))$
Pests, Gen. 1 (G1)
Treatment (TR)
Pests, Gen. 2 (G2)
$\texttt{TR} \mid \texttt{G1} \sim Ber\left(\texttt{logit}^{-1}\left[\frac{\texttt{G1} - 5}{2.5}\right]\right)$
$\texttt{G2} \mid \texttt{G1} \sim Pois(\texttt{G1} \times 10)$
Loss (LO)
$\texttt{LO} \mid \texttt{G2}, \texttt{TR} \sim nc\chi^2\left(1, \left[\texttt{G2} \times \left\{1 - \frac{2\texttt{TR}}{3}\right\}\right]^2\right)$

Figure 3.4
The DAG and the local probability distributions for the pest BN. The $\texttt{g}(PR)$ function is defined in the legend of Table 3.2.

$$
\begin{aligned}
\mathtt{PR} &\sim \mathit{Mu}\left(1, p = (0.7, 0.2, 0.1)\right)\\
\mathtt{CL} &\sim \mathit{Beta}\left(3, 1\right)\\
\mathtt{G1} \mid \mathtt{PR} = p, \mathtt{CL} = c &\sim \mathit{Pois}\left(c \times \mathtt{g}\left(p\right)\right)\\
\mathtt{G2} \mid \mathtt{G1} = g_1 &\sim \mathit{Pois}\left(10 g_1\right)\\
\mathtt{TR} \mid \mathtt{G1} = g_1 &\sim \mathit{Ber}\left(\mathtt{logit}^{-1}\left[\frac{g_1 - 5}{2.5}\right]\right)\\
\mathtt{LO} \mid \mathtt{G2} = g_2, \mathtt{TR} = t &\sim nc\chi^2\left(1, \left[g_2 \times \left\{1 - \frac{2t}{3}\right\}\right]^2\right)
\end{aligned}
$$

Table 3.2
Probability distributions proposed for the DAG shown in Figure 3.4. $\mathtt{g}$ is a known function giving the potential of $G1$ for a given class of the last crop: $\mathtt{g(1)} = \mathtt{1}, \mathtt{g(2)} = \mathtt{3}, \mathtt{g(3)} = \mathtt{10}$. *Ber* denotes a Bernoulli distribution, *Pois* denotes a Poisson distribution and $nc\chi^2$ denotes a non-central Chi-square distribution (see Appendix B for their definitions and fundamental properties).

parameters of the distributions have been replaced by constants which are given in the following R list.

```
> dat0 <- list(p.PR = c(0.7, 0.2, 0.1),
+              a.CL = 3, b.CL = 1,
+              g.G1 = c(1, 3, 10),
+              k.G2 = 10,
+              m.TR = 5, s.TR = 2.5,
+              r.LO = 1/3, d.LO = 1)
```

By modifying `dat0` we can perform a *sensitivity analysis* and study the effect of different parameter values on the BN with the help of some loops.

3.2.2 Exploring

Several aspects of this BN are interesting to investigate, either from a theoretical or a practical point of view. In the following, we will focus on a few key questions to illustrate how to interpret the BN and how to use JAGS to that end.

The first question is how to quantify the effect of the last crop on the loss caused by the pest. A simple answer could be the expected loss when `PR` is fixed to 1, 2 or 3. After saving the BUGS model in a file called `inclu.pest.jam`, this could be obtained as follows.

```
> exp.loss  <- rep(NA, 3)
> names(exp.loss) <- paste("PR=", 1:3, sep = "")
> qua.loss <- exp.loss
> for (PR in 1:3) {
+   dat1 <- dat0
+   dat1$PR <- PR
+   mopest <- jags.model(file = "inclu.pest.jam", data = dat1,
+               quiet = TRUE)
+   update(mopest, 3000)
+   sipest <-
+     coda.samples(model = mopest, variable.names = "LO",
+                  n.iter  =  50000)
+   summa <- summary(sipest)
+   exp.loss[PR] <- summa$statistics["Mean"]
+   qua.loss[PR] <- summa$quantiles["75%"]
+ }#FOR
> mean3 <- mean(sipest[[1]][, "LO"])
> round(c(exp.loss, MEAN = mean(exp.loss)), 1)
PR=1 PR=2 PR=3 MEAN
 7.4 18.1 37.6 21.0
```

Note that we do not really need to use JAGS since the evidence is given on the root nodes; such a simple simulation is straightforward to write in R. For instance, we can implement it as follows for PR=1.

```
> set.seed(567)
> nbs <- 50000
> PR <- 1
> g <- function(pr) c(1, 3, 10)[pr]
> CL <- rbeta(nbs, 3, 1)
> G1 <- rpois(nbs, CL * g(PR))
> G2 <- rpois(nbs, G1 * 10)
> il <- function(x) {
+   exp((x - 5) / 2.5)/(1 + exp((x - 5) / 2.5))
+ }#IL
> TR <- rbinom(nbs, 1, il(G1))
> x.lo <- G2 * (1 - (1-1/3)*TR)
> LO <- rchisq(nbs, 1, ncp = x.lo)
> round(mean(LO), 1)
[1] 7.4
```

However, JAGS gives us the ability to condition on every possible combination of the nodes in our BN, greatly increasing our ability to answer complex questions.

Just by looking at the expectation of LO, it is obvious that the last crop has a strong influence on the loss in the current one. On the other hand, we can argue that this is not very informative because very different distributions can

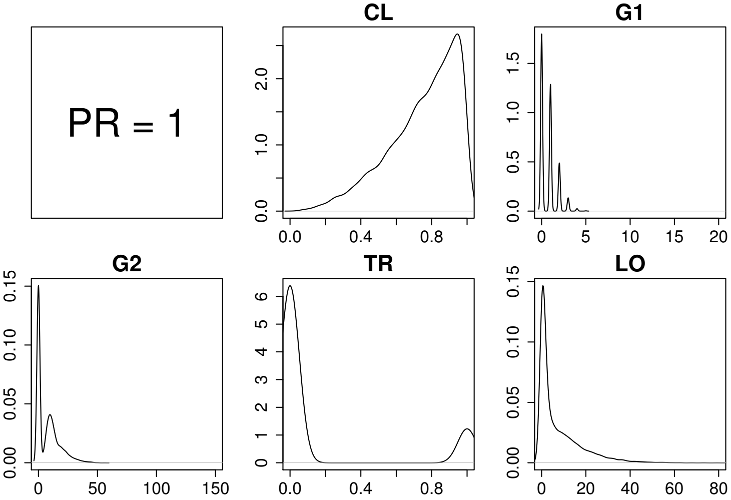


Figure 3.5
Marginal distributions of the nodes when conditioning on PR=1. For the sake of simplicity, discrete nodes are represented as continuous variables, the gaps between integer values being smoothed or not depending on the scale. In particular, note the effect of the smoothing onto the 0/1 variable TR.

have the same expectation. Therefore, it is more effective to use an MCMC simulation to plot the empirical density of the LO node. This can be achieved simply by replacing `mean(sipest[[1]][, "LO"])` with `plot(sipest[[1]][, "LO"])` in the first R code snippet we used in this section. Figures 3.5, 3.6 and 3.7 show the marginal distributions of all nodes (except PR of course) when conditioning on each of the three possible values of PR.

The empirical densities obtained for node LO are consistent with the expectations we obtained above, but reveal that these distributions are far from being symmetric. As a result, in this case expectations are misleading summary statistics because they are be located in the tails of the distributions. Clearly, some quantiles or the corresponding tail probabilities (*i.e.*, the probability of obtaining a value more extreme than that we are considering) would be better choices. Here are the 75% quantiles that we computed previously with `summary`.

```
> round(qua.loss)
PR=1 PR=2 PR=3
  11   26   49
```

They strongly support our belief that the last crop has a marked effect on the loss in the current crop. Examining distributions of the other nodes provides

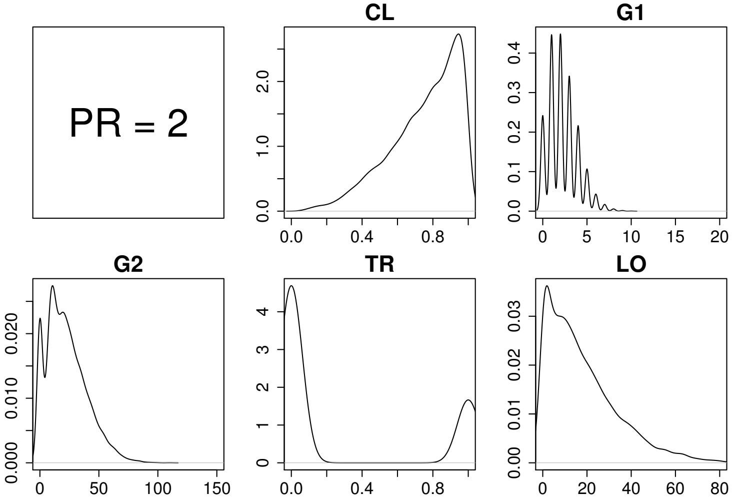


Figure 3.6
Marginal distributions of the nodes when conditioning on PR=2.

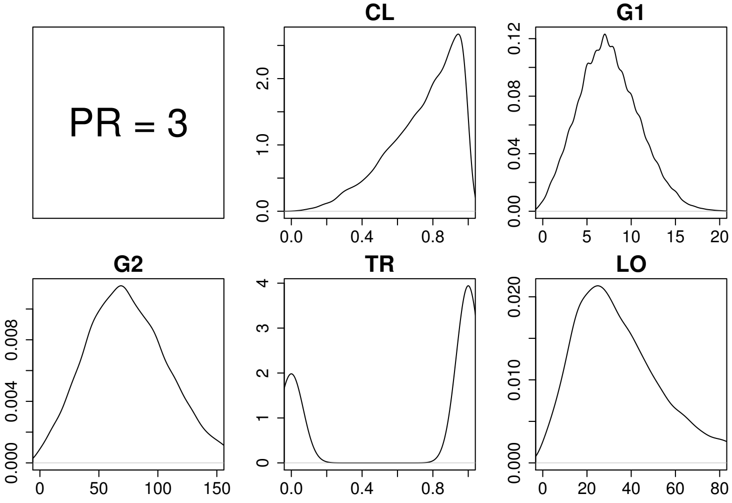


Figure 3.7
Marginal distributions of the nodes when conditioning on PR=3.

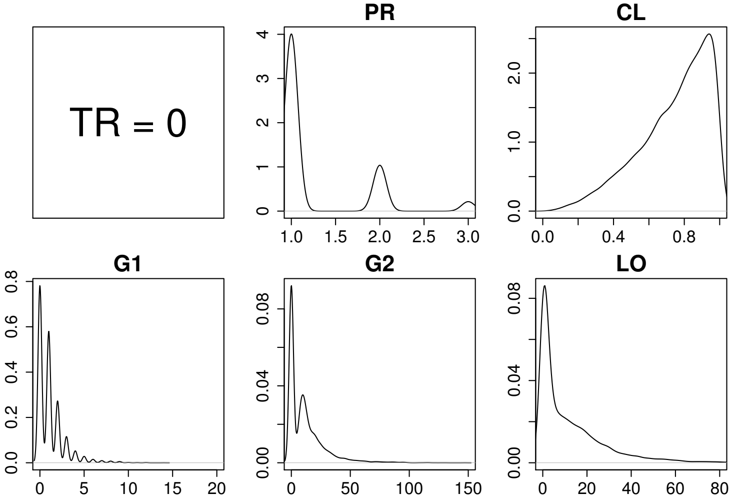


Figure 3.8
Marginal distributions of the nodes when conditioning on TR=0.

additional insight on this phenomenon. In a few words: if the climate remains identical, G1 and G2 increase in proportion to the last crop as well as the amount of pest treatment, but obviously not enough to offset the loss.

Now that we have established the influence of the last crop, we may be interested in studying the effect of treatment by conditioning on its values, TR = 0 and TR = 1. Figures 3.8 and 3.9 show the resulting marginal distributions for the remaining nodes. In this case, we are using the BN instead of an ad-hoc simulation because the required calculations are too complex to perform in a few lines of R code. To do that would require to reimplement the MCMC algorithms used by JAGS; or to derive in closed form the updated distribution of the BN for each treatment and write the corresponding R code. The latter is problematic because it requires the inversion of many dependence relationships in the BN, and the local distributions of the nodes become much more complicated as a result.

Surprisingly, the loss is higher when the treatment is applied than otherwise; the expected loss is 12.3 for TR = 1 and 4.5 for TR = 0! Likewise, the 75% quantiles are 18.3 and 6.5, respectively. The reason for this unexpected behaviour is that we are not conditioning only the loss in the current crop on the treatment; that distribution can be read directly from the last equation of Table 3.2 for any value of the second parent, G2. We are indirectly conditioning all the remaining variables as well, thus modifying their distributions. For instance, this is clearly the case for PR, as more preceding crops of type 3 are present when the treatment is performed. G2 is much larger as a result but,

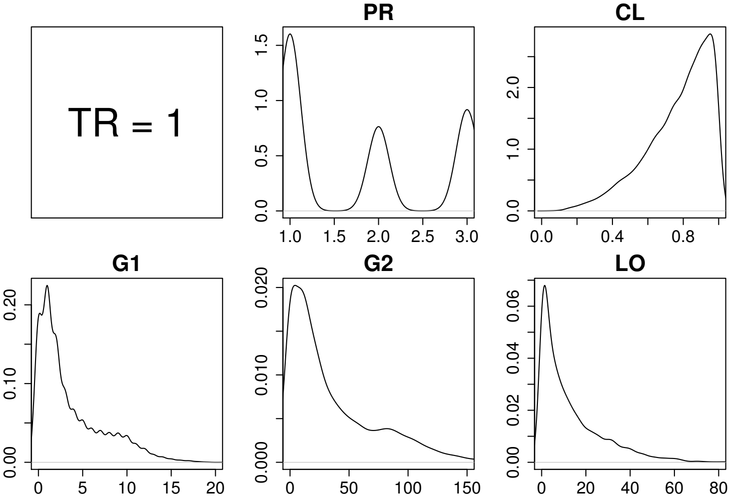


Figure 3.9
Marginal distributions of the nodes when conditioning on `TR=1`.

although the frequency of treated crops is greater, the loss is greater than for the cases where treatment was not applied.

For the moment, we based our inference just on the marginal distributions of the nodes conditional on a single node. In fact, many other configurations are possible, such as fixing several nodes and considering simultaneously the marginal or joint distributions of the rest.

An example of the latter is shown in Figure 3.10. Variables `G2` and `LO` are used to draw a scatter plot of the simulations with different symbols to mark treated and untreated crops. Even though the scatter plot is very cluttered around the origin, making it impossible to distinguish treated and untreated crops, a distinct trend is evident for the other points. Their distribution can be split into two almost non-overlapping clusters defined by the treatment values. This is a logical consequence of the local distribution of `LO | G2, PR`. Surely, when planning next year's harvest it would be very useful to plot the loss for each treatment. In more complex settings, such multivariate diagrams can give interesting insights about the system under consideration.

3.3 About BUGS

Since we used JAGS to manipulate BNs, one could wonder whether we could use JAGS and **rjags** exclusively and thus avoid the limitations of other R

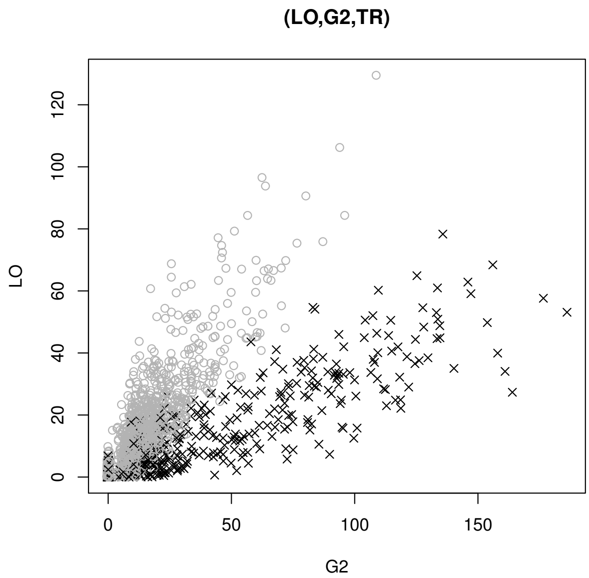


Figure 3.10
Global joint distribution of nodes `LO` (ordinates), `G2` (abscissas) and `TR` (grey "o" for the untreated crops and black "x" for the treated ones).

packages. This is not desirable, for two reasons. First, working with a limited class of BNs is more computationally efficient, both in terms of speed and in the number of nodes that can be handled. Second, JAGS does not implement structure learning.

Furthermore, JAGS is a general Bayesian statistics tool; in that context BNs are just used as a convenient way of defining priors and likelihood functions. It is a credit to the authors of the original BUGS program that the possibilities offered by the software architecture they conceived are far beyond their original expectations.

Note that WinBUGS and OpenBUGS, which have no Linux graphical interface, include a complete set of tools to interpret MCMC simulations. However, it is often preferable to use both programs through their R interfaces, the **R2WinBUGS** (Sturtz et al., 2005) and **BRugs** (Thomas et al., 2006) packages, in combination with the **coda** package (Plummer et al., 2006) to handle the objects returned by the MCMC calculations. This solution provides a more versatile set of tools for inference, and makes it easier to perform complex analyses. A short introduction to the capabilities of BUGS packages is provided in Section 5.2.

3.4 Further Reading

Hybrid BNs are rarely covered in books on graphical models, due to their complexity compared to discrete BNs and GBNs. The particular case of conditional linear Gaussian networks, which combine discrete and continuous variables using a mixture of normal distributions, is explored in Koller and Friedman (2009, Section 5.5.1 and Chapter 14), Koski and Noble (2009, Section 8.9) and Kjærluff and Madsen (2008, Sections 4.1.2 and 5.1.2).

Furthermore, we suggest Lunn et al. (2012) as a most useful book on modelling complex distributions with graphical models and MCMC using BUGS.

Exercises

Exercise 3.1 *Explain why it is logical to get a three-step function for the discretised approach in Figure 3.2.*

Exercise 3.2 *Starting from the BUGS model in Section 3.1.1, write another BUGS model for the discretised model proposed in Section 3.1.2. The functions required for this task are described in the JAGS manual.*

Exercise 3.3 *Let* $d = 6.0, 6.1, 6.2, 6.4$.

1. *Using the BUGS model proposed in Section 3.1.1, write the R script to estimate* $P(\mathtt{S} = \mathtt{s2} \mid \mathtt{D} = \mathtt{d})$ *for the continuous approach demonstrated in the same section.*
2. *Using the BUGS model obtained in Exercise 3.2, write the R script to estimate* $P(\mathtt{S} = \mathtt{s2} \mid \mathtt{D} = \mathtt{d})$ *for the discretised approach suggested in Section 3.1.2.*

And check the results with Figure 3.2.

Exercise 3.4 *In Section 3.1.1, the probability that the supplier is* `s1` *knowing that the diameter is* 6.2 *was estimated to be* 0.1824 *which is not identical to the value obtained with JAGS.*

1. *Explain why the calculation with the R function* `dnorm` *is right and why the value* 0.1824 *is correct. Can you explain why the JAGS result is not exact? Propose a way to improve it.*
2. *Would this value be different if we modify the marginal distribution for the two suppliers?*

Exercise 3.5 *Revisiting the discretisation in Section 3.1.2, compute the conditional probability tables for* `D | S` *and* `S | D` *when the interval boundaries are set to* $(6.10, 6.18)$ *instead of* $(6.16, 6.19)$.

Compared to the results presented in Section 3.1.2, what is your conclusion?

4

Theory and Algorithms for Bayesian Networks

In this chapter we will provide the theoretical foundations underpinning the classes of BNs we explored in Chapter 1 (discrete BNs), Chapter 2 (GBNs) and Chapter 3 (hybrid BNs). In particular, we will introduce the formal definition of a BN and its fundamental properties. We will then show how these properties are at the base of BN learning and inference.

4.1 Conditional Independence and Graphical Separation

BNs are a class of *graphical models*, which allow an intuitive representation of the probabilistic structure of multivariate data using graphs. We have introduced them in Chapter 1 as the combination of:

- a set of random variables $\mathbf{X} = \{X_1, X_2, \ldots, X_p\}$ describing the quantities of interest. The multivariate probability distribution of $\mathbf{X}$ is called the *global distribution* of the data, while the univariate ones associated with each $X_i \in \mathbf{X}$ are called *local distributions*;
- a *directed acyclic graph* (DAG), denoted $G = (\mathbf{V}, A)$. Each node $v \in \mathbf{V}$ is associated with one variable X_i. The directed arcs $a \in A$ that connect them represent direct probabilistic dependencies; so if there is no arc connecting two nodes the corresponding variables are either independent or conditionally independent given a subset of the remaining variables.

The link between the graphical separation (denoted $\perp\!\!\!\perp_G$) induced by the absence of a particular arc and probabilistic independence (denoted $\perp\!\!\!\perp_P$) provides a direct and easily interpretable way to express the relationships between the variables. Following the seminal work of Pearl (1988), we distinguish three possible ways in which the former *maps* to the latter.

Definition 4.1 (Maps) *Let M be the dependence structure of the probability distribution P of $\mathbf{X}$, that is, the set of conditional independence relationships linking any triplet $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ of subsets of $\mathbf{X}$. A graph G is a dependency map (or D-map) of M if there is a one-to-one correspondence between the random variables in $\mathbf{X}$ and the nodes $\mathbf{V}$ of G such that for all disjoint subsets $\mathbf{A}$, $\mathbf{B}$,*

$\mathbf{C}$ *of* $\mathbf{X}$ *we have*

$$\mathbf{A} \perp\!\!\!\perp_P \mathbf{B} \mid \mathbf{C} \Longrightarrow \mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}. \tag{4.1}$$

Similarly, G is an independency map (or I-map) of M if

$$\mathbf{A} \perp\!\!\!\perp_P \mathbf{B} \mid \mathbf{C} \Longleftarrow \mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}. \tag{4.2}$$

G is said to be a perfect map of M if it is both a D-map and an I-map, that is

$$\mathbf{A} \perp\!\!\!\perp_P \mathbf{B} \mid \mathbf{C} \Longleftrightarrow \mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}, \tag{4.3}$$

and in this case G is said to be faithful or isomorphic to M.

In the case of a D-map, the probability distribution of $\mathbf{X}$ determines which arcs are present in the DAG G. Nodes that are connected (*i.e.*, not separated) in G correspond to dependent variables in $\mathbf{X}$; however, nodes that are separated in G do not necessarily correspond to conditionally independent variables in $\mathbf{X}$. On the other hand, in the case of an I-map we have that the arcs present in the DAG G determine which variables are conditionally independent in $\mathbf{X}$. Therefore, nodes that are found to be separated in G correspond to conditionally independent variables in $\mathbf{X}$, but nodes that are connected in G do not necessarily correspond to dependent variables in $\mathbf{X}$. In the case of a perfect map, there is a one-to-one correspondence between graphical separation in G and conditional independence in $\mathbf{X}$.

The graphical separation in Definition 4.1 is established using *d-separation*, which we first introduced in Section 1.6.1. It is formally defined as follows.

Definition 4.2 (d-separation) *If $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are three disjoint subsets of nodes in a DAG G, then $\mathbf{C}$ is said to d-separate $\mathbf{A}$ from $\mathbf{B}$, denoted $\mathbf{A} \perp\!\!\!\perp_G \mathbf{B} \mid \mathbf{C}$, if along every path between a node in $\mathbf{A}$ and a node in $\mathbf{B}$ there is a node v satisfying one of the following two conditions:*

1. *v has converging arcs (i.e., there are two arcs pointing to v from the adjacent nodes in the path) and neither v nor any of its descendants (i.e., the nodes that can be reached from v) are in $\mathbf{C}$.*
2. *v is in $\mathbf{C}$ and does not have converging arcs.*

As an example, consider again the three fundamental connections shown in Figure 1.3, Section 1.6.1. The first was a serial connection from Sex to Education to Residence ($\texttt{S} \rightarrow \texttt{E} \rightarrow \texttt{R}$), and we were investigating $\texttt{S} \perp\!\!\!\perp_G \texttt{R} \mid \texttt{E}$. The node `E`, which plays the role of $v \in \mathbf{C}$ in Definition 4.2, matches the second condition and d-separates `S` and `R`. As a result, we can conclude that $\texttt{S} \perp\!\!\!\perp_G \texttt{R} \mid \texttt{E}$ holds and, in turn, we can determine that `S` and `R` are conditionally independent ($\texttt{S} \perp\!\!\!\perp_P \texttt{R} \mid \texttt{E}$) using Definition 4.1. An identical reasoning leads to the conclusion that $\texttt{O} \perp\!\!\!\perp_G \texttt{R} \mid \texttt{E}$ and $\texttt{O} \perp\!\!\!\perp_P \texttt{R} \mid \texttt{E}$ holds for the divergent connection formed by Education, Occupation and Residence ($\texttt{O} \leftarrow \texttt{E} \rightarrow \texttt{R}$). On the other hand, in the convergent connection formed by Age, Sex and Education ($\texttt{A} \rightarrow \texttt{E} \leftarrow \texttt{S}$) we have that $\texttt{A} \not\perp\!\!\!\perp_G \texttt{S} \mid \texttt{E}$. Unlike the serial and divergent connections, the node in the middle of the connection does not d-separate the other two since `E` does not match any of the two conditions in Definition 4.2.

4.2 Bayesian Networks

Having defined a criterion to determine whether two nodes are connected or not, and how to map those connections (or the lack thereof) to the probability distribution of $\mathbf{X}$, we are now ready to introduce the formal definition of a BN.

Definition 4.3 (BNs) *Given a probability distribution P on a set of variables $\mathbf{X}$, a DAG $G = (\mathbf{X}, A)$ is called a BN and denoted $\mathcal{B} = (G, \mathbf{X})$ if and only if G is a minimal I-map of P, so that none of its arcs can be removed without destroying its I-mapness.*

This definition highlights two fundamental properties of BNs. First, assuming that the DAG is an I-map leads to the general formulation of the decomposition of the global distribution $\Pr(\mathbf{X})$ introduced in Equation (1.1) on page 7:

$$\Pr(\mathbf{X}) = \prod_{i=1}^{p} \Pr(X_i \mid \Pi_{X_i}) \tag{4.4}$$

where Π_{X_i} is the set of the parents of X_i. If X_i has two or more parents it depends on their joint distribution, because each pair of parents forms a convergent connection centred on X_i and we cannot establish their independence. This decomposition is preferable to that obtained from the *chain rule*,

$$\Pr(\mathbf{X}) = \prod_{i=1}^{p} \Pr(X_i \mid X_{i+1}, \ldots, X_p) \tag{4.5}$$

because the conditioning sets are typically smaller. Only when the ordering of the variables is topological, the *chain rule* simplifies to the decomposition in Equation (1.1). In the general case the *chain rule* is more difficult to write, even for the simple multinomial and Gaussian cases presented in Chapters 1 and 2.

Another result along the same lines is called the *local Markov property*, which can be combined with the chain rule above to get the decomposition in Equation (4.4).

Definition 4.4 (Local Markov property) *Each node X_i is conditionally independent of its non-descendants (e.g., nodes X_j for which there is no path from X_i to X_j) given its parents.*

Compared to the previous decomposition, it highlights the fact that parents are not completely independent from their children in the BN; a trivial application of Bayes' theorem to invert the direction of the conditioning shows how information on a child can change the distribution of the parent.

Second, assuming the DAG is an I-map also means that serial and divergent connections result in equivalent factorisations of the variables involved. It is easy to show that

$$\underbrace{\Pr(X_i)\Pr(X_j \mid X_i)\Pr(X_k \mid X_j)}_{\text{serial connection}} = \Pr(X_j, X_i)\Pr(X_k \mid X_j) =$$
$$= \underbrace{\Pr(X_i \mid X_j)\Pr(X_j)\Pr(X_k \mid X_j)}_{\text{divergent connection}}. \quad (4.6)$$

Then $X_i \rightarrow X_j \rightarrow X_k$ and $X_i \leftarrow X_j \rightarrow X_k$ are equivalent. As a result, we can have BNs with different arc sets that encode the same conditional independence relationships and represent the same global distribution in different (but probabilistically equivalent) ways. Such DAGs are said to belong to the same *equivalence class*.

Theorem 4.1 (Equivalence classes) *Two DAGs defined over the same set of variables are equivalent if and only if they have the same skeleton (i.e., the same underlying undirected graph) and the same v-structures.*

In other words, the only arcs whose directions are important are those that are part of one or more *v-structures*.

Definition 4.5 (V-structures) *A convergent connection $X_i \rightarrow X_k \leftarrow X_j$ is called a v-structure if there is no arc connecting X_i and X_j. In addition, X_k is often called the collider node and the connection is then called an unshielded collider, as opposed to a shielded collider in which either $X_i \rightarrow X_j$ or $X_i \leftarrow X_j$.*

As underlined both in Section 1.6.1 and in Section 4.1, convergent connections that are also v-structures have different characteristics than serial and divergent connections for both graphical separation and probabilistic independence. Therefore, the directions of their arcs cannot be changed without altering the global distribution.

Consider, for example, the graphs in Figure 4.1. Using **bnlearn**, we can create the DAG in the top left panel,

```
> X <- paste("[X1][X3][X5][X6|X8][X2|X1][X7|X5][X4|X1:X2]",
+           "[X8|X3:X7][X9|X2:X7][X10|X1:X9]", sep = "")
> dag <- model2network(X)
```

and get its skeleton (top right panel) and v-structures as follows.

```
> skel <- skeleton(dag)
> vstructs(dag)
     X    Z     Y
[1,] "X1" "X10" "X9"
[2,] "X3" "X8"  "X7"
[3,] "X2" "X9"  "X7"
```

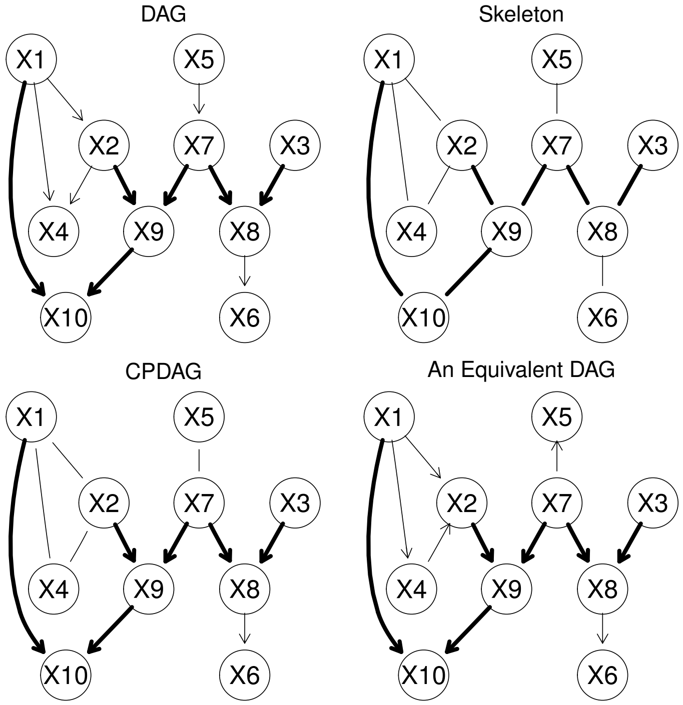


Figure 4.1
A DAG (top left), its underlying undirected graph (the *skeleton*, top right), its CPDAG (bottom left) and another DAG in the same equivalence class (bottom right). Arcs that belong to a v-structure are drawn with a thicker line width.

Those two quantities identify the equivalence class `dag` belongs to, which is represented by the *completed partially directed graph* (CPDAG) shown in the bottom left panel. We can obtain it from `dag` with `cpdag` function.

```
> cp1 <- cpdag(dag)
```

Following Theorem 4.1, not all arcs in this graph are directed; therefore, it is not a DAG because it is only partially directed.

Arcs that were part of one or more v-structures in `dag`, highlighted with a thicker line width, are still directed. Of the remaining arcs, some are directed and some are not. The arc `X8` → `X6` is still directed because its other possible direction (`X6` → `X8`) would introduce additional v-structures in the CPDAG, *e.g.*, `X7` → `X8` ← `X6` and `X3` → `X8` ← `X6`. Since these v-structures are not present in `dag`, any CPDAG including them would not be a valid representation of `dag`'s equivalence class. Therefore, the *partially directed acyclic graph*

(PDAG) obtained from dag's skeleton and its v-structures must be *completed* into a CPDAG by *compelling* the direction of X8 → X6, which is then called a *compelled arc*. Similarly, if disregarding the direction of an arc would result in one of its possible direction introducing cycles, such an arc is considered compelled and its original direction preserved.

On the other hand, the direction of the arc X5 → X7 is not preserved in the CPDAG. Neither X5 → X7 nor X7 → X5 introduce any v-structure in the graph, as they are part only of serial (the former) or divergent connections (the latter). The same holds for the arcs connecting X1, X2 and X4. We can easily verify with set.arc and cpdag that changing the direction of any of those arcs in a way that does not introduce cycles or v-structures results in another DAG (bottom right panel) belonging to the same equivalence class.

```
> dag2 <- dag
> dag2 <- set.arc(dag2, "X7", "X5")
> dag2 <- set.arc(dag2, "X4", "X2")
> dag2 <- set.arc(dag2, "X1", "X2")
> dag2 <- set.arc(dag2, "X1", "X4")
> cp2 <- cpdag(dag2)
> all.equal(cp1, cp2)
[1] "Different arc sets"
```

It is important to note that even though X1 → X4 ← X2 is a convergent connection, it is not a v-structure because X1 and X2 are connected by X1 → X2. As a result, we are no longer able to identify which nodes are the parents in the connection. For example, we can show that $\{\mathtt{X1}, \mathtt{X2}, \mathtt{X4}\}$ have exactly the same probability distribution in the two DAGs in Figure 4.1:

$$\begin{aligned}\underbrace{\Pr(\mathtt{X1})\Pr(\mathtt{X2} \mid \mathtt{X1})\Pr(\mathtt{X4} \mid \mathtt{X1},\mathtt{X2})}_{\mathtt{X1}\to\mathtt{X4}\leftarrow\mathtt{X2},\ \mathtt{X1}\to\mathtt{X2}} &= \Pr(\mathtt{X1})\frac{\Pr(\mathtt{X2},\mathtt{X1})}{\Pr(\mathtt{X1})}\frac{\Pr(\mathtt{X4},\mathtt{X1},\mathtt{X2})}{\Pr(\mathtt{X1},\mathtt{X2})} = \\ = \Pr(\mathtt{X1})\Pr(\mathtt{X4},\mathtt{X2} \mid \mathtt{X1}) &= \underbrace{\Pr(\mathtt{X1})\Pr(\mathtt{X2} \mid \mathtt{X4},\mathtt{X1})\Pr(\mathtt{X4} \mid \mathtt{X1})}_{\mathtt{X4}\to\mathtt{X2}\leftarrow\mathtt{X1},\ \mathtt{X1}\to\mathtt{X4}}. \end{aligned} \tag{4.7}$$

Therefore, the fact that the two parents in a convergent connection are not connected by an arc is crucial in the identification of the correct CPDAG.

4.3 Markov Blankets

The decomposition of the global distribution in Equation (4.4) provides a convenient way to split $\mathbf{X}$ into manageable parts, and identifies in the parents of each node the set of conditioning variables of each local distribution. This is indeed very useful in manually defining a BN or for learning it from data.

However, for the purpose of inference it may be preferable to use Bayes' theorem as suggested by Definition 4.4 and consider also the children of a node to increase the amount of information extracted from the BN.

Going back to the first DAG in Figure 4.1, if we want to perform a query on X9 instead of using

$$\texttt{X9} \sim \Pr(\texttt{X9} \mid \texttt{X2}, \texttt{X7}) \tag{4.8}$$

which incorporates only the information provided by X2 and X7, we may prefer to include more nodes to use the information encoded in the BN to a greater extent, and make inference more powerful as a consequence.

In the limit case, we could decide to condition on all the other nodes in the BN,

$$\texttt{X9} \sim \Pr(\texttt{X9} \mid \texttt{X1}, \texttt{X2}, \texttt{X3}, \texttt{X4}, \texttt{X5}, \texttt{X6}, \texttt{X7}, \texttt{X8}, \texttt{X10}), \tag{4.9}$$

to prevent any bias or loss of power. However, this is unfeasible for most real-world problems, and in fact it would not be much different from using the global distribution. Instead, we can use d-separation to reduce the set of conditioning nodes we need to consider. For example, adding X5 to the conditioning variables in the example Equation (4.8) is pointless, because it is d-separated (and therefore conditionally independent) from X9 by $\{\texttt{X2}, \texttt{X7}, \texttt{X10}\}$:

```
> dsep(dag, x = "X9", y = "X5", z = c("X2", "X7", "X10"))
[1] TRUE
```

The minimal subset of nodes we need to condition upon is called the *Markov blanket*, and is defined as follows.

Definition 4.6 (Markov blanket) *The Markov blanket of a node $A \in \mathbf{V}$ is the minimal subset $\mathbf{S}$ of $\mathbf{V}$ such that*

$$A \perp\!\!\!\perp_G \mathbf{V} - \mathbf{S} - A \mid \mathbf{S}. \tag{4.10}$$

Corollary 4.1 (Markov blankets and faithfulness) *Assuming faithfulness, Definition 4.6 implies $\mathbf{S}$ is the minimal subset of $\mathbf{V}$ such that*

$$A \perp\!\!\!\perp_P \mathbf{V} - \mathbf{S} - A \mid \mathbf{S}. \tag{4.11}$$

Corollary 4.1 provides a formal definition of the set we are looking for: it is the subset of nodes that makes the rest redundant when performing inference on a given node. In other words, it casts Markov blanket identification as a general *feature selection* or *variable selection* problem. Assuming faithfulness makes sure that the set is indeed minimal. Assuming the BN is just an I-map, as in the standard definition, does not guarantee that all the nodes in the Markov blanket are really required to get complete probabilistic independence because not all nodes that are shown to be connected in G are in fact probabilistically dependent.

Theorem 4.2 (Composition of the Markov blanket) *The Markov blanket of a node A is the set consisting of the parents of A, the children of A and all the other nodes sharing a child with A.*

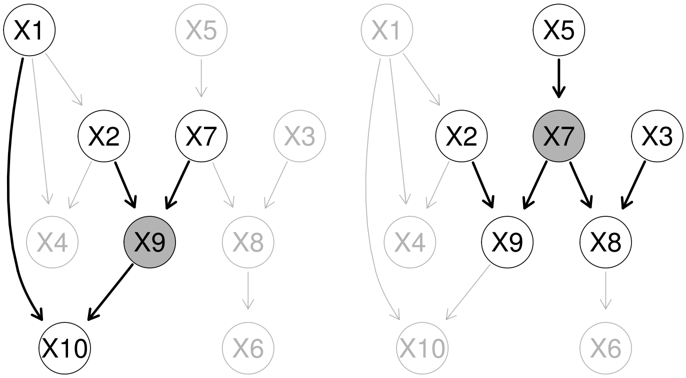


Figure 4.2
The Markov blanket of node X9 (on the left) and that of node X7 (on the right).

Theorem 4.2 identifies which nodes should be included in the Markov blanket to d-separate the target node from the rest of the DAG. Parents and children are required to block the paths that satisfy the first condition in Definition 4.2. Nodes that share a child with the target node, sometimes called *spouses*, are required to block the paths that satisfy the second condition. The target node itself is not part of its Markov blanket.

As an example, the Markov blankets of nodes X9 and X7 are shown in Figure 4.2. We can identify the nodes that belong to each with the mb function from **bnlearn**.

```
> mb(dag, node = "X9")
[1] "X1"  "X10" "X2"  "X7"
> mb(dag, node = "X7")
[1] "X2" "X3" "X5" "X8" "X9"
```

Using the functions parents and children we can also check that the Markov blanket of X9 is composed by the nodes identified by Definition 4.6. First, we identify the parents and the children of X9.

```
> par.X9 <- parents(dag, node = "X9")
> ch.X9 <- children(dag, node = "X9")
```

Then we identify all the parents of the children of X9 by calling the parents function for each child through sapply.

```
> sp.X9 <- sapply(ch.X9, parents, x = dag)
```

We must remove X9 itself from sp.X9, because a node by definition is not part of its own Markov blanket. Finally, we merge parX9, ch.X9 and sp.X9 to form the Markov blanket.

```
> sp.X9 <- sp.X9[sp.X9 != "X9"]
> unique(c(par.X9, ch.X9, sp.X9))
[1] "X2"  "X7"  "X10" "X1"
```

Equivalently, we might have used nbr to identify both parents and children in a single function call, as they are the neighbours of X9. The Markov blanket of X7 can be similarly investigated.

We can also test whether the Markov blanket of X9 d-separates X9 from all the other nodes in dag.

```
> V <- setdiff(nodes(dag), "X9")
> S <- mb(dag, "X9")
> sapply(setdiff(V, S), dsep, bn = dag, y = "X9", z = S)
  X3   X4   X5   X6   X8
TRUE TRUE TRUE TRUE TRUE
```

Similarly, for X7 we have

```
> V <- setdiff(nodes(dag), "X7")
> S <- mb(dag, "X7")
> sapply(setdiff(V, S), dsep, bn = dag, y = "X7", z = S)
  X1  X10   X4   X6
TRUE TRUE TRUE TRUE
```

as expected.

Furthermore, it can be shown as a corollary of Theorem 4.2 that Markov blankets are symmetric.

Corollary 4.2 (Symmetry of Markov blankets) *Theorem 4.2 defines a symmetric relationship: if node A is in the Markov blanket of B, then B is in the Markov blanket of A.*

So, for example, if we take each node in the Markov blanket S of X7 in turn and identify its Markov blanket, we have that X7 belongs to each of those Markov blankets.

```
> belongs <- logical(0)
> for (node in S)
+   belongs[node] <- "X7" %in% mb(dag, node)
> belongs
  X2   X3   X5   X8   X9
TRUE TRUE TRUE TRUE TRUE
```

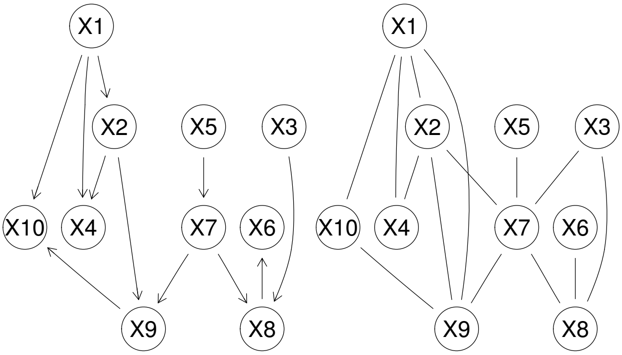


Figure 4.3
The DAG from Figures 4.1 and 4.2 (left) and its moral graph (right). Some nodes have been moved to give a clearer view of the relationships.

4.4 Moral Graphs

In Section 4.2 we introduced an alternative graphical representation of the DAG underlying a BN: the CPDAG of the equivalence class the BN belongs to. Another graphical representation that can be derived from the DAG is the *moral graph*.

The moral graph is an undirected graph that is constructed as follows:

1. connecting the non-adjacent nodes in each v-structure with an undirected arc;
2. ignoring the direction of the other arcs, effectively replacing them with undirected arcs.

This transformation is called *moralisation* because it “marries” non-adjacent parents sharing a common child. In the case of our example `dag`, we can create the moral graph with the `moral` function as follows; the result is shown in Figure 4.3.

```
> mg1 <- moral(dag)
```

It is important to note that different DAGs may result in identical moral graphs because of the nature of the transformations above; for instance, adding an arc from `X7` to `X3` to `dag` does not alter its moral graph.

```
> all.equal(moral(dag),
+           moral(set.arc(dag, from = "X7", to = "X3")))
[1] TRUE
```

It is also easy, for the sake of the example, to perform moralisation manually. First, we identify the v-structures in the DAG and we link the parents within each v-structure with an undirected arc (*e.g.*, an edge).

```
> mg2 <- dag
> vs <- vstructs(dag)
> for (i in seq(nrow(vs)))
+   mg2 <- set.edge(mg2, from = vs[i, "X"], to = vs[i, "Y"],
+              check.cycles = FALSE)
```

This step appears to introduce potential cycles in the resulting PDAG. However, we can safely ignore them since we are going to construct the undirected graph underlying mg2, thus replacing each directed arc with an undirected one.

```
> mg2 <- skeleton(mg2)
> all.equal(mg1, mg2)
[1] TRUE
```

Moralisation has several uses. First, it provides a simple way to transform a BN into the corresponding *Markov network*, a graphical model using undirected graphs instead of DAGs to represent dependencies. These models have a long history in statistics, and have been studied in such classic monographs as Whittaker (1990) and Lauritzen (1996). On the one hand, all dependencies are explicitly represented in a Markov network, even those that would be implicitly implied by v-structures in a BN. On the other hand, the addition of undirected arcs makes the graph structure less informative, because we cannot tell that nodes that were parents in a v-structure are marginally independent when we are not conditioning on their common child. Second, moral graphs provide the foundation for exact inference on BNs through the junction tree algorithm, which will be introduced in Section 4.6.2.

4.5 Bayesian Network Learning

In the field of BNs, model selection and estimation are collectively known as *learning*, a name borrowed from artificial intelligence and machine learning. BN learning is usually performed as a two-step process:

1. *structure learning*, learning the structure of the DAG;
2. *parameter learning*, learning the local distributions implied by the structure of the DAG learned in the previous step.

Both steps can be performed either as *unsupervised learning*, using the information provided by a data set, or as *supervised learning*, by interviewing experts in the fields relevant for the phenomenon being modelled. Combining both approaches is common. Often the prior information available on the phenomenon is not enough for an expert to completely specify a BN. Even specifying the DAG structure is often impossible, especially when a large number of variables are involved. This is the case, for example, for gene network analysis.

This workflow is inherently Bayesian. Consider a data set $\mathcal{D}$ and a BN $\mathcal{B} = (G, \mathbf{X})$. If we denote the parameters of the global distribution of $\mathbf{X}$ with Θ, we can assume without loss of generality that Θ uniquely identifies $\mathbf{X}$ in the parametric family of distributions chosen for modelling $\mathcal{D}$ and write $\mathcal{B} = (G, \Theta)$. BN learning can then be formalised as

$$\underbrace{\Pr(\mathcal{B} \mid \mathcal{D}) = \Pr(G, \Theta \mid \mathcal{D})}_{\text{learning}} \quad = \quad \underbrace{\Pr(G \mid \mathcal{D})}_{\text{structure learning}} \quad \cdot \quad \underbrace{\Pr(\Theta \mid G, \mathcal{D})}_{\text{parameter learning}} \ . \tag{4.12}$$

The decomposition of $\Pr(G, \Theta \mid \mathcal{D})$ reflects the two steps described above, and underlies the logic of the learning process.

Structure learning can be done in practice by finding the DAG G that maximises

$$\Pr(G \mid \mathcal{D}) \propto \Pr(G)\Pr(\mathcal{D} \mid G) = \Pr(G)\int \Pr(\mathcal{D} \mid G, \Theta)\Pr(\Theta \mid G)d\Theta, \tag{4.13}$$

using Bayes' theorem to decompose the posterior probability of the DAG (*i.e.*, $\Pr(G \mid \mathcal{D})$) into the product of the prior distribution over the possible DAGs (*i.e.*, $\Pr(G)$) and the probability of the data (*i.e.*, $\Pr(\mathcal{D} \mid G)$). Clearly, it is not possible to compute the latter without also estimating the parameters Θ of G; therefore, Θ has to be integrated out of Equation (4.13) to make $\Pr(G \mid \mathcal{D})$ independent of any specific choice of Θ.

The prior distribution $\Pr(G)$ provides an ideal way to introduce any prior information available on the conditional independence relationships between the variables in $\mathbf{X}$. We may, for example, require that one or more arcs should be present in or absent from the DAG, to account for the insights gained in previous studies. We may also require that some arcs, if present in the DAG, must be oriented in a specific direction when that direction is the only one that makes sense in the light of the logic underlying the phenomenon being modelled. This was the case in the examples presented in Chapter 1 and Chapter 2. In Chapter 1, Age (`A`) and Sex (`S`) cannot be the head of any arcs because they are demographic indicators. It also makes little sense to allow either `A` → `S` or `S` → `A` to be present in the DAG. Similar considerations can be made for the genetic potential (`G`) and the environmental potential (`E`) in Chapter 2.

The most common choice for $\Pr(G)$ is a non-informative prior over the space of the possible DAGs, assigning the same probability to every DAG. Some DAGs may be excluded outright due to prior information, as we discussed above. More complex priors (known as *structural priors*) are also possible, but rarely used in practice for two reasons. First, using a uniform probability distribution makes $\Pr(G)$ irrelevant in maximising $\Pr(G \mid \mathcal{D})$. This makes it convenient for both computational and algebraic reasons. Second, the number of possible DAGs increases super-exponentially in the number of nodes. In a DAG with p nodes we have $\frac{1}{2}p(p-1)$ possible arcs, given by the pairs of different nodes in $\mathbf{V}$. Even disregarding the arcs' directions, this means that there are $2^{O(p^2)}$ possible DAGs. Specifying a prior distribution over such a large number of DAGs is a challenging task for even small problems.

Computing $\Pr(\mathcal{D} \mid G)$ is also problematic from both a computational and an algebraic point of view. Starting from the decomposition into local distributions, we can further factorise $\Pr(\mathcal{D} \mid G)$ in a similar way:

$$\begin{aligned}\Pr(\mathcal{D} \mid G) &= \int \prod_{i=1}^{p} \left[\Pr(X_i \mid \Pi_{X_i}, \Theta_{X_i}) \Pr(\Theta_{X_i} \mid \Pi_{X_i})\right] d\Theta = \\ &= \prod_{i=1}^{p} \left[\int \Pr(X_i \mid \Pi_{X_i}, \Theta_{X_i}) \Pr(\Theta_{X_i} \mid \Pi_{X_i}) d\Theta_{X_i}\right] = \\ &= \prod_{i=1}^{p} \mathrm{E}_{\Theta_{X_i}} \left[\Pr(X_i \mid \Pi_{X_i})\right]. \end{aligned} \tag{4.14}$$

Functions that can be factorised in this way are called *decomposable.* If all expectations can be computed in close form, $\Pr(\mathcal{D} \mid G)$ can be computed in a reasonable time even for large data sets. This is possible both for the multinomial distribution assumed for discrete BNs (via its conjugate Dirichlet posterior) and for the multivariate Gaussian distribution assumed for continuous BNs (via its conjugate Inverse Wishart distribution). For discrete BNs, $\Pr(\mathcal{D} \mid G)$ can be estimated with the *Bayesian Dirichlet equivalent uniform* (BDeu) score from Heckerman et al. (1995). Since it is the only member of the BDe family of scores in common use, it is often referred to simply as BDe. As we have seen in Chapter 1, BDe assumes a flat prior over both the space of the DAGs and the parameter space of each node:

$$\Pr(G) \propto 1 \qquad \text{and} \qquad \Pr(\Theta_{X_i} \mid \Pi_{X_i}) = \alpha_{ij} = \frac{\alpha}{|\Theta_{X_i}|}. \tag{4.15}$$

The only parameter of BDe is the *imaginary sample size* α associated with the Dirichlet prior, which determines how much weight is assigned to the prior as the size of an imaginary sample supporting it.

Under these assumptions, BDe takes the form

$$\begin{aligned}\mathrm{BDe}(G, \mathcal{D}) &= \prod_{i=1}^{p} \mathrm{BDe}(X_i, \Pi_{X_i}) = \\ &= \prod_{i=1}^{p} \prod_{j=1}^{q_i} \left\{ \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ij} + n_{ijk}))}{\Gamma(\alpha_{ijk})} \right\}\end{aligned} \tag{4.16}$$

where:

- p is the number of nodes in G;
- r_i is the number of categories for the node X_i;
- q_i is the number of configurations of the categories of the parents of X_i;
- n_{ijk} is the number of samples which have the jth category for node X_i and the kth configuration for its parents.

The corresponding posterior probability for GBNs is called *Bayesian Gaussian equivalent uniform* (BGeu) from Geiger and Heckerman (1994), which is again commonly referred to as BGe. Similarly to BDe, it assumes a non-informative prior over both the space of the DAGs and the parameter space of each node; and its only parameter is the imaginary sample size α. Its expression is very complicated, and will not be reported here.

As a result of the difficulties outlined above, two alternatives to the use of $\Pr(\mathcal{D} \mid G)$ in structure learning have been developed. The first one is the use of the *Bayesian Information criterion* (BIC) as an approximation of $\Pr(\mathcal{D} \mid G)$, as

$$\mathrm{BIC}(G, \mathcal{D}) \to \log \mathrm{BDe}(G, \mathcal{D}) \qquad \text{as the sample size } n \to \infty. \tag{4.17}$$

BIC is decomposable and only depends on the likelihood function,

$$\mathrm{BIC}(G, \mathcal{D}) = \sum_{i=1}^{p} \left[\log \Pr(X_i \mid \Pi_{X_i}) - \frac{|\Theta_{X_i}|}{2} \log n \right] \tag{4.18}$$

which makes it very easy to compute (see Equation (1.14) for discrete BNs and Equation (2.13) for GBNs). The second alternative is to avoid the need to define a measure of goodness-of-fit for the DAG and to use conditional independence tests to learn the DAG structure one arc at a time. All these approaches to structure learning will be covered in Section 4.5.1.

Once we have learned the DAG structure we can move to parameter learning, that is, we can estimate the parameters of $\mathbf{X}$. Assuming that parameters belonging to different local distributions are independent, we actually need to estimate only the parameters of one local distribution at a time. Following the Bayesian approach outlined in Equation (4.12), this would require to find the value of the Θ that maximises $\Pr(\Theta \mid G, \mathcal{D})$ through its components $\Pr(\Theta_{X_i} \mid X_i, \Pi_{X_i})$. Other approaches to parameter estimation, such as maximum likelihood regularised estimation, are also common; their advantages and disadvantages will be covered in Section 4.5.2.

Local distributions in practice involve only a small number of nodes, *i.e.*, X_i and it parents Π_{X_i}. Furthermore, their dimension usually does not scale with the number of nodes in the BN (and is often assumed to be bounded by a constant when computing the computational complexity of algorithms), thus avoiding the so called *curse of dimensionality.* This means that each local distribution has a comparatively small number of parameters to estimate from the sample, and that estimates are more accurate due to the better ratio between the size of Θ_{X_i} and the sample size.

4.5.1 Structure Learning

Several algorithms have been presented in literature for this problem, thanks to the application of results arising from probability, information and optimisation theory. Despite the (sometimes confusing) variety of theoretical backgrounds and terminology they can all be traced to only three approaches: *constraint-based*, *score-based* and *hybrid.*

All these structure learning algorithms operate under a set of common assumptions:

- There must be a one-to-one correspondence between the nodes in the DAG and the random variables in $\mathbf{X}$; this means in particular that there must not be multiple nodes which are deterministic functions of a single variable.

- All the relationships between the variables in $\mathbf{X}$ must be conditional independencies, because they are by definition the only kind of relationships that can be expressed by a BN.

- Every combination of the possible values of the variables in $\mathbf{X}$ must represent a valid, observable (even if really unlikely) event. This assumption implies a strictly positive global distribution, which is needed to have uniquely determined Markov blankets and, therefore, a uniquely identifiable model. Constraint-based algorithms work even when this is not true, because the existence of a perfect map is also a sufficient condition for the uniqueness of the Markov blankets (Pearl, 1988).

- Observations are treated as independent realisations of the set of nodes. If some form of temporal or spatial dependence is present, it must be specifically accounted for in the definition of the network, as in *dynamic Bayesian networks.*

4.5.1.1 Constraint-based Algorithms

Constraint-based algorithms are based on the seminal work of Pearl on maps and its application to causal graphical models. His *Inductive Causation* (IC) algorithm (Verma and Pearl, 1991) provides a framework for learning the DAG structure of BNs using conditional independence tests.

The details of the IC algorithm are described in Algorithm 4.1. The first step identifies which pairs of variables are connected by an arc, regardless of its direction. These variables cannot be independent given any other subset of variables, because they cannot be d-separated. This step can also be seen as a backward selection procedure starting from the saturated model with

Algorithm 4.1 Inductive Causation Algorithm

```
1. For each pair of nodes $A$ and $B$ in $\mathbf{V}$ search for set $\mathbf{S}_{AB} \subset V$ such that $A$ and $B$ are independent given $\mathbf{S}_{AB}$ and $A, B \notin \mathbf{S}_{AB}$. If there is no such a set, place an undirected arc between $A$ and $B$.
2. For each pair of non-adjacent nodes $A$ and $B$ with a common neighbour $C$, check whether $C \in \mathbf{S}_{AB}$. If this is not true, set the direction of the arcs $A - C$ and $C - B$ to $A \rightarrow C$ and $C \leftarrow B$.
3. Set the direction of arcs which are still undirected by applying recursively the following two rules:
    (a) if $A$ is adjacent to $B$ and there is a strictly directed path from $A$ to $B$ then set the direction of $A - B$ to $A \rightarrow B$;
    (b) if $A$ and $B$ are not adjacent but $A \rightarrow C$ and $C - B$, then change the latter to $C \rightarrow B$.
4. Return the resulting CPDAG.
```

a complete graph and pruning it based on statistical tests for conditional independence. The second step deals with the identification of the v-structures among all the pairs of non-adjacent nodes A and B with a common neighbour C. By definition, v-structures are the only fundamental connection in which the two non-adjacent nodes are not independent conditional on the third one. Therefore, if there is a subset of nodes that contains C and d-separates A and B, the three nodes are part of a v-structure centred on C. This condition can be verified by performing a conditional independence test for A and B against every possible subset of their common neighbours that includes C. At the end of the second step, both the skeleton and the v-structures of the network are known, so the equivalence class the BN belongs to is uniquely identified. The third and last step of the IC algorithm identifies compelled arcs and orients them recursively to obtain the CPDAG describing the equivalence class identified by the previous steps.

A major problem of the IC algorithm is that the first two steps cannot be applied in the form described in Algorithm 4.1 to any real-world problem due to the exponential number of possible conditional independence relationships. This has led to the development of improved algorithms such as:

- *PC*: the first practical application of the IC algorithm (Spirtes et al., 2000);
- *Grow-Shrink* (GS): based on the *Grow-Shrink Markov blanket* algorithm (Margaritis, 2003), a simple forward selection Markov blanket detection approach;
- *Incremental Association* (IAMB): based on the *Incremental Association Markov blanket* algorithm (Tsamardinos et al., 2003), a two-phase selection scheme;

- *Fast Incremental Association* (Fast-IAMB): a variant of IAMB which uses speculative stepwise forward selection to reduce the number of conditional independence tests (Yaramakala and Margaritis, 2005);

- *Interleaved Incremental Association* (Inter-IAMB): another variant of IAMB which uses forward stepwise selection (Tsamardinos et al., 2003) to avoid false positives in the Markov blanket detection phase.

All these algorithms, with the exception of PC, first learn the Markov blanket of each node. This preliminary step greatly simplifies the identification of neighbours. This in turn results in a significant reduction in the number of conditional independence tests, and therefore of the overall computational complexity of the learning algorithm. Further improvements are possible by leveraging the symmetry of Markov blankets (Corollary 4.2). As far as the quality of the learned CPDAGs is concerned, on average Inter-IAMB produces fewer false positives than GS, IAMB or Fast-IAMB while having a comparable number of false negatives. The PC algorithm as extended in Kalisch and Bühlmann (2007), Kalisch and Bühlmann (2008) and Bühlmann et al. (2010) is also competitive. In the case of high-dimensional data sets, the best choice is probably the Semi-Interleaved Hiton-PC from Aliferis et al. (2010), which can scale well up to thousands of variables.

Conditional independence tests used to learn discrete BNs are functions of the observed frequencies $\{n_{ijk}, i = 1, \ldots, R, j = 1, \ldots, C, k = 1, \ldots, L\}$ for the random variables X and Y and all the configurations of the conditioning variables $\mathbf{Z}$. We introduced two such tests in Section 1.5.1:

- the *mutual information* test, an information-theoretic distance measure defined as
$$\mathrm{MI}(X, Y \mid \mathbf{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{C}\sum_{k=1}^{L} \frac{n_{ijk}}{n} \log \frac{n_{ijk}n_{++k}}{n_{i+k}n_{+jk}} \tag{4.19}$$
and equivalent to the log-likelihood ratio test G^2 (they differ by a $2n$ factor, where n is the sample size);

- the classic *Pearson's* X^2 test for contingency tables,
$$\mathrm{X}^2(X, Y \mid \mathbf{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{C}\sum_{k=1}^{L} \frac{(n_{ijk} - m_{ijk})^2}{m_{ijk}}, \quad \text{where} \quad m_{ijk} = \frac{n_{i+k}n_{+jk}}{n_{++k}}. \tag{4.20}$$

Another possibility is the shrinkage estimator for the mutual information defined by Hausser and Strimmer (2009) and studied in the context of BNs in Scutari and Brogini (2012).

Test	Asymptotic χ^2	Monte Carlo	Sequential Monte Carlo	Semiparametric χ^2
mutual information (G^2)	`"mi"`	`"mc-mi"`	`"smc-mi"`	`"sp-mi"`
Pearson's X^2	`"x2"`	`"mc-x2"`	`"smc-x2"`	`"sp-x2"`
mutual information (shrinkage)	`"mi-sh"`	–	–	–

Table 4.1
Labels of the conditional independence tests for discrete BNs implemented in **bnlearn**.

For all the tests above, the null hypothesis of independence can be tested using:

- the asymptotic $\chi^2_{(R-1)(C-1)L}$ distribution, which is very fast to compute but requires an adequate sample size to be accurate;

- the Monte Carlo permutation approach or the faster, sequential Monte Carlo permutation approaches described in Edwards (2000). Both approaches ensure that the tests are unbiased, and that they are valid even for small sample sizes. However, they are computationally expensive;

- the semiparametric χ^2 distribution described in Tsamardinos and Borboudakis (2010), which represents a compromise between the previous two approaches.

Several of these combinations of tests statistics and distributions are implemented in **bnlearn**, for use both in structure learning and as standalone tests in `ci.test` (for some examples of the latter, see Section 1.5.1). The labels used to identify them throughout the package are reported in Table 4.1.

In the case of GBNs, conditional independence tests are functions of the partial correlation coefficients $\rho_{XY|\mathbf{Z}}$ of X and Y given $\mathbf{Z}$. Two common conditional independence tests are:

- the exact t test for Pearson's correlation coefficient, defined as

$$\mathrm{t}(X, Y \mid \mathbf{Z}) = \rho_{XY|\mathbf{Z}} \sqrt{\frac{n - |\mathbf{Z}| - 2}{1 - \rho^2_{XY|\mathbf{Z}}}} \tag{4.21}$$

  and distributed as a Student's t with $n - |\mathbf{Z}| - 2$ degrees of freedom;

Test	Exact distribution	Asymptotic distribution	Monte Carlo	Sequential Monte Carlo
t test	`"cor"`	–	`"mc-cor"`	`"smc-xcor"`
Fisher's Z test	–	`"zf"`	`"mc-zf"`	`"smc-zf"`
mutual information	–	`"mi-g"`	`"mc-mi-g"`	`"smc-mi-g"`
mutual information (shrinkage)	–	`"mi-g-sh"`	–	–

Table 4.2
Labels of the conditional independence tests for GBNs implemented in **bnlearn**.

- *Fisher's Z* test, a transformation of $\rho_{XY|\mathbf{Z}}$ with an asymptotic normal distribution and defined as

$$\mathrm{Z}(X, Y \mid \mathbf{Z}) = \log\left(\frac{1+\rho_{XY|\mathbf{Z}}}{1-\rho_{XY|\mathbf{Z}}}\right)\frac{\sqrt{n-|\mathbf{Z}|-3}}{2} \qquad (4.22)$$

  where n is the number of observations and $|\mathbf{Z}|$ is the number of nodes belonging to $\mathbf{Z}$.

Another possible choice is the mutual information test defined in Kullback (1968), which is again proportional to the corresponding log-likelihood ratio test and has an asymptotic χ^2_1 distribution. The shrinkage estimator for the covariance matrix developed by Shäfer and Strimmer (2005) can also be used to construct a regularised mutual information test, again with a χ^2_1 asymptotic distribution. The labels used to identify these tests and their Monte Carlo variants in **bnlearn** are reported in Table 4.2.

Constraint-based structure learning algorithms are implemented in **bnlearn** in the `gs`, `iamb`, `fast.iamb` and `inter.iamb` functions. All these functions accept the following arguments:

- `x`, the data the network will be learned from;
- `whitelist` and `blacklist`, to force the inclusion or the exclusion of specific arcs from the network;
- `test`, the label of the conditional independence test to be used. It defaults to `"mi"` for discrete BNs and `"cor"` for GBNs;
- `alpha`, the type I error threshold for the test. It defaults to $\alpha = 0.05$, as is common in literature. Note that, regardless of the dimension of the BN, there

is no need to apply any multiple testing adjustment to α, because constraint-based learning algorithms are largely self-adjusting in that respect;

- `B`, the number of permutations for Monte Carlo tests;
- `debug`, which prints the sequence of tests performed by the learning algorithm. Setting it to `TRUE` may be useful both for understanding how structure learning works in practice, for debugging, and also to assess the swiftness of the learning algorithm.

Consider one more time the example of structure learning presented in Section 2.5 for the `cropdata1` data set, which contains 200 observations. By default, Grow-Shrink uses the exact t test for partial correlations with $\alpha = 0.05$.

```
> bn.cor <- gs(cropdata1, test = "cor", alpha = 0.05)
> modelstring(bn.cor)
[1] "[E][G][N][V|E:G][W|V][C|N:W]"
```

The small sample size seems to reduce the power of the tests, and the arc `V` → `N` is missing from the DAG as a result. Therefore, we may wonder whether changing the conditional independence test to Fisher's Z test or to a Monte Carlo test may help in learning the correct structure.

```
> bn.zf <- gs(cropdata1, test = "zf", alpha = 0.05)
> bn.mc <- gs(cropdata1, test = "mc-cor", B = 1000)
> all.equal(bn.cor, bn.zf)
[1] TRUE
> all.equal(bn.cor, bn.mc)
[1] TRUE
```

As we can see for the output above, both tests result in the same DAG as the exact t test. Likewise, using IAMB instead of Grow-Shrink does not result in the correct structure.

```
> bn.iamb <- iamb(cropdata1, test = "cor", alpha = 0.05)
> all.equal(bn.cor, bn.iamb)
[1] TRUE
```

Therefore, we may presume that the problems in learning `V` → `N` lie more in the small sample size than in the limitations of a particular algorithm or a particular test. If we set the `debug` argument to `TRUE`, we can identify exactly which tests fail to detect the dependence between `N` and `V`.

```
> gs(cropdata1, test = "cor", alpha = 0.05, debug = TRUE)
[...]
----------------------------------------------------------------
* learning the markov blanket of N .
[...]
  * checking node V for inclusion.
    > N indep. V given ' C ' (p-value: 0.3048644).
  * checking node W for inclusion.
    > node W included in the markov blanket (p-value: 7.89e-07).
    > markov blanket ( 2 nodes ) now is ' C W '.
    > restarting grow loop.
  * checking node V for inclusion.
    > N indep. V given ' C W ' (p-value: 0.1416876).
  * checking node W for exclusion (shrinking phase).
    > node W remains in the markov blanket. (p-value: 7.89e-07)
[...]
```

When learning the Markov blanket of N, we have that node C is included before V. Apparently, the dependence between C and N is much stronger than the dependence between V and N; the latter is not statistically significant when the former is taken into account. As a result, the marginal dependence test between V and N is significant at $\alpha = 0.05$ while the same test conditional on C is not.

```
> ci.test("N", "V", test = "cor", data = cropdata1)
        Pearson's Correlation

data:  N ~ V
cor = -0.0589, df = 198, p-value = 0.4074
alternative hypothesis: true value is not equal to 0
> ci.test("N", "V", "C", test = "cor", data = cropdata1)
        Pearson's Correlation

data:  N ~ V | C
cor = -0.216, df = 197, p-value = 0.002187
alternative hypothesis: true value is not equal to 0
```

Now that we have diagnosed the problem behind the missing V → N arc, we can include it using the whitelist argument and thus obtain the correct DAG (stru2 in Section 2.5).

```
> bn.cor <- gs(cropdata1, test = "cor", alpha = 0.05,
+             whitelist = c("V", "N"))
> all.equal(bn.cor, dag.bnlearn)
[1] TRUE
```

Algorithm 4.2 Hill-Climbing Algorithm

```
1. Choose a network structure $G$ over $\mathbf{V}$, usually (but not necessarily)
   empty.
2. Compute the score of $G$, denoted as $Score_G = \text{Score}(G)$.
3. Set $maxscore = Score_G$.
4. Repeat the following steps as long as $maxscore$ increases:
   (a) for every possible arc addition, deletion or reversal not resulting
       in a cyclic network:
        i. compute the score of the modified network $G^*$,
           $Score_{G^*} = \text{Score}(G^*)$:
       ii. if $Score_{G^*} > Score_G$, set $G = G^*$ and $Score_G = Score_{G^*}$.
   (b) update $maxscore$ with the new value of $Score_G$.
5. Return the DAG $G$.
```

4.5.1.2 Score-based Algorithms

Score-based learning algorithms represent the application of heuristic optimisation techniques to the problem of learning the structure of a BN. Each candidate BN is assigned a *network score* reflecting its goodness of fit, which the algorithm then attempts to maximise. Some examples from this class of algorithms are:

- *greedy search* algorithms such as *hill-climbing* with *random restarts* or *tabu search* (Bouckaert, 1995). These algorithms explore the search space starting from a network structure (usually without any arc) and adding, deleting or reversing one arc at a time until the score can no longer be improved (see Algorithm 4.2);

- *genetic* algorithms, which mimic natural evolution through the iterative selection of the "fittest" models and the hybridisation of their characteristics (Larrañaga et al., 1997). In this case the search space is explored through the *crossover* (which combines the structure of two networks) and *mutation* (which introduces random alterations) stochastic operators;

- *simulated annealing* (Bouckaert, 1995). This algorithm performs a stochastic local search by accepting changes that increase the network score and, at the same time, allowing changes that decrease it with a probability inversely proportional to the score decrease.

A comprehensive review of these heuristics, as well as related approaches from the field of artificial intelligence, are provided in Russell and Norvig (2009).

As far as network scores are concerned, the only two options in common use are those we introduced in Section 4.5: posterior probabilities arising from flat priors such as BDe (for discrete BNs) and BGe (for GBNs), and the BIC score. In **bnlearn**, they are labelled "bde", "bge", "bic" (for discrete BNs) and "bic-g" (for GBNs). This lack of options, compared to the number of conditional independence tests we considered in the previous section, can be attributed to the difficulty of specifying a tractable likelihood or posterior distribution for the DAG.

As an example of score-based structure learning, consider once again the structure learning example from Section 1.5. After considering some manual arc additions and removal, we called the hill-climbing implementation in **bnlearn** and obtained a DAG with a better BIC score.

```
> learned <- hc(survey, score = "bic")
> modelstring(learned)
[1] "[R][E|R][T|R][A|E][O|E][S|E]"
> score(learned, data = survey, type = "bic")
[1] -1998.432
```

The tabu function, which implements tabu search, returns the same DAG. Both hc and tabu have argument sets similar to those of the functions presented in the previous section; the main difference is the presence of a score argument instead of test, alpha and B. In particular, we can investigate the workings of hc by setting the debug argument to TRUE, just in the same way we did for gs in the previous section.

```
> learned <- hc(survey, score = "bic", debug = TRUE)
----------------------------------------------------------------
* starting from the following network:

  Random/Generated Bayesian network

  model:
   [A][R][E][O][S][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

* current score: -2008.943
[...]
----------------------------------------------------------------
* best operation was: adding R -> E .
* current network is :
[...]
----------------------------------------------------------------
* best operation was: adding E -> S .
[...]
----------------------------------------------------------------
* best operation was: adding R -> T .
[...]
----------------------------------------------------------------
* best operation was: adding E -> A .
[...]
----------------------------------------------------------------
* best operation was: adding E -> O .
[...]
```

The search for the network that optimises the BIC score starts, by default, from the empty DAG. The operation that increases the BIC score the most is, at each step, the addition of one of the arcs that will be present in final DAG (see Figure 4.4).

Neither `hc` nor `tabu` are able to learn the true DAG. There are many reasons for such a behaviour. For instance, it is possible for both algorithms to get stuck at a local maximum because of an unfavourable choice for the starting point of the search. This does not appear to be the case for the `survey` data, because even when starting from the true DAG (`survey.dag`) the final DAG is not the true one.

```
> survey.dag <- model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
> learned.start <- hc(survey, score = "bic", start = survey.dag)
> modelstring(learned.start)
[1] "[S][E|S][A|E][O|E][R|E][T|R]"
> all.equal(cpdag(learned), cpdag(learned.start))
[1] TRUE
```

As an alternative, we could also start the search from a randomly generated graph.

```
> hc(survey, score = "bic", start = random.graph(names(survey)))
```

Furthermore, it is interesting to note that the networks returned for the different starting DAGs fall in the same equivalence class. This suggests that the dependencies specified by `survey.dag` are not completely supported by the `survey` data, and that `hc` and `tabu` do not seem to be affected by convergence or numerical problems.

4.5.1.3 Hybrid Algorithms

Hybrid learning algorithms combine constraint-based and score-based algorithms to offset the respective weaknesses and produce reliable network struc-

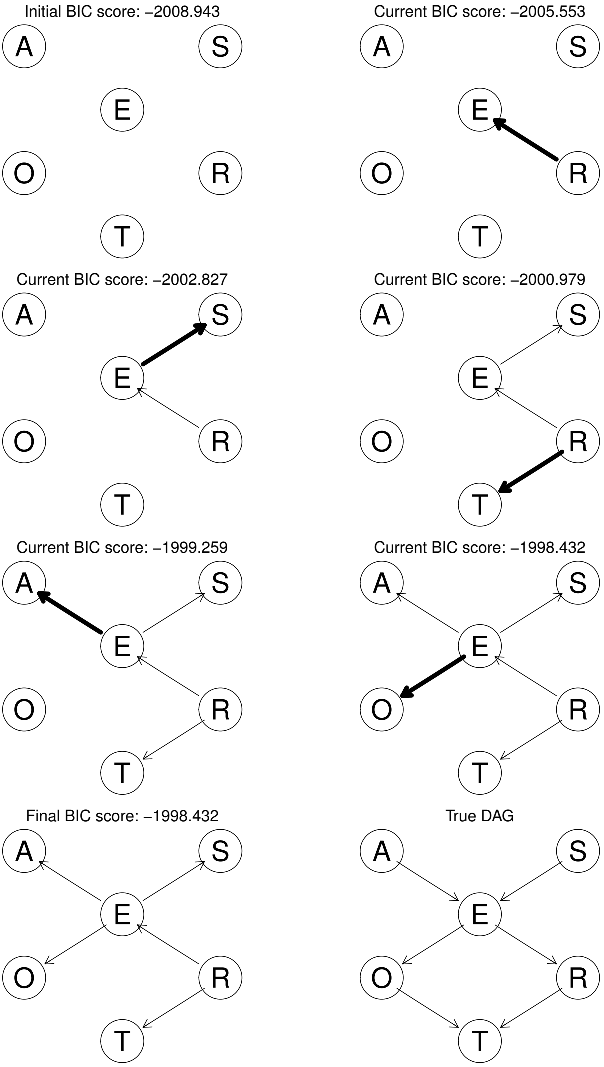

Initial BIC score: −2008.943
A
S
E
O
R
T
Current BIC score: −2005.553
A
S
E
O
R
T
Current BIC score: −2002.827
A
S
E
O
R
T
Current BIC score: −2000.979
A
S
E
O
R
T
Current BIC score: −1999.259
A
S
E
O
R
T
Current BIC score: −1998.432
A
S
E
O
R
T
Final BIC score: −1998.432
A
S
E
O
R
T
True DAG
A
S
E
O
R
T

Figure 4.4
Steps performed by hill-climbing when learning the DAG with optimal BIC score from the `survey` data. The DAG in the bottom right panel is the original DAG from Chapter 1.

Algorithm 4.3 Sparse Candidate Algorithm

```
1. Choose a network structure $G$ over $\mathbf{V}$, usually (but not necessarily)
   empty.
2. Repeat the following steps until convergence:
   (a) restrict: select a set $\mathbf{C}_i$ of candidate parents for each node
       $X_i \in \mathbf{V}$, which must include the parents of $X_i$ in $G$;
   (b) maximise: find the network structure $G^*$ that maximises
       Score($G^*$) among the networks in which the parents of each
       node $X_i$ are included in the corresponding set $\mathbf{C}_i$;
   (c) set $G = G^*$.
3. Return the DAG $G$.
```

tures in a wide variety of situations. The two best-known members of this family are the *Sparse Candidate* algorithm (SC) by Friedman et al. (1999) and the *Max-Min Hill-Climbing* algorithm (MMHC) by Tsamardinos et al. (2006). The former is illustrated in Algorithm 4.3.

Both these algorithms are based on two steps, called *restrict* and *maximise*. In the first one, the candidate set for the parents of each node X_i is reduced from the whole node set $\mathbf{V}$ to a smaller set $C_i \subset \mathbf{V}$ of nodes whose behaviour has been shown to be related in some way to that of X_i. This in turn results in a smaller and more regular search space. The second step seeks the network that maximises a given score function, subject to the constraints imposed by the $\mathbf{C}_i$ sets.

In the Sparse Candidate algorithm these two steps are applied iteratively until there is no change in the network or no network improves the network score; the choice of the heuristics used to perform them is left to the implementation. On the other hand, in the Max-Min Hill-Climbing algorithm *Restrict* and *Maximise* are performed only once; the *Max-Min Parents and Children* (MMPC) heuristic is used to learn the candidate sets $\mathbf{C}_i$ and a hill-climbing greedy search to find the optimal network.

MMHC is implemented in **bnlearn** in the `mmhc` function,

```
> mmhc(survey)
```

which is equivalent to the more general `rsmax2` function when the `restrict` argument is set to `"mmpc"` and the `maximize` argument is set to `"hc"`.

```
> rsmax2(survey, restrict = "mmpc", maximize = "hc")
```

`rsmax2` implements a single-iteration variant of the Sparse Candidate algorithm; the algorithms used in each step can be specified independently. Suppose, for example, we want to use Hiton-PC with Pearson's X^2 test in the restrict step and tabu search with the BDe score and imaginary sample size

5 in the maximise step. They are, after all, among the best algorithms in the respective groups. We can do that as follows.

```
> rsmax2(survey, restrict = "si.hiton.pc", test = "x2",
+   maximize = "tabu", score = "bde",
+   maximize.args = list(iss = 5))
```

Clearly, even though any combination of constraint-based algorithms (for the restrict step), score-based algorithms (for the maximise step), conditional independence tests and network scores will work, some make more sense than others.

4.5.2 Parameter Learning

Once the structure of the BN has been learned from the data, the task of estimating and updating the parameters of the global distribution is greatly simplified by the decomposition into local distributions. Two approaches are common in literature: *maximum likelihood estimation*, and *Bayesian estimation*. Examples of both were covered in Section 1.4 (for discrete BNs) and Section 2.4 (for GBNs). Other choices, such as the shrinkage estimators presented in Hausser and Strimmer (2009) and Shäfer and Strimmer (2005) are certainly possible. It is important to note that the approach used to learn the structure of the BNs does necessarily determine which approaches can be used in parameter learning. For instance, using posterior densities in both structure and parameter learning makes the interpretation of the BN and its inference straightforward. However, using a Monte Carlo permutation test for structure learning and posterior estimates for parameter learning is also common, as is using shrinkage approaches and maximum likelihood parameter estimates.

Even though local distributions in practice involve only a small number of variables, and their dimension usually does not scale with the size of the BN, parameter estimation is still problematic in some situations. For example, it is increasingly common to have sample sizes much smaller than the number of variables included in the model. This is typical of high-throughput biological data sets, such as microarrays, that have a few ten or hundred observations and thousands of genes. In this setting, which is called "small n, large p", estimates have a high variability unless particular care is taken in both structure and parameter learning.

4.6 Bayesian Network Inference

Learning the structure and the parameters of a BN provides significant insights into the nature of the data. For instance, it highlights the dependence structure of the data and, under the right conditions, the causal structure as well. Furthermore, the parameters associated with each node provide a concise description of that node's behaviour relative to its parents. However, there are many questions that can only be answered by incorporating new evidence in the BN or by investigating the probability of complex events. Several examples of such questions can be found in Sections 1.6 (for discrete BNs) and 2.6 (for GBNs).

4.6.1 Probabilistic Reasoning and Evidence

BNs, like other statistical models, can be used to answer questions about the nature of the data that go beyond the mere description of the behaviour of the observed sample. The techniques used to obtain those answers are known in general as *inference*. For BNs, the process of answering these questions is also known as *probabilistic reasoning* or *belief updating*, while the questions themselves are called *queries*. Both names were introduced by Pearl (1988) and borrowed from expert systems theory (*e.g.*, you would submit a *query* to an *expert* to get an opinion, and *update your beliefs* accordingly), and have completely replaced traditional statistical terminology in recent texts such as Koller and Friedman (2009).

In practice, probabilistic reasoning on BNs works in the framework of Bayesian statistics and focuses on the computation of posterior probabilities or densities. For example, suppose we have learned a BN $\mathcal{B}$ with structure G and parameters Θ, under one of the distributional assumptions we explored in Chapters 1, 2 and 3. Subsequently, we want to use $\mathcal{B}$ to investigate the effects of a new piece of *evidence* $\mathbf{E}$ using the knowledge encoded in $\mathcal{B}$, that is, to investigate the posterior distribution $\Pr(\mathbf{X} \mid \mathbf{E}, \mathcal{B}) = \Pr(\mathbf{X} \mid \mathbf{E}, G, \Theta)$.

The approaches used for this kind of analysis vary depending on the nature of $\mathbf{E}$ and on the nature of information we are interested in. The two most common kinds of evidence are:

- *hard evidence*, an instantiation of one or more variables in the network. In other words,

  $$\mathbf{E} = \{X_{i_1} = e_1, X_{i_2} = e_2, \ldots, X_{i_k} = e_k\}, \qquad i_1 \neq \ldots \neq i_k \in \{1, \ldots n\}, \tag{4.23}$$

  which ranges from the value of a single variable X_i to a complete specification for $\mathbf{X}$. Such an instantiation may come, for instance, from a new (partial or complete) observation recorded after $\mathcal{B}$ was learned;

- *soft evidence*, a new distribution for one or more variables in the network. Since both the network structure and the distributional assumptions are treated as fixed, soft evidence is usually specified as a new set of parameters,

  $$\mathbf{E} = \left\{X_{i_1} \sim (\Theta_{X_{i_1}}), X_{i_2} \sim (\Theta_{X_{i_2}}), \ldots, X_{i_k} \sim (\Theta_{X_{i_k}})\right\}. \tag{4.24}$$

  This new distribution may be, for instance, the null distribution in a hypothesis testing problem: the use of a specific conditional probability table in case of discrete BNs or a zero value for some regression coefficients in case of GBNs.

As far as queries are concerned, we will focus on *conditional probability* (CPQ) and *maximum a posteriori* (MAP) queries, also known as *most probable explanation* (MPE) queries. Both apply mainly to hard evidence, even though they can be used in combination with soft evidence.

Conditional probability queries are concerned with the distribution of a subset of variables $\mathbf{Q} = \{X_{j_1}, \ldots, X_{j_l}\}$ given some hard evidence $\mathbf{E}$ on another set $X_{i_1}, \ldots, X_{i_k}$ of variables in $\mathbf{X}$. The two sets of variables can be assumed to be disjoint. The distribution we are interested in is

$$\mathrm{CPQ}(\mathbf{Q} \mid \mathbf{E}, \mathcal{B}) = \Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta) = \Pr(X_{j_1}, \ldots, X_{j_l} \mid \mathbf{E}, G, \Theta), \tag{4.25}$$

which is the marginal posterior probability distribution of $\mathbf{Q}$, *i.e.*,

$$\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta) = \int \Pr(\mathbf{X} \mid \mathbf{E}, G, \Theta) \, d(\mathbf{X} \setminus \mathbf{Q}). \tag{4.26}$$

This class of queries has many useful applications due to their versatility. For instance, it can be used to assess the interaction between two sets of experimental design factors for a trait of interest; the latter would be considered as the hard evidence E, while the former would play the role of the set of query variables $\mathbf{Q}$. As another example, the odds of an unfavourable outcome $\mathbf{Q}$ can be assessed for different sets of hard evidence $\mathbf{E}_1$, $\mathbf{E}_2$, $\ldots$, $\mathbf{E}_m$.

Maximum a posteriori queries are concerned with finding the configuration $\mathbf{q}^*$ of the variables in $\mathbf{Q}$ that has the highest posterior probability (for discrete BNs) or the maximum posterior density (for GBNs and hybrid BNs),

$$\mathrm{MAP}(\mathbf{Q} \mid \mathbf{E}, \mathcal{B}) = \mathbf{q}^* = \operatorname*{argmax}_{\mathbf{q}} \Pr(\mathbf{Q} = \mathbf{q} \mid \mathbf{E}, G, \Theta). \tag{4.27}$$

Applications of this kind of query fall into two categories: imputing missing data from partially observed hard evidence, where the variables in $\mathbf{Q}$ are not observed and are to be imputed from those in $\mathbf{E}$, or comparing $\mathbf{q}^*$ with the observed values for the variables in $\mathbf{Q}$ for completely observed hard evidence.

Both conditional probability queries and maximum a posteriori queries can also be used with soft evidence, albeit with different interpretations. On one hand, when $\mathbf{E}$ encodes hard evidence it is not stochastic; it is an observed value. In this case, $\Pr(\mathbf{Q} = \mathbf{q} \mid \mathbf{E}, G, \Theta)$ is not stochastic. On the other hand, when $\mathbf{E}$ encodes soft evidence it is still a random variable, and in turn $\Pr(\mathbf{Q} = \mathbf{q} \mid \mathbf{E}, G, \Theta)$ is stochastic. Therefore, the answers provided by the queries described in this section must be manipulated and evaluated according to the nature of the evidence they are based on.

It must be underlined that all techniques presented in this section are applications of the same basic principle: we modify the joint probability distribution of the nodes to incorporate a new piece of information. In the case of hard evidence, the distribution is conditioned on the values of some nodes; in the case of soft evidence some local distributions are modified.

Algorithm 4.4 Junction Tree Clustering Algorithm

1. **Moralise:** create the moral graph of the BN $\mathcal{B}$ as explained in Section 4.4.
2. **Triangulate:** break every cycle spanning 4 or more nodes into subcycles of exactly 3 nodes by adding arcs to the moral graph, thus obtaining a *triangulated graph*.
3. **Cliques:** identify the *cliques* $C_1, \ldots, C_k$ of the triangulated graph, *i.e.*, maximal subsets of nodes in which each element is adjacent to all the others.
4. **Junction Tree:** create a tree in which each clique is a node, and adjacent cliques are linked by arcs. The tree must satisfy the *running intersection property:* if a node belongs to two cliques C_i and C_j, it must be also included in all the cliques in the (unique) path that connects C_i and C_j.
5. **Parameters:** use the parameters of the local distributions of $\mathcal{B}$ to compute the parameter sets of the compound nodes of the junction tree.

4.6.2 Algorithms for Belief Updating

The estimation of the posterior probabilities and densities in the previous section is a fundamental problem in the evaluation of queries. Queries involving very small probabilities or large networks are particularly problematic even with the best algorithms available in literature, because they present both computational and probabilistic challenges.

Algorithms for belief updating can be classified either as *exact* or *approximate*. Both build upon the fundamental properties of BNs introduced in Section 4.2 to avoid the curse of dimensionality through the use of *local computations*, that is, by only using local distributions. So, for instance, the marginalisation in Equation (4.26) can be rewritten as

$$\begin{aligned}
\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta) &= \int \Pr(\mathbf{X} \mid \mathbf{E}, G, \Theta)\, d(\mathbf{X} \setminus \mathbf{Q}) \\
&= \int \left[\prod_{i=1}^{p} \Pr(X_i \mid \mathbf{E}, \Pi_{X_i}, \Theta_{X_i}) \right] d(\mathbf{X} \setminus \mathbf{Q}) \\
&= \prod_{i : X_i \in \mathbf{Q}} \int \Pr(X_i \mid \mathbf{E}, \Pi_{X_i}, \Theta_{X_i})\, dX_i. \qquad (4.28)
\end{aligned}$$

The correspondence between d-separation and conditional independence can also be used to further reduce the dimension of the problem. From Definition 4.2, variables that are d-separated from $\mathbf{Q}$ by $\mathbf{E}$ cannot influence the outcome

Algorithm 4.5 Logic Sampling Algorithm

```
1. Order the variables in $\mathbf{X}$ according to the topological partial ordering implied by $G$, say $X_{(1)} \prec X_{(2)} \prec \ldots \prec X_{(p)}$.
2. Set $n_{\mathbf{E}} = 0$ and $n_{\mathbf{E},\mathbf{q}} = 0$.
3. For a suitably large number of samples $\mathbf{x} = (x_1, \ldots, x_p)$:
   (a) generate $x_{(i)}, i = 1, \ldots, p$ from $X_{(i)} \mid \Pi_{X_{(i)}}$ taking advantage of the fact that, thanks to the topological ordering, by the time we are considering $X_i$ we have already generated the values of all its parents $\Pi_{X_{(i)}}$;
   (b) if $\mathbf{x}$ includes $\mathbf{E}$, set $n_{\mathbf{E}} = n_{\mathbf{E}} + 1$;
   (c) if $\mathbf{x}$ includes both $\mathbf{Q} = \mathbf{q}$ and $\mathbf{E}$, set $n_{\mathbf{E},\mathbf{q}} = n_{\mathbf{E},\mathbf{q}} + 1$.
4. Estimate $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$ with $n_{\mathbf{E},\mathbf{q}}/n_{\mathbf{E}}$.
```

of the query. Therefore, they may be completely disregarded in computing posterior probabilities.

Exact inference algorithms combine repeated applications of Bayes' theorem with local computations to obtain exact values $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$. Due to their nature, such algorithms are feasible only for small or very simple networks, such as trees and polytrees. In the worst case, their computational complexity is exponential in the number of variables.

The two best-known exact inference algorithms are *variable elimination* and belief updates based on *junction trees*. Both were originally derived for discrete networks, and have been later extended to the continuous and hybrid cases. Variable elimination uses the structure of the BN directly, specifying the optimal sequence of operations on the local distributions and how to cache intermediate results to avoid unnecessary computations. On the other hand, belief updates can also be performed by transforming the BN into a junction tree first. As illustrated in Algorithm 4.4, a junction tree is a transformation of the moral graph of $\mathcal{B}$ in which the original nodes are clustered to reduce any network structure into a tree. Subsequently, belief updates can be performed efficiently using Kim and Pearl's Message Passing algorithm. The derivation and the details of the major steps of this algorithm are beyond the scope of this book; for an exhaustive explanation and step-by-step examples, we refer the reader to Korb and Nicholson (2004) and Koller and Friedman (2009).

Approximate inference algorithms use Monte Carlo simulations to sample from the global distribution of $\mathbf{X}$ and thus estimate $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$. In particular, they generate a large number of samples from $\mathcal{B}$ and estimate the relevant conditional probabilities by weighting the samples that include both $\mathbf{E}$ and $\mathbf{Q} = \mathbf{q}$ against those that include only $\mathbf{E}$. In computer science, these random samples are often called *particles*, and the algorithms that make use of them are known as *particle filters* or *particle-based methods.*

The approaches used for both random sampling and weighting vary greatly, and their combination has resulted in several approximate algorithms being proposed in literature. Random sampling ranges from the generation of independent samples to more complex MCMC schemes, such as those we explored in Chapter 3. Common choices are either *rejection sampling* or *importance sampling*. Furthermore, weights range from uniform to likelihood functions to various estimates of posterior probability. The simplest combination of these two classes of approaches is known as either *forward* or *logic sampling*; it is described in Algorithm 4.5 and illustrated in detail in both Korb and Nicholson (2004) and Koller and Friedman (2009). Logic sampling combines rejection sampling and uniform weights, essentially counting the proportion of generated samples including $\mathbf{E}$ that also include $\mathbf{Q} = \mathbf{q}$.

Consider again, as an example, one of the queries we explored in Section 1.6.2.2: assessing from our survey the probability to find a man driving a car given he has high school education.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+            evidence = (E == "high"), n = 10^6)
[1] 0.34345
```

The first part of Algorithm 4.5 concerns the creation of the particles. In **bnlearn**, the sampling of random observations from a BN is implemented in the `rbn` function, which takes a `bn.fit` object and the number of samples to generate as arguments.

```
> particles <- rbn(bn, 10^6)
> head(particles, n = 5)
      A    E   O     R S     T
1 adult high emp   big M other
2 adult high emp   big F train
3 adult high emp   big M other
4 joung high emp small F train
5 joung high emp   big M   car
```

Calling `rbn` corresponds to steps 1 and 3(a). Step 3(b) requires finding the particles that match the evidence, `E == "high"`, and counting them to obtain $n_{\mathbf{E}}$.

```
> partE <- particles[(particles[, "E"] == "high"), ]
> nE <- nrow(partE)
```

Similarly, step 3(c) requires finding in `partE` those particles that also match the event we are investigating, `(S == "M") & (T == "car")`, and counting them to obtain $n_{\mathbf{E},\mathbf{q}}$.

```
> partEq <-
+   partE[(partE[, "S"] == "M") & (partE[, "T"] == "car"), ]
> nEq <- nrow(partEq)
```

Algorithm 4.6 Likelihood Weighting Algorithm

```
1. Order the variables in $\mathbf{X}$ according to the topological ordering implied by $G$, say $X_{(1)} \prec X_{(2)} \prec \ldots \prec X_{(p)}$.
2. Set $w_{\mathbf{E}} = 0$ and $w_{\mathbf{E},\mathbf{q}} = 0$.
3. For a suitably large number of samples $\mathbf{x} = (x_1, \ldots, x_p)$:
    (a) generate $x_{(i)}, i = 1, \ldots, p$ from $X_{(i)} \mid \Pi_{X_{(i)}}$ using the values $e_1, \ldots, e_k$ specified by the hard evidence $\mathbf{E}$ for $X_{i_1}, \ldots, X_{i_k}$.
    (b) compute the weight $w_{\mathbf{x}} = \prod \Pr(X_{i^*} = e_* \mid \Pi_{X_{i^*}})$
    (c) set $w_{\mathbf{E}} = w_{\mathbf{E}} + w_{\mathbf{x}}$;
    (d) if $\mathbf{x}$ includes $\mathbf{Q} = \mathbf{q}$ , set $w_{\mathbf{E},\mathbf{q}} = w_{\mathbf{E},\mathbf{q}} + w_{\mathbf{x}}$.
4. Estimate $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$ with $w_{\mathbf{E},\mathbf{q}}/w_{\mathbf{E}}$.
```

Finally, in step 4 we compute the conditional probability for the query as $n_{\mathbf{E},\mathbf{q}}/n_{\mathbf{E}}$.

```
> nEq/nE
[1] 0.34366
```

The discrepancy in the last digits is the natural consequence of a stochastic simulation. Clearly, such an algorithm can be very inefficient if $\Pr(\mathbf{E})$ is small, because most particles will be discarded without contributing to the estimation of $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$. However, its simplicity makes it easy to implement and very general in its application; it allows for very complex specifications of $\mathbf{E}$ and $\mathbf{Q}$ for both $\mathrm{MAP}(\mathbf{Q} \mid \mathbf{E}, \mathcal{B})$ and $\mathrm{CPQ}(\mathbf{Q} \mid \mathbf{E}, \mathcal{B})$.

An improvement over logic sampling, designed to solve this problem, is an application of importance sampling called *likelihood weighting* and illustrated in Algorithm 4.6. Unlike logic sampling, all the particles generated by likelihood weighting include the evidence $\mathbf{E}$ by design. However, this means that we are not sampling from the original BN any more, but we are sampling from a second BN in which all the nodes $X_{i_1}, \ldots, X_{i_k}$ in $\mathbf{E}$ are fixed. This network is called the *mutilated network*.

```
> mutbn <- mutilated(bn, list(E = "high"))
> mutbn$E

  Parameters of node E (multinomial distribution)

Conditional probability table:
 high  uni 
    1    0 
```

As a result, simply sampling from `mutbn` is not a valid approach. If we do so, the probability we obtain is $\Pr(\mathbf{Q}, \mathbf{E} \mid G, \Theta)$, not the conditional probability $\Pr(\mathbf{Q} \mid \mathbf{E}, G, \Theta)$.

```
> particles <- rbn(bn, 10^6)
> partQ <- particles[(particles[, "S"] == "M") &
+                    (particles[, "T"] == "car"), ]
> nQ <- nrow(partQ)
> nQ/10^6
[1] 0.33741
```

The role of the weights is precisely to adjust for the fact that we are sampling from `mutbn` instead of the original `bn`. As we can see from step 3(b), the weights are just the likelihood components associated with the nodes of `bn` we are conditioning on (`E` in this case) for the `particles`.

```
> w <- logLik(bn, particles, nodes = "E", by.sample = TRUE)
```

Having estimated the weights, we can now perform steps 3(c), 3(d) and 4 and obtain the estimated conditional probability for the query.

```
> wEq <- sum(exp(w[(particles[, "S"] == "M") &
+                  (particles[, "T"] == "car")]))
> wE <- sum(exp(w))
> wEq/wE
[1] 0.34275
```

The value of `wEq/wE` ratio is the same as the exact conditional probability we obtained using **gRain** in Section 1.6.2.1. It is a more precise estimate than that obtained above from logic sampling using the same number of particles.

More conveniently, we can perform likelihood weighting with `cpquery` by setting `method = "lw"` and specifying the evidence as a named list with one element for each node we are conditioning on.

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+           evidence = list(E = "high"), method = "lw")
[1] 0.34267
```

The estimate we obtain is still very precise despite the fact that we are not increasing the number of particles to 10^6 as we did for logic sampling.

At the other end of the spectrum, there are in literature more complex approximate algorithms that can estimate even very small probabilities with high precision. Two examples are the *adaptive importance sampling* (AIS-BN) scheme by Cheng and Druzdzel (2000) and the *evidence pre-propagation importance sampling* (EPIS-BN) by Yuan and Druzdzel (2003). Both can estimate conditional probabilities as small as 10^{-41}, and they also perform better on large networks. However, their assumptions often restrict them to discrete data and may require the specification of nontrivial tuning parameters.

4.7 Causal Bayesian Networks

Throughout this book, we have defined BNs in terms of conditional independence relationships and probabilistic properties, without any implication that arcs should represent cause-and-effect relationships. The existence of equivalence classes of networks indistinguishable from a probabilistic point of view provides a simple proof that arc directions are not indicative of causal effects.

However, from an intuitive point of view it can be argued that a "good" BN should represent the causal structure of the data it is describing. Such BNs are usually fairly sparse, and their interpretation is at the same time clear and meaningful, as explained by Pearl (2009) in his book on causality:

> It seems that if conditional independence judgments are byproducts of stored causal relationships, then tapping and representing those relationships directly would be a more natural and more reliable way of expressing what we know or believe about the world. This is indeed the philosophy behind causal BNs.

This is the reason why building a BN from expert knowledge in practice codifies known and expected causal relationships for a given phenomenon.

Learning such causal models, especially from observational data, presents significant challenges. In particular, three additional assumptions are needed:

- each variable $X_i \in \mathbf{X}$ is conditionally independent of its non-effects, both direct and indirect, given its direct causes. This assumption is called the *causal Markov assumption*, and represents the causal interpretation of the local Markov property in Definition 4.4;
- there must exist a DAG which is faithful to the probability distribution $\mathbf{P}$ of $\mathbf{X}$, so that the only dependencies in $\mathbf{P}$ are those arising from d-separation in the DAG;
- there must be no *latent variables* (unobserved variables influencing the variables in the network) acting as *confounding factors*. Such variables may induce spurious correlations between the observed variables, thus introducing bias in the causal network. Even though this is often listed as a separate assumption, it is really a corollary of the first two: the presence of unobserved variables violates the faithfulness assumption (because the network structure does not include them) and possibly the causal Markov property (if an arc is wrongly added between the observed variables due to the influence of the latent one).

These assumptions are difficult to verify in real-world settings, as the set of the potential confounding factors is not usually known. At best, we can address this issue, along with selection bias, by implementing a carefully planned experimental design.

Furthermore, even when dealing with interventional data collected from a scientific experiment (where we can control at least some variables and observe the resulting changes), there are usually multiple equivalent BNs that represent reasonable causal models. Many arcs may not have a definite direction, resulting in substantially different DAGs. When the sample size is small there may also be several non-equivalent BNs fitting the data equally well. Therefore, in general we are not able to identify a single, "best", causal BN but rather a small set of likely causal BNs that fit our knowledge of the data.

An example of the bias introduced by the presence of a latent variable was illustrated by Edwards (2000, page 113) using the `marks` data. `marks`, which is included in the **bnlearn** package, consists of the exam scores of 88 students across 5 different topics, namely: mechanics (`MECH`), vectors (`VECT`), algebra (`ALG`), analysis (`ANL`) and statistics (`STAT`). The scores are bounded in the interval $[0, 100]$. This data set was originally investigated by Mardia et al. (1979) and subsequently in classical books on graphical models such as Whittaker (1990).

```
> data(marks)
> head(marks)
  MECH VECT ALG ANL STAT
1   77   82  67  67   81
2   63   78  80  70   81
3   75   73  71  66   81
4   55   72  63  70   68
5   63   63  65  70   63
6   53   61  72  64   73
```

Edwards noted that the students apparently belonged to two groups (which we will call `A` and `B`) with substantially different academic profiles. He then assigned each student to one of those two groups using the EM algorithm to impute group membership as a latent variable (`LAT`). The EM algorithm assigned the first 52 students (with the exception of number 45) to belong to group `A`, and the remainder to group `B`.

```
> latent <- factor(c(rep("A", 44), "B",
+                   rep("A", 7), rep("B", 36)))
> modelstring(hc(marks[latent == "A", ]))
[1] "[MECH][ALG|MECH][VECT|ALG][ANL|ALG][STAT|ALG:ANL]"
> modelstring(hc(marks[latent == "B", ]))
[1] "[MECH][ALG][ANL][STAT][VECT|MECH]"
> modelstring(hc(marks))
[1] "[MECH][VECT|MECH][ALG|MECH:VECT][ANL|ALG][STAT|ALG:ANL]"
```

As we can see from Figure 4.5 (top left and top right panels) and from the `modelstring` calls above, the BNs learned from group `A` and group `B` are

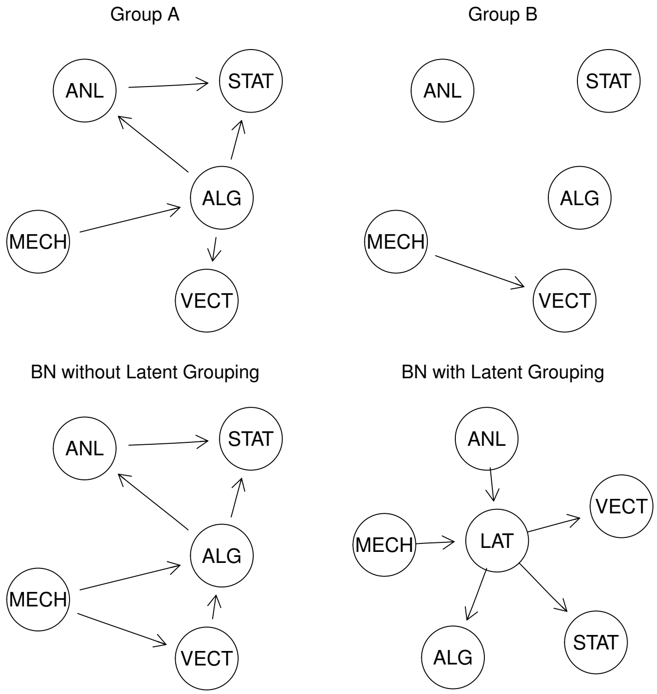


Figure 4.5
BNs learned from the `marks` data set when considering only group `A` (top left), when considering only group `B` (top right), when considering both (bottom left) and from discretised data set after the inclusion of the latent variable `LAT` (bottom right).

completely different. Furthermore, they are both different from the BN learned from the whole data set (bottom left panel).

If we want to learn a single BN while taking `LAT` into account, we can discretise the `marks` data and include the `latent` variable when learning the structure of the (now multinomial) BN. Again, we obtain a BN whose DAG (bottom right panel) is completely different from those above.

```
> dmarks <- discretize(marks, breaks = 2, method = "interval")
> modelstring(hc(cbind(dmarks, LAT = latent)))
[1] "[MECH][ANL][LAT|MECH:ANL][VECT|LAT][ALG|LAT][STAT|LAT]"
```

This BN provides a simple interpretation of the relationships between the topics: the grades in mechanics and analysis can be used to infer which group a student belongs to, and that in turn influences the grades in the remaining topics. We can clearly see that any causal relationship we would have inferred from a DAG learned without taking LAT into account would be potentially spurious. In fact, we could even question the assumption that the data are a random sample from a single population and have not been manipulated in some way.

4.8 Further Reading

Basic definitions and properties of BNs are detailed in many books, each with its own different perspective; but perhaps the most clear and concise are still the seminal works Pearl (1988) and Pearl (2009). We would also like to suggest Castillo et al. (1997) for a formal introduction to the theory of graphical models; and the tome from Koller and Friedman (2009), which is a very comprehensive if lengthy reference. Different aspects are also covered in Murphy (2012).

Various inference approaches are covered in different books: from variable elimination in Koller and Friedman (2009, Chapter 9) and Russell and Norvig (2009, Section 14.4); to junction trees in Koller and Friedman (2009, Chapter 10), Korb and Nicholson (2004, Chapter 3), Castillo et al. (1997, Chapter 8) and Koski and Noble (2009, Chapter 10); to logic sampling and likelihood weighting in Koller and Friedman (2009, Chapter 12), Korb and Nicholson (2004, Chapter 3) and Castillo et al. (1997, Chapter 9). Pearl (1988, Chapters 4 and 5) is also an interesting read about exact inference. In addition, Koller and Friedman (2009, Chapter 13) describes more inference algorithms specific to MAP queries.

Learning has fewer references compared to inference, and structure learning even fewer still. Parameter learning for discrete BNs is covered in most books, including Koller and Friedman (2009, Chapter 17) and Neapolitan (2003, Section 7.1); GBNs are covered in Neapolitan (2003, Section 7.2) and in Koller and Friedman (2009, Section 17.2.4). Constraint-based structure learning is introduced in Korb and Nicholson (2004, Chapter 8) and Neapolitan (2003, Chapter 10); Edwards (2000, Chapter 5) provides an exhaustive list of conditional independence tests to use in this context. Score-based structure learning is introduced in Korb and Nicholson (2004, Chapter 9) and Castillo et al. (1997, Chapter 11), and a more general take on the algorithms that can be used for this task is in Russell and Norvig (2009, Chapter 4). Hybrid structure learning is mentioned in Koski and Noble (2009, Section 6.3).

Finally, for the interested reader, we suggest Spirtes et al. (2000) as a very thorough reference to causal BNs and using constraint-based algorithms such as PC to learn them.

Exercises

Exercise 4.1 *Consider the* `survey` *data set from Chapter 1.*

1. Learn a BN with the IAMB algorithm and the asymptotic mutual information test.

2. Learn a second BN with IAMB but using only the first 100 *observations of the data set. Is there a significant loss of information in the resulting BN compared to the BN learned from the whole data set?*

3. Repeat the structure learning in the previous point with IAMB and the Monte Carlo and sequential Monte Carlo mutual information tests. How do the resulting networks compare with the BN learned with the asymptotic test? Is the increased execution time justified?

Exercise 4.2 *Consider again the* `survey` *data set from Chapter 1.*

1. Learn a BN using Bayesian posteriors for both structure and parameter learning, in both cases with `iss = 5`*.*

2. Repeat structure learning with `hc` *and* 3 *random restarts and with* `tabu`*. How do the BNs differ? Is there any evidence of numerical or convergence problems?*

3. Use increasingly large subsets of the `survey` *data to check empirically that BIC and BDe are asymptotically equivalent.*

Exercise 4.3 *Consider the* `marks` *data set from Section 4.7.*

1. Create a `bn` *object describing the graph in the bottom right panel of Figure 4.5 and call it* `mdag`*.*

2. Construct the skeleton, the CPDAG and the moral graph of `mdag`*.*

3. Discretise the `marks` *data using* `"interval"` *discretisation with* 2, 3 *and* 4 *intervals.*

4. Perform structure learning with `hc` *on each of the discretised data sets; how do the resulting DAGs differ?*

5

Software for Bayesian Networks

The difficulty of implementing versatile software for general classes of graphical models and the varying focus in different disciplines limit the applications of BNs compared to the state of the art in the literature. Nevertheless, the number of R packages for BNs has been slowly increasing in recent years. In this chapter we will provide an overview of available software packages, without pretending to be exhaustive, and we will introduce some classic R packages dealing with different aspects of BN learning and inference.

5.1 An Overview of R Packages

There are several packages on CRAN dealing with BNs; the versions used in writing this book are reported in parentheses below. They can be divided in two categories: those that implement structure and parameter learning and those that focus only on parameter learning and inference (see Table 5.1).

Packages **bnlearn** (version 3.5; Scutari, 2010), **deal** (version 1.2-37; Bøttcher and Dethlefsen, 2003), **pcalg** (version 1.1-6; Kalisch et al., 2012) and **catnet** (version 1.14.2; Balov and Salzman, 2013) fall into the first category.

bnlearn offers a wide variety of structure learning algorithms (spanning all the three classes covered in this book, with several tests and network scores), parameter learning approaches (maximum likelihood for discrete and continuous data, Bayesian estimation for discrete data) and inference techniques (cross-validation, bootstrap, conditional probability queries and prediction). It is also the only package that keeps a clear separation between the structure of a BN and the associated local probability distributions; they are implemented as two different classes of R objects.

deal implements structure and parameter learning using a Bayesian approach. It is one of the few R packages that handles BNs combining discrete and continuous nodes with a conditional Gaussian distribution. The variances of continuous nodes depend on their discrete parents, and continuous nodes are not allowed to be parents of discrete ones. The network structure is learned with the hill-climbing greedy search described in Algorithm 4.2, with the posterior density of the network as a score function and random restarts to avoid local maxima.

	bnlearn	**catnet**	**deal**	**pcalg**
discrete data	Yes	Yes	Yes	Yes
continuous data	Yes	No	Yes	Yes
mixed data	No	No	Yes	No
constraint-based learning	Yes	No	No	Yes
score-based learning	Yes	Yes	Yes	No
hybrid learning	Yes	No	No	No
structure manipulation	Yes	Yes	No	No
parameter estimation	Yes	Yes	Yes	Yes
prediction	Yes	Yes	No	No
approximate inference	Yes	No	No	No

	gRbase	**gRain**	**rbmn**
discrete data	Yes	Yes	No
continuous data	Yes	No	Yes
mixed data	No	No	No
constraint-based learning	No	No	No
score-based learning	No	No	No
hybrid learning	No	No	No
structure manipulation	Yes	No	No
parameter estimation	No	No	Yes
prediction	No	Yes	Yes
approximate inference	No	Yes	No

Table 5.1
Feature matrix for the R packages covered in Section 5.1.

pcalg provides a free software implementation of the PC algorithm, and it is specifically designed to estimate and measure causal effects. It handles both discrete and continuous data, and can account for the effects of latent variables on the network. The latter is achieved through a modified PC algorithm known as *Fast Causal Inference* (FCI), first proposed by Spirtes et al. (2000).

catnet models discrete BNs using frequentist techniques. Structure learning is performed in two steps. First, the node ordering of the DAG is learned from the data using simulated annealing; alternatively, a custom node ordering can be specified by the user. An exhaustive search is then performed among the network structures with the given node ordering and the exact maximum likelihood solution is returned. Parameter learning and prediction are also implemented. Furthermore, an extension of this approach for mixed data (assuming a Gaussian mixture distribution) has been made available from CRAN in package **mugnet** (version 1.01.3; Balov, 2013).

Packages **gRbase** (version 1.6-12; Højsgaard et al., 2013) and **gRain** (version 1.2-2; Højsgaard, 2013) fall into the second category. They focus on manipulating the parameters of the network, on prediction, and on inference, under the assumption that all variables are discrete. Neither **gRbase** nor **gRain** implement any structure or parameter learning algorithm, so the BN must be completely specified by the user.

rbmn (version 0.9-2; Denis, 2013) is devoted to linear GBNs and more specifically on deriving closed form expressions for the joint and conditional distributions of subsets of nodes. Conversion functions to and from **bnlearn** objects are available. No structure learning is implemented.

5.1.1 The deal Package

The **deal** package implements structure learning following step-by-step a classic Bayesian workflow, which we will illustrate using the `marks` data set from Section 4.7. First of all, we load the `marks` data set from **bnlearn** and we add the latent grouping, so that we have both discrete and continuous variables in the data.

```
> library(bnlearn)
> data(marks)
> latent <- factor(c(rep("A", 44), "B",
+                    rep("A", 7), rep("B", 36)))
> marks$LAT <- latent
```

The first step consists in defining an object of class `network` which identifies the variables in the BN and whether they are continuous or discrete.

```
> library(deal)
> net <- network(marks)
> net
## 6 ( 1 discrete+ 5 ) nodes;score=  ;relscore=
1        MECH       continuous()
2        VECT       continuous()
3        ALG       continuous()
4        ANL       continuous()
5        STAT       continuous()
6        LAT       discrete(2)
```

Subsequently, we define an uninformative prior distribution on `net` with `jointprior` and we set the imaginary sample size to 5.

```
> prior <- jointprior(net, N = 5)
Imaginary sample size: 5
```

Once we have defined the prior, we can perform a hill-climbing search for the DAG with the highest posterior probability using `autosearch`.

```
> net <- learn(net, marks, prior)$nw
> best <- autosearch(net, marks, prior)
```

The resulting posterior probability is a combination of BDe and BGe, since the data are assumed to follow a conditional Gaussian distribution. In particular:

- Discrete nodes can only have other discrete nodes as their parents, and in this case the local distribution looks like a local distribution for the BDe score.

- Continuous nodes can have both continuous and discrete parents. The part of the local distribution that derives from continuous parents has the same form as a local distribution for the BGe score, while the part that derives from discrete parents is constructed like a mixture.

It is important to note that **deal** has a `modelstring` function which produces a compact string representation of a DAG in the same format as **bnlearn**'s `modelstring`. This is very convenient to import and export DAGs between the two packages.

```
> mstring <- deal::modelstring(best$nw)
> mstring
[1] "[MECH|VECT:ALG][VECT|LAT][ALG|VECT:LAT][ANL|ALG:LAT]
                                             [STAT|ALG:ANL][LAT]"
```

As we can see in the output above, `LAT` is a root node, because it is the only discrete node and other nodes can only be its descendants. It is also important to note that since **deal** and **bnlearn** provide many functions with identical names, using the explicit `::` notation is advisable.

Using `compare` we can see that the DAG we just learned (`dag.deal`) is completely different from that in Section 4.7 (here `dag.bnlearn`).

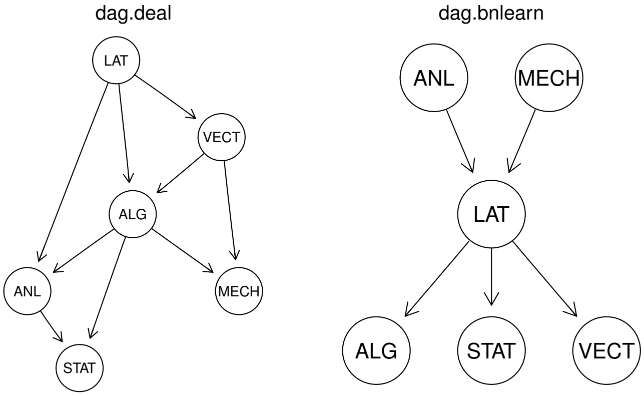


Figure 5.1
DAGs learned from the `marks` data with **deal** (left) and **bnlearn** (right) after adding latent grouping.

```
> dag.bnlearn <- model2network(
+      "[ANL][MECH][LAT|ANL:MECH][VECT|LAT][ALG|LAT][STAT|LAT]")
> dag.deal <- model2network(mstring)
> unlist(bnlearn::compare(cpdag(dag.deal), cpdag(dag.bnlearn)))
tp fp fn
 1  4  8
```

The two DAGs, which are shown in Figure 5.1, have CPDAGs that do not share any arc! In fact, `dag.deal` is quite similar to the GBN in the bottom left panel of Figure 4.5, which was learned before introducing `LAT`. This is, once more, an example of how different parametric assumptions and data transformation can deeply affect the results of BN learning.

5.1.2 The catnet Package

The **catnet** package also divides structure learning in discrete steps, but from a frequentist perspective: it learns the DAG that maximises BIC instead of posterior probability. Following Friedman and Koller (2003), **catnet** starts from the consideration that finding the optimal DAG given a topological ordering is a much simpler task than doing the same in the general case. Therefore, it implements the following workflow:

1. learn the optimal topological ordering from the data, or let the user specify one; and then
2. learn the optimal structure of the DAG conditional on the topological ordering.

The first step is implemented in the cnSearchSA function using simulated annealing, which we apply below to the discretised marks data from **bnlearn**. Learning can be customised with several optional arguments such as the maximum number of parents allowed for each node (maxParentSet and parentSizes), the maximum complexity of the network (maxComplexity) and the prior probability of inclusion of each arc (edgeProb).

```
> library(catnet)
> dmarks <- discretize(marks, breaks = 2, method = "interval")
> ord <- cnSearchSA(dmarks, maxParentSet = 2)
> ord
 Number of nodes    = 6,
 Sample size        = 88,
 Number of networks = 14
 Processing time    = 0.122
```

ord is an object of class catNetworkEvaluate, which contains a set of networks with the learned topological ordering, along with other useful quantities. We can access them as follows, and we can explore their properties with various **catnet** functions.

```
> nets <- ord@nets
> nets[[1]]
A catNetwork object with  6  nodes,  0  parents,  2  categories,
 Likelihood =  -3.889193 , Complexity =  6 .
```

The second step is implemented in cnFindBIC, which finds the network with the best BIC score among those stored in ord.

```
> best <- cnFindBIC(ord, nrow(dmarks))
> best
A catNetwork object with  6  nodes,  2  parents,  2  categories,
 Likelihood =  -3.046226 , Complexity =  12 .
```

Unlike **deal**, **catnet** provides functions to explore the learning process and the resulting networks and to make use of the latter. For instance, we can use cnSamples to generate numsamples random samples from best, which is useful to implement approximate inference. They are returned in a data frame with the same structure as that used to learn best.

```
> cnSamples(best, numsamples = 4)
           MECH        VECT         ALG         ANL       STAT LAT
1   (38.5,77.1] (45.5,82.1] (47.5,80.1] [8.94,39.5] [8.93,45]   B
2   (38.5,77.1] (45.5,82.1] [14.9,47.5] [8.94,39.5] [8.93,45]   B
3 [-0.077,38.5] [8.93,45.5] [14.9,47.5] [8.94,39.5] [8.93,45]   B
4   (38.5,77.1] (45.5,82.1] [14.9,47.5] [8.94,39.5] [8.93,45]   B
```

Another possibility is to extract the arc set from `best` using `cnMatEdges`, and import the DAG in **bnlearn**.

```
> em <- empty.graph(names(dmarks))
> arcs(em) <- cnMatEdges(best)
```

This makes it possible to use other functions in **bnlearn**, such as `bn.fit` to learn the parameters, and to export the BN to **gRain** to perform exact inference.

5.1.3 The pcalg Package

The **pcalg** package is unique in its focus on causal inference, and provides the only R implementation of the PC algorithm. Unlike other packages, it allows the user to provide a custom function to perform conditional independence tests. This means that, in principle, we can model any kind of data as long as we can define a suitable test function. `gaussCItest` (Student's t test for correlation), `condIndFisherZ` (Fisher's Z test), `gSquareDis` (G^2 test) and `disCItest` (Pearson's X^2 test) implement commonly used tests, and make the analysis of discrete BNs and GBNs possible out of the box.

However, this flexibility requires more effort in setting up structure learning: the user must produce the sufficient statistics for the conditional independence tests from the data as a preliminary step. In the case of the `marks` data (minus the latent grouping), this means computing the sample size and the correlation matrix of the variables.

```
> library(pcalg)
> marks <- marks[, colnames(marks) != "LAT"]
> suffStat <- list(C = cor(marks), n = nrow(marks))
```

We can then use `pc` with the specified test (`indepTest`) and type I error threshold (`alpha`) to learn the DAG.

```
> pc.fit <- pc(suffStat, indepTest = gaussCItest,
+             p = ncol(marks), alpha = 0.05)
```

Note that the data are never used directly, but only through the sufficient statistics in `suffStat`. The `pc.fit` object stores the DAG in an object of class `graphNEL`, which is defined in the **graph** package.

```
> pc.fit@graph
A graphNEL graph with directed edges
Number of Nodes = 5
Number of Edges = 7
```

As a result, it is easy to import the DAG in **bnlearn** with the `as.bn` function, or to manipulate and explore it directly with **graph** and **Rgraphviz**.

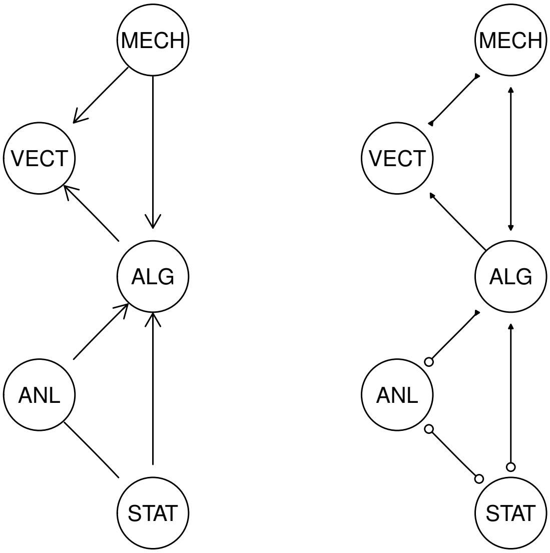


Figure 5.2
DAGs learned from the `marks` data with the PC (left) and FCI (right) algorithms without considering latent grouping.

The FCI algorithm, however, is more appropriate for the `marks` data because of the influence of the latent variable `LAT`. The syntax of the `fci` function is the same as that of `pc`, differing only in some optional arguments which are not used here.

```
> fci.fit <- fci(suffStat, indepTest = gaussCItest,
+                p = ncol(marks), alpha = 0.05)
```

A visual comparison of `pc.fit` and `fci.fit`, shown in Figure 5.2, makes it clear the two graphs should be interpreted differently even though they look very similar. `fci.fit` has several types of arrowheads, which are not present in `pc.fit`, to distinguish between the causal effects originating for observed variables as opposed to latent ones. A detailed treatment of their meaning, which would be quite involved, is beyond the scope of this overview; we refer the interested reader to **pcalg**'s documentation and the literature referenced therein.

5.2 BUGS Software Packages

In Chapter 3 we focused on JAGS, because it is well suited to run under the Linux systems that are common in modern computational facilities and because it has a convenient R interface in **rjags**. In addition, it has a very good numerical precision since it is based on R's math library. Other BUGS implementations exist that are even more popular than JAGS, but they are largely equivalent in terms of features, differing only in very minor details; two examples are OpenBUGS (Lunn et al., 2009) and WinBUGS (Lunn et al., 2000).

It is also worth mentioning a newcomer in the field of MCMC implementations using BNs to describe the data model: Stan (http://mc-stan.org; Stan Development Team, 2013). The documentation and the project look promising in terms of features and performance, but we have no direct experience in using it for applied research.

5.2.1 Probability Distributions

JAGS implements a large number of probability distributions, which can be used for the local distributions of nodes of a BN. Unfortunately, their parameterisation is sometimes subtly different from that used in R; we saw in Section 3.1 that precision is used to describe the spread of a normal distribution instead of the standard deviation. The reader is advised to be very careful; parameters may be valid even when they are misspecified and subsequent simulations will be incorrect, without generating any warning or error.

The following distributions are available:

- 15 real-valued distributions: Beta, Chi-square, Double exponential, Exponential, F, Gamma, Generalised gamma, Logistic, Log-normal, Non-central Chi-square, Normal, Pareto, Student's t, Uniform and Weibull.
- 7 discrete distributions: Beta-binomial, Bernoulli, Binomial, Categorical, Non-central hypergeometric, Negative binomial and Poisson.
- 5 multivariate distributions, either continuous: Dirichlet, Normal, Wishart, Student's t; or discrete: Multinomial.

See the JAGS manual (Plummer, 2012) for the complete details.

5.2.2 Complex Dependencies

The models we used in Chapter 3 are quite simple. BUGS allows random variables to be related via arbitrary mathematical expressions, including trigonometric functions, gamma and log-gamma functions, vector and matrix functions (such as the log-determinant of a square matrix or the inverse of a positive definite matrix). A complete reference is included in the JAGS manual (Plummer, 2012).

Mathematical expressions can be used to define the parameters of both mathematical functions and statistical distributions, as we saw for TR in the pest model:

```
TR ~ dbern(logit((G1 - m.TR)/s.TR));
```

Nodes with common characteristics can be defined in a single statement using loops, possibly nested. However, the number of iterations must be known and set when compiling the model. It must be an integer, a fixed parameter or a data value; not the current value of a node. This constraint is a consequence of the declarative nature of BUGS: statements are not parsed during the execution of the program, only at compile time. To clarify,

```
for (ii in 3:5) { node[ii] ~ dnorm(ii,1); }
```

is strictly equivalent to:

```
node[3] ~ dnorm(3,1);
node[4] ~ dnorm(4,1);
node[5] ~ dnorm(5,1);
```

5.2.3 Inference Based on MCMC Sampling

MCMC algorithms are not straightforward to use and analyse, and as is written in red in the first page of the WinBUGS help: "*Beware: MCMC sampling can be dangerous!*" The danger is that often we obtain numerical results but we have no way to check whether they are valid or not. It is beyond the scope of this book to cover this topic in much detail; we refer interested readers to Robert and Casella (2009). A few, important points are sketched below.

MCMC samples are drawn from the joint posterior distribution of the parameters or variables of interest. They are obtained with complicated (and self-calibrated, in BUGS software packages) iterative algorithms in which the values drawn at the ith iteration depend on those drawn at the $(i-1)$th iteration. Therefore:

- Successive draws are not independent, especially when the algorithm is said to be *not mixing well.* In that case many iterations are needed for some nodes to move across their support and generate markedly different values. In other words, samples are autocorrelated. Fortunately, autocorrelation can be easily checked and weakened by keeping one sample every k for a suitably large k. This process is called *thinning.*

- It can be shown that MCMC algorithms sample from the desired posterior distribution when the number of iterations tends to infinity. For this reason, we always perform a burn-in phase whose draws are discarded from the results. However, there is no systematic way of deciding when to stop the burn-in and start collecting MCMC samples. Not knowing the shape of the posterior distribution, we are not able to assess whether convergence has been attained or not. Only rough diagnostics are available. In some cases, several weeks of computations are needed to obtain similar results in JAGS and OpenBUGS!

- MCMC algorithms are iterative, so they need some initial values from which to start; this is called the *initialisation* of the algorithm. BUGS can generate them automatically; but sometimes this process fails with an error message like the following.

  ```
  Erreur dans jags.model(file = "pest.jam", data = dat1) :
   Error in node G2 Observed node inconsistent with
   unobserved parents at initialisation
  ```

  For this reason, we included a statement such as the following to define the model in our `R` code.

  ```
  ini <- list(G1 = 2);
  momar <- jags.model(file = "inclu.pest.jam", inits = ini,
             data = dat1)
  ```

  In other words, we use 2 as the initial value for `G1`, while letting JAGS initialise other nodes automatically. In this particular case, we can easily debug the error. Indeed, in Table 3.2, the distribution of `G2` is defined as a Poisson whose parameter is proportional to `G1`. When `G2` is positive and `G1`, which is again Poisson, is initialised to zero the values of the two nodes are inconsistent: the proposed pair (`G1 = 0`, `G2 > 0`) is invalid. The custom initialisation does not have this flaw.

  In the case of complicated models with a large number of nodes, deciding how to initialise them is far from trivial; a simple solution is to use their empirical estimates.

- MCMC algorithms are pseudo-random algorithms in the sense that they are based on the generation of pseudo-random numbers. As usual, it is strongly advised to save the seed which initialises the generator to be able to reproduce and investigate the results of the simulation.

5.3 Other Software Packages

5.3.1 BayesiaLab

BayesiaLab is produced by Bayesia (Laval, France) and provides an integrated workspace to handle BNs. Its main feature is a user-friendly and powerful graphical interface, which is essential to make BNs accessible to people without programming skills. Numerous types of plots and convenient reports can be generated. It can also import BN objects from other software and export Markov blankets to external programming languages.

The main goals of BayesiaLab are the translation of expert knowledge into a BN, the discovery of structure from data sets, and combining both to make the best possible use of available information. Temporal aspects can also be incorporated in the BN, and causal modelling for decision making can be performed by including specialised nodes. Even if some standard continuous distributions can be used to define the categories of the nodes, BNs are basically restricted to discrete variables.

In addition, BayesiaLab implements several algorithms to optimise the layout of the nodes for better visualising DAGs, and supports interactive manipulation of the proposed layouts.

Written in Java, it can be installed on all main platforms. An evaluation version limited to 10 nodes and $1,000$ database observations can be freely downloaded from

`http://library.bayesia.com/display/BlabC/Evaluation+License`.

5.3.2 Hugin

Hugin is a software package commercialised by Hugin Expert A/S and developed in collaboration with researchers from Aalborg University (Denmark). Its first release was a command-line tool created in the context of an ESPRIT project (Andersen et al., 1989) whose final outcome was the MUNIN expert system (Andreassen et al., 1989).

Modern versions of Hugin provide a graphical interface that allows users to perform BN learning and inference without the need of learning a programming language. It supports decision trees, discrete BNs, GBNs and hybrid BNs assuming a conditional Gaussian distribution. Exact inference is implemented using junction trees (Algorithm 4.4), and is complemented by a sensitivity analysis of the resulting posterior probabilities. Furthermore, missing values are supported both in learning and inference using the EM algorithm.

A demo version of Hugin can be freely downloaded from

`http://www.hugin.com/productsservices/demo/hugin-lite`,

but is limited to handle at most 50 states and learn BNs from samples of at most 500 cases. An interface to R is provided by package **RHugin**, which can be downloaded from:

`http://rhugin.r-forge.r-project.org`.

5.3.3 GeNIe

GeNIe and its underlying library SMILE (Structural Modelling, Inference, and Learning Engine) are developed by Druzdzel's Decision Systems Laboratory. The former provides an intuitive graphical interface to the latter. SMILE is cross-platform and very powerful, but requires working knowledge of either C++, .NET or Java. On the other hand, GeNIe only runs on Windows.

GeNIe focuses on inference, and implements several exact and approximate algorithms such as the Adaptive Importance Sampling (AIS-BN) from Cheng and Druzdzel (2000). Both discrete BNs and GBNs are supported. As far as structure learning is concerned, GeNIe implements naive Bayes classifiers, the PC algorithm and two greedy search heuristics using BDe as a network score.

Binaries for both GeNIe and SMILE can be freely downloaded from

`http://genie.sis.pitt.edu/index.php/downloads`;

note though that both are proprietary software, so sources are not available.

Exercises

Exercise 5.1 *One essential task in any analysis is to import and export the R objects describing models from different packages. This is all the more true in the case of BN modelling, as no package implements all of structure learning, parameter learning and inference.*

1. *Create the* `dag.bnlearn` *object from Section 5.1.1.*
2. *Export it to* **deal**.
3. *Import the result back into* **bnlearn**.
4. *Export* `dag.bnlearn` *to* **catnet** *and import it back in* **bnlearn**.
5. *Perform parameter learning using the discretised* `dmarks` *and* `dag.bnlearn` *and export it to a DSC file, which can be read in Hugin and GeNIe.*

Exercise 5.2 *Learn a GBN from the* `marks` *data (without the* `LAT` *variable) using* **pcalg** *and a custom test that defines dependence as significant if the corresponding partial correlation is greater than* 0.50.

Exercise 5.3 *Reproduce the example of structure learning from Section 5.1.1 using* **deal**, *but set the imaginary sample size to* 20. *How does the resulting network change?*

6

Real-World Applications of Bayesian Networks

We will now apply the concepts we introduced in the previous chapters to the analysis of two real-world data sets from life sciences: a protein-signalling data set from which we want to discover interactions and pathways characterising some biological processes in human cells, and a medical diagnostic data set which we will use as a basis to predict a human's body composition.

6.1 Learning Protein-Signalling Networks

BNs provide a versatile tool for the analysis of many kinds of biological data, such as *single-nucleotide polymorphism* (SNP) data and *gene expression* profiles. Following the work of Friedman et al. (2000), the expression level or the allele frequency of each gene is associated with one node. In addition, we can include additional nodes denoting other attributes that affect the system, such as experimental conditions, temporal indicators, and exogenous cellular conditions. As a result, we can model in a single, comprehensive BN both the biological mechanisms we are interested in and the external conditions influencing them at the same time. Some interesting applications along these lines are studied, among others, in Yu et al. (2004); Zou and Conzen (2005); Morota et al. (2012); Villa-Vialaneix et al. (2013); Hartley and Sebastiani (2013).

An outstanding example of how such data can be analysed effectively is presented in Sachs et al. (2005) using protein-signalling data. BNs were used to represent complex direct and indirect relationships among multiple interacting molecules while accommodating biological noise. The data consist in the simultaneous measurements of 11 phosphorylated proteins and phospholipids derived from thousands of individual primary immune system cells, subjected to both general and specific molecular interventions. The former ensure that the relevant signalling pathways are active, while the latter make causal inference possible by elucidating arc directions through stimulatory cues and inhibitory interventions.

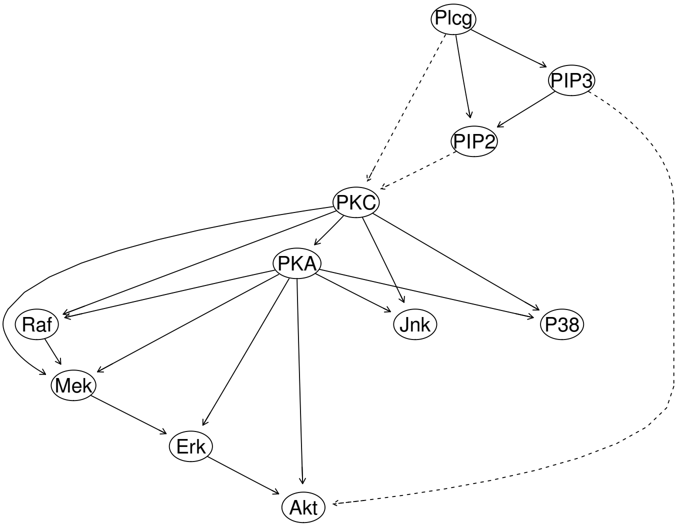


Figure 6.1
Protein-signalling network from Sachs et al. (2005). Signalling pathways that are known from literature but were not captured by the BN are shown with a dashed line.

The analysis described in Sachs et al. (2005) can be summarised as follows.

1. Outliers were removed and the data were discretised using the approach described in Hartemink (2001), because the distributional assumptions required by GBNs did not hold.
2. Structure learning was performed multiple times. In this way, a larger number of DAGs were explored in an effort to reduce the impact of locally optimal (but globally suboptimal) networks on learning and subsequent inference.
3. The DAGs learned in the previous step were averaged to produce a more robust model. This practice, known as *model averaging*, is known to result in a better predictive performance than choosing a single, high-scoring BN. The averaged DAG was created using the arcs present in at least 85% of the DAGs. This proportion measures the *strength* of each arc and provides the means to establish its *significance* given a *threshold* (85% in this case).
4. The validity of the averaged BN was evaluated against well-established signalling pathways from literature.

The final BN is shown in Figure 6.1. It includes 11 nodes, one for each protein and phospholipid: `PKC`, `PKA`, `Raf`, `Mek`, `Erk`, `Akt`, `Jnk`, `P38`, `Plcg` (standing for

Plcγ), PIP2 and PIP3. Every arc in that BN has been successfully validated, with very few exceptions:

- the arc between Pclg and PIP3 should be oriented in the other direction;
- there is a missing arc from PIP3 to Akt;
- there are two missing arcs from Plcg and PIP2 to PKC.

We will reproduce these steps in the remainder of this section, using **bnlearn** and other packages covered in Section 5.1 to integrate missing functionality.

6.1.1 A Gaussian Bayesian Network

For the moment, we will consider only the 853 data manipulated with general interventions (*i.e.*, the observational data); we will investigate the complete data set (*i.e.*, both the observational and the interventional data) in Section 6.1.5.

```
> library(bnlearn)
> sachs <- read.table("sachs.data.txt", header = TRUE)
> head(sachs)
   Raf   Mek  Plcg  PIP2  PIP3   Erk  Akt PKA   PKC  P38  Jnk
1 26.4 13.20  8.82 18.30 58.80  6.61 17.0 414 17.00 44.9 40.0
2 35.9 16.50 12.30 16.80  8.13 18.60 32.5 352  3.37 16.5 61.5
3 59.4 44.10 14.60 10.20 13.00 14.90 32.5 403 11.40 31.9 19.5
4 73.0 82.80 23.10 13.50  1.29  5.83 11.8 528 13.70 28.6 23.1
5 33.7 19.80  5.19  9.73 24.80 21.10 46.1 305  4.66 25.7 81.3
6 18.8  3.75 17.60 22.10 10.90 11.90 25.7 610 13.70 49.1 57.8
```

The data are continuous, as they represent the concentration of the molecules under investigation. Therefore, the standard approach in literature is to assume the concentrations follow a Gaussian distribution and to use a GBN to build the protein-signalling network.

However, the DAGs learned under this parametric assumption are not satisfactory. For example, using Inter-IAMB we obtain a DAG with only 8 arcs (compared to the 17 in Figure 6.1) and only 2 of them are directed.

```
> dag.iamb <- inter.iamb(sachs, test = "cor")
> narcs(dag.iamb)
[1] 8
> directed.arcs(dag.iamb)
     from  to
[1,] "P38" "PKC"
[2,] "Jnk" "PKC"
```

Other combinations of constraint-based algorithms and conditional independence tests, such as gs with test = "mc-cor", return the same DAG. The same is true for score-based and hybrid algorithms. If we compare dag.iamb with the DAG from Figure 6.1 (minus the arcs that were missed in the original paper), we can see that they have completely different structures.

```
> sachs.modelstring <-
+   paste("[PKC][PKA|PKC][Raf|PKC:PKA][Mek|PKC:PKA:Raf]",
+         "[Erk|Mek:PKA][Akt|Erk:PKA][P38|PKC:PKA]",
+         "[Jnk|PKC:PKA][Plcg][PIP3|Plcg][PIP2|Plcg:PIP3]")
> dag.sachs <- model2network(sachs.modelstring)
> unlist(compare(dag.sachs, dag.iamb))
tp fp fn
 0  8 17
```

Comparing the two DAGs again, but disregarding arc directions, reveals that some of the dependencies are correctly identified by inter.iamb but their directions are not.

```
> unlist(compare(skeleton(dag.sachs), skeleton(dag.iamb)))
tp fp fn
 8  0  9
```

The reason for the discrepancy between dag.sachs and dag.iamb is apparent from a graphical exploratory analysis of the data. Firstly, the empirical distributions of the molecules' concentrations are markedly different from a normal distribution. As an example, we plotted the distributions of Mek, P38, PIP2 and PIP3 in Figure 6.2. All are strongly skewed, because concentrations are positive numbers but a lot of them are small, and therefore cluster around zero. As a result, the variables are not symmetric and clearly violate the distributional assumptions underlying GBNs. Secondly, the dependence relationships in the data are not always linear; this is the case, as shown in Figure 6.3, for PKA and PKC. Most conditional independence tests and network scores designed to capture linear relationships have very low power in detecting nonlinear ones. In turn, structure learning algorithms using such statistics are not able to correctly learn the arcs in the DAG.

6.1.2 Discretising Gene Expressions

Since we have concluded that GBNs are not appropriate for the Sachs et al. (2005) data, we must now consider some alternative parametric assumptions. One possibility could be to explore monotone transformations like the logarithm. Another possibility would be to specify an appropriate conditional distribution for each variable, thus obtaining a hybrid network like those we explored in Chapter 3. However, this approach requires substantial prior knowledge on the signalling pathways, which may or may not be available.

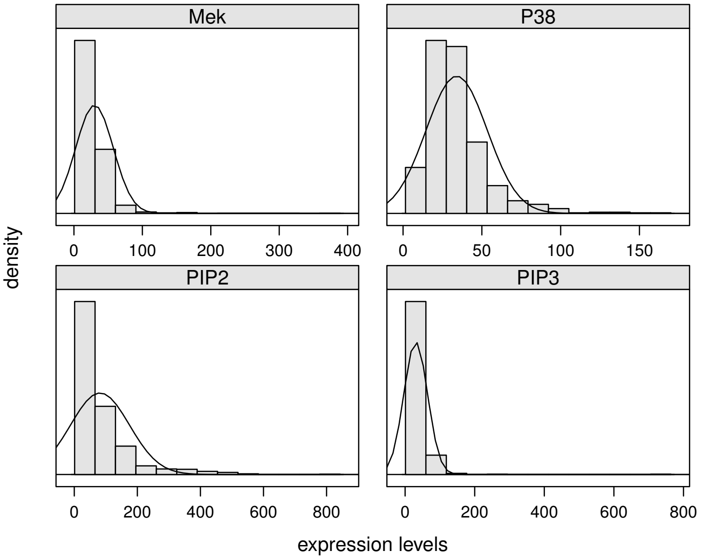


Figure 6.2
Densities of `Mek`, `P38`, `PIP2` and `PIP3`. Histograms represent the empirical densities computed from the `sachs` data, while the curves are normal density functions with the appropriate means and variances. Note that on the right side of each histogram there are almost imperceptible bars, justifying the range of the abscissas and making the normal distribution completely irrelevant.

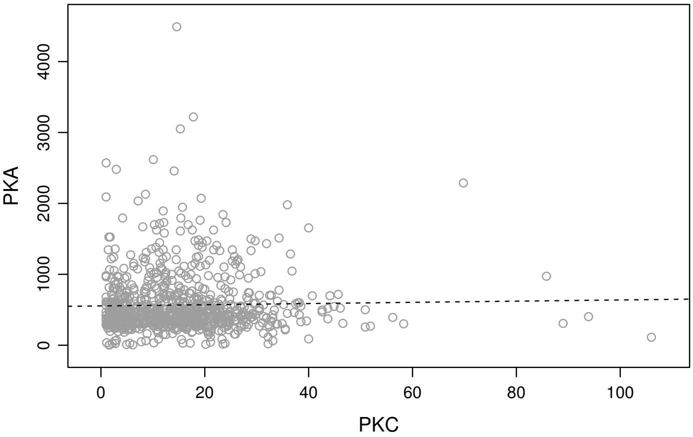


Figure 6.3
The (nonlinear) relationship between `PKA` and `PKC`; the dashed line is the fitted regression line for `PKA` against `PKC`.

Algorithm 6.1 Hartemink's Information-Preserving Discretisation

```
1. Discretise each variable independently using quantile discretisation
   and a large number $k_1$ of intervals, e.g., $k_1 = 50$ or even $k_1 = 100$.
2. Repeat the following steps until each variable has $k_2 \ll k_1$ intervals,
   iterating over each variable $X_i$, $i = 1, \ldots, p$ in turn:
   (a) compute
       $$\mathrm{M}_{X_i} = \sum_{j \neq i} \mathrm{MI}(X_i, X_j);$$
   (b) for each pair $l$ of adjacent intervals of $X_i$, collapse them in a
       single interval, and with the resulting variable $X_i^*(l)$ compute
       $$\mathrm{M}_{X_i^*(l)} = \sum_{j \neq i} \mathrm{MI}(X_i^*(l), X_j);$$
   (c) set $X_i = \mathrm{argmax}_{X_i(l)} \mathrm{M}_{X_i^*(l)}$.
```

In the case of Sachs et al. (2005), such information was indeed available from literature. However, the aim of the analysis was to use BNs as an automated probabilistic method to verify such information, not to build a BN with prior information and use it as an expert system.

Consequently, Sachs et al. (2005) decided to *discretise* the data and to model them with a discrete BN, which can accommodate skewness and non-linear relationships at the cost of losing the ordering information. Since the variables in the BN represent concentration levels, it is intuitively appealing to discretise them into three levels corresponding to *low*, *average* and *high* concentrations.

To that end, we can use the `discretize` function in **bnlearn**, which implements three common discretisation methods:

- *quantile discretisation*: each variable is discretised independently into k intervals delimited by its $0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{(k-1)}{k}, 1$ empirical quantiles;
- *interval discretisation*: each variable is discretised independently into k equally-spaced intervals;
- *information-preserving discretisation*: variables are jointly discretised while preserving as much of the pairwise mutual information between the variables as possible.

The last approach has been introduced by Hartemink (2001), and is described in detail in Algorithm 6.1. The key idea is to initially discretise each variable into a large number k_1 of intervals, thus losing as little information as possible. Subsequently, the algorithm iterates over the variables and collapses, for each of them, the pair of adjacent intervals that minimises the loss of pairwise mutual information. The algorithm stops when all variables have $k_2 \ll k_1$ intervals left. The resulting set of discretised variables reflects the dependence structure of the original data much better than either quantile or interval discretisation would allow, because the discretisation takes pairwise dependencies into account. Clearly some information is always lost in the process; for instance, higher-level dependencies are completely disregarded and therefore are not likely to be preserved.

Algorithm 6.1 is implemented in the `discretize` function (`method = "hartemink"`) along with quantile (`method = "quantile"`) and interval discretisation (`method = "interval"`).

```
> dsachs <- discretize(sachs, method = "hartemink",
+              breaks = 3, ibreaks = 60, idisc = "quantile")
```

The relevant arguments are `idisc` and `ibreaks`, which control how the data are initially discretised, and `breaks`, which specifies the number of levels of each discretised variable. `ibreaks` corresponds to k_1 in Algorithm 6.1, while `breaks` corresponds to k_2. Choosing good values for these arguments is a trade-off between quality and speed; high values of `ibreaks` preserve the characteristics of the original data to a greater extent, whereas smaller values result in much smaller memory usage and shorter running times.

6.1.3 Model Averaging

Following the analysis in Sachs et al. (2005), we can improve the quality of the structure learned from the data by averaging multiple CPDAGs. One possible approach to that end is to apply bootstrap resampling to `dsachs` and learn a set of 500 network structures.

```
> boot <- boot.strength(dsachs, R = 500, algorithm = "hc",
+            algorithm.args = list(score = "bde", iss = 10))
```

In the code above we learn a CPDAG with hill-climbing from each of the `R` bootstrap samples. Each variable in `dsachs` is now a factor with three levels, so we use the BDe score with a very low imaginary sample size.

The `boot` object returned by `boot.strength` is a data frame containing the strength of all possible arcs (in the `strength` column) and the probability of their direction (in the `direction` column) conditional on the fact that the `from` and `to` nodes are connected by an arc. This structure makes it easy to select the most significant arcs, as below.

```
> boot[(boot$strength > 0.85) & (boot$direction >= 0.5), ]
   from   to strength direction
1   Raf  Mek    1.000  0.520000
23 Plcg PIP2    1.000  0.517000
```

```
24  Plcg PIP3    1.000  0.531000
34  PIP2 PIP3    1.000  0.511000
56   Erk  Akt    1.000  0.568000
57   Erk  PKA    0.994  0.588531
67   Akt  PKA    1.000  0.586000
89   PKC  P38    1.000  0.512000
90   PKC  Jnk    1.000  0.511000
100  P38  Jnk    0.958  0.506263
```

Arcs are considered significant if they appear in at least 85% of the networks, and in the direction that appears most frequently. Arcs that are score equivalent in all the CPDAGs are considered to appear 50% of the time in each direction. Since all the values in the `direction` column above are close to 0.5, we can infer that the direction of the arcs is not well established and that they are probably all score equivalent. Interestingly, lowering the threshold from 85% to 50% does not change the results of the analysis, which seems to indicate that in this case the results are not sensitive to its value.

Having computed the significance for all possible arcs, we can now build the averaged network using `averaged.network` and the 85% threshold.

```
> avg.boot <- averaged.network(boot, threshold = 0.85)
```

The averaged network `avg.boot` contains the same arcs as the network learned from the observational data in Sachs et al. (2005), which is shown in Figure 6.4. Even though the probability of the presence of each arc and of its possible directions are computed separately in `boot.strength`, we are not able to determine with any confidence which direction has better support from the discretised data. Therefore, we remove the directions from the arcs in `avg.boot`, which amounts to constructing its skeleton.

```
> avg.boot <- skeleton(avg.boot)
```

As an alternative, we can average the results of several hill-climbing searches, each starting from a different DAG. Such DAGs can be generated randomly from a uniform distribution over the space of connected graphs with the MCMC algorithm proposed by Ide and Cozman (2002) and implemented in `random.graph` as `method = "ic-dag"`. This ensures that no systematic bias is introduced in the learned DAGs. In addition, keeping only one randomly generated DAG every 50 ensures that the DAGs are different from each other so that the search space is covered as thoroughly as possible.

```
> nodes <- names(dsachs)
> start <- random.graph(nodes = nodes, method = "ic-dag",
+              num = 500, every = 50)
> netlist <- lapply(start, function(net) {
+   hc(dsachs, score = "bde", iss = 10, start = net)
+ })
```

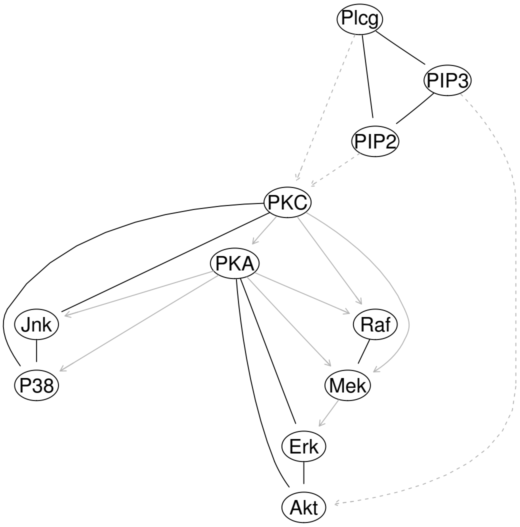


Figure 6.4
The undirected graph learned from the observational data in Sachs et al. (2005). Missing arcs from the DAG in Figure 6.1 are plotted in grey.

After using `lapply` to iterate easily over the DAGs in the `start` list and pass each of them to `hc` with the `start` argument, we obtain a list of `bn` objects, which we call `netlist`. We can pass such a list to the `custom.strength` function to obtain a data frame with the same structure as those returned by `boot.strength`.

```
> rnd <- custom.strength(netlist, nodes = nodes)
> rnd[(rnd$strength > 0.85) & (rnd$direction >= 0.5), ]
> avg.start <- averaged.network(rnd, threshold = 0.85)
```

```
    from   to strength direction
11   Mek  Raf        1     0.509
33  PIP2 Plcg        1     0.509
34  PIP2 PIP3        1     0.508
43  PIP3 Plcg        1     0.501
56   Erk  Akt        1     0.504
57   Erk  PKA        1     0.503
67   Akt  PKA        1     0.500
77   PKA  Akt        1     0.500
90   PKC  Jnk        1     0.502
99   P38  PKC        1     0.505
100  P38  Jnk        1     0.507
```

The arcs identified as significant with this approach are the same as in avg.boot (even though some are reversed), thus confirming the stability of the averaged network obtained from bootstrap resampling. Arc directions are again very close to 0.5, to the point we can safely disregard them. A comparison of the equivalence classes of avg.boot and avg.start shows that the two networks are equivalent.

```
> all.equal(cpdag(avg.boot), cpdag(avg.start))
[1] "Different number of directed/undirected arcs"
```

Furthermore, note how averaged networks, like the networks they are computed from, are not necessarily completely directed. In that case, it is not possible to compute their score directly. However, we can identify the equivalence class the averaged network belongs to (with cpdag) and then select a DAG within that equivalence class (with cextend).

```
> score(cextend(cpdag(avg.start)), dsachs, type = "bde",
+   iss = 10)
[1] -8498.88
```

Since all networks in the same equivalence class have the same score (for score-equivalent functions), the value returned by score is a correct estimate for the original, partially directed network.

We can also compute averaged network structures from bootstrap samples using the algorithms implemented in **catnet**, **deal** and **pcalg**. For this purpose, and to facilitate interoperability, we can again use the custom.strength function. In addition to a list of bn objects, custom.strength also accepts a list of arc sets, which can be created using functions from other packages, and returns a data frame with the same columns as that returned by boot.strength. For example, we can replace the hill-climbing search used above with the simulated annealing search implemented in **catnet** as follows.

```
> library(catnet)
> netlist <- vector(500, mode = "list")
> ndata <- nrow(dsachs)
> nodes <- names(dsachs)
> netlist <- lapply(netlist, function(net) {
+   boot <- dsachs[sample(ndata, replace = TRUE), ]
+   top.ord <- cnSearchOrder(boot)
+   best <- cnFindBIC(top.ord, ndata)
+   cnMatEdges(best)
+ })
> sann <- custom.strength(netlist, nodes = nodes)
```

The code above is similar to that used to create avg.start. After creating an empty list with 500 slots to hold the arc sets, we iterate over it using lapply.

At each iteration:

1. we create the bootstrap sample `boot` by subsetting the original data frame `dsachs` with the `sample` function and `replace = TRUE`;
2. we learn the topological ordering `top.ord` of the nodes from the bootstrap sample using `cnSearchOrder` from **catnet**;
3. we learn (again from the data) the DAG with the best BIC score given `top.ord` using `cnFindBIC` from **catnet**;
4. we extract the arcs from `best` using `cnMatEdges` from **catnet**.

Finally, we perform model averaging with `custom.strength` from **bnlearn**. The result is stored in the `sann` object, which we can investigate as before.

```
> sann[(sann$strength > 0.85) & (sann$direction >= 0.5), ]
    from   to strength direction
1    Raf  Mek     1.00       0.5
11   Mek  Raf     1.00       0.5
23  Plcg PIP2     0.99       0.5
33  PIP2 Plcg     0.99       0.5
34  PIP2 PIP3     1.00       0.5
44  PIP3 PIP2     1.00       0.5
56   Erk  Akt     1.00       0.5
66   Akt  Erk     1.00       0.5
67   Akt  PKA     1.00       0.5
77   PKA  Akt     1.00       0.5
89   PKC  P38     1.00       0.5
90   PKC  Jnk     1.00       0.5
99   P38  PKC     1.00       0.5
109  Jnk  PKC     1.00       0.5
> avg.catnet <- averaged.network(sann, threshold = 0.85)
```

Unlike `avg.boot` and `avg.start`, `avg.catnet` is clearly an undirected graph; both directions have probability of exactly 0.5 for every arc, because the 500 DAGs learned from the bootstrap samples do not contain any v-structure. Furthermore, we can see with `narcs` that `avg.catnet` only has 7 arcs compared to the 10 of `avg.start`.

```
> narcs(avg.catnet)
[1] 7
> narcs(avg.start)
[1] 10
```

It seems also not to fit the data very well; the BDe score returned by `score` for the same `iss` we used in structure learning is higher for `avg.start` than for `avg.catnet`.

```
> score(cextend(cpdag(avg.catnet)), dsachs, type = "bde",
+   iss = 10)
[1] -8548.34
> score(cextend(cpdag(avg.start)), dsachs, type = "bde",
+   iss = 10)
[1] -8498.88
```

Such differences can be attributed to the different scores and structure learning algorithms used to build the sets of high-scoring networks. In particular, it is very common for arc directions to differ between learning algorithms.

6.1.4 Choosing the Significance Threshold

The value of the threshold beyond which an arc is considered significant, which is called the *significance threshold*, does not seem to have a huge influence on the analysis in Sachs et al. (2005). In fact, any value between 0.5 and 0.85 yields exactly the same results. So, for instance:

```
> all.equal(averaged.network(boot, threshold = 0.50),
+           averaged.network(boot, threshold = 0.70))
[1] TRUE
```

The same holds for `avg.catnet` and `avg.start`. However, this is often not the case. Therefore, it is important to use a statistically motivated algorithm for choosing a suitable threshold instead of relying on ad-hoc values.

A solution to this problem is presented in Scutari and Nagarajan (2013), and implemented in **bnlearn** as the default value for the `threshold` argument in `averaged.network`. It is used when we do not specify `threshold` ourselves as in the code below.

```
> averaged.network(boot)

  Random/Generated Bayesian network

  model:
   [Raf][Plcg][Erk][PKC][Mek|Raf][PIP2|Plcg][Akt|Erk]
   [P38|PKC][Jnk|PKC][PIP3|Plcg:PIP2][PKA|Erk:Akt]
  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:                0.958
```

The value of the threshold is computed as follows. If we denote the arc strengths stored in `boot` as $\hat{\mathbf{p}} = \{\hat{p}_i, i = 1, \ldots, k\}$, and $\hat{\mathbf{p}}_{(\cdot)}$ is

$$\hat{\mathbf{p}}_{(\cdot)} = \{0 \leqslant \hat{p}_{(1)} \leqslant \hat{p}_{(2)} \leqslant \ldots \leqslant \hat{p}_{(k)} \leqslant 1\}, \tag{6.1}$$

then we can define the corresponding arc strength in the (unknown) averaged network $G = (\mathbf{V}, A_0)$ as

$$\tilde{p}_{(i)} = \begin{cases} 1 & \text{if } a_{(i)} \in A_0 \\ 0 & \text{otherwise} \end{cases}, \tag{6.2}$$

that is, the set of strengths that characterises any arc as either significant or non-significant without any uncertainty. In other words,

$$\tilde{\mathbf{p}}_{(\cdot)} = \{0, \ldots, 0, 1, \ldots, 1\}. \tag{6.3}$$

The proportion t of elements of $\tilde{\mathbf{p}}_{(\cdot)}$ that are equal to 1 determines the number of arcs in the averaged network, and is a function of the significance threshold we want to estimate. One way to do that is to find the value $\hat{t}$ that minimises the L_1 norm

$$L_1\left(t; \hat{\mathbf{p}}_{(\cdot)}\right) = \int \left| F_{\hat{\mathbf{p}}_{(\cdot)}}(x) - F_{\tilde{\mathbf{p}}_{(\cdot)}}(x; t) \right| dx \tag{6.4}$$

between the cumulative distribution functions of $\tilde{\mathbf{p}}_{(\cdot)}$ and $\hat{\mathbf{p}}_{(\cdot)}$, and then to include every arc that satisfies

$$a_{(i)} \in A_0 \Longleftrightarrow \hat{p}_{(i)} > F^{-1}_{\hat{\mathbf{p}}_{(\cdot)}}(\hat{t}) \tag{6.5}$$

in the average network. This amounts to finding the averaged network whose arc set is "closest" to the arc strength computed from the data, with $F^{-1}_{\hat{\mathbf{p}}_{(\cdot)}}(\hat{t})$ acting as the significance threshold.

For the `dsachs` data, the estimated value for the threshold is 0.958; so, any arc with a `strength` value strictly greater than that is considered significant. The resulting averaged network is similar to that obtained with the 85% threshold from Sachs et al. (2005). Compared to `avg.boot`, only the arc `P38` → `Jnk` is missing, which is an improvement because it was a false positive not present in the validated DAG shown in Figure 6.1.

```
> unlist(compare(cpdag(dag.sachs), cpdag(avg.boot)))
tp fp fn
 1  9 16
> unlist(compare(cpdag(dag.sachs),
+                    cpdag(averaged.network(boot))))
tp fp fn
 0  9 17
```

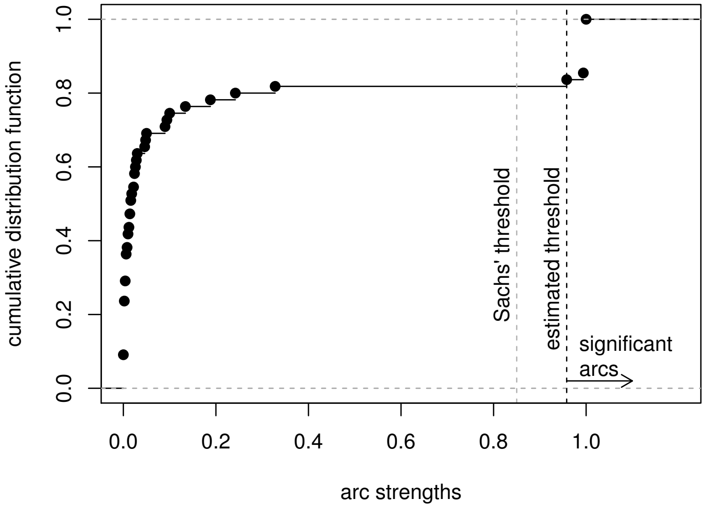


Figure 6.5
Cumulative distribution function of the arc strength values computed with bootstrap resampling from `dsachs`. The vertical dashed lines correspond to the estimated (black) and ad-hoc (grey) significance thresholds.

The reason for the insensitivity of the averaged network to the value of the threshold is apparent from the plot of $F_{\hat{\mathbf{p}}_{(\cdot)}}$ in Figure 6.5: arcs that are well supported by the data are clearly separated from those that are not, with the exception of `P38` $\rightarrow$ `Jnk` (which has strength 0.958, *i.e.*, the estimated threshold). The arc with highest `strength` after `P38` $\rightarrow$ `Jnk` is `Plcg` $\rightarrow$ `Akt` with `strength` 0.328, so any threshold that falls between 0.958 and 0.328 results in the same averaged network.

6.1.5 Handling Interventional Data

Usually, all the observations in a sample are collected under the same general conditions. This is true both for *observational data*, in which treatment allocation is outside the control of the investigator, and for *experimental data*, which are collected from randomised controlled trials. As a result, the sample can be easily modelled with a single BN, because all the observations follow the same probability distribution.

However, this is not the case when several samples resulting from different experiments are analysed together with a single, encompassing model. Such an approach is called *meta analysis* (see Kulinskaya et al., 2008, for a gentle introduction). First, environmental conditions and other exogenous factors may differ between those experiments. Furthermore, the experiments may be different in themselves; for example, they may explore different treatment regimes or target different populations.

This is the case with the protein-signalling data analysed in Sachs et al. (2005). In addition to the data set we have analysed so far, which is subject only to general stimuli meant to activate the desired paths, 9 other data sets subject to different stimulatory cues and inhibitory interventions were used to elucidate the direction of the causal relationships in the network. Such data are often called *interventional*, because the values of specific variables in the model are set by an external intervention of the investigator.

Overall, the 10 data sets contain $5,400$ observations; in addition to the 11 signalling levels analysed above, the protein which is activated or inhibited (`INT`) is recorded for each sample.

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

One intuitive way to model these data sets with a single, encompassing BN is to include the intervention `INT` in the network and to make all variables depend on it. This can be achieved with a `whitelist` containing all possible arcs from `INT` to the other nodes, thus forcing these arcs to be present in the learned DAG.

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

Using tabu search instead of hill-climbing improves the stability of the score-based search; once a locally optimum DAG is found, tabu search performs an additional 50 iterations (as specified by the `tabu` argument) to ensure that no other (and potentially better) local optimum is found.

We can also let the structure learning algorithm decide which arcs connecting `INT` to the other nodes should be included in the DAG. To this end, we can use the `tiers2blacklist` function to blacklist all arcs toward `INT`, thus ensuring that only outgoing arcs may be included in the DAG. In the general case, `tiers2blacklist` builds a blacklist such that all arcs going from a node in a particular element of the `nodes` argument to a node in one of the previous elements are blacklisted.

```
> tiers <- list("INT", names(isachs)[1:11])
> bl <- tiers2blacklist(nodes = tiers)
> dag.tiers <- tabu(isachs, blacklist = bl,
+                score = "bde", iss = 1, tabu = 50)
```

The BNs learned with these two approaches are shown in Figure 6.6. Some of the structural features detected in Sachs et al. (2005) are present in both dag.wh and dag.tiers. For example, the interplay between Plcg, PIP2 and PIP3 and between PKC, P38 and Jnk are both modelled correctly. The lack of any direct intervention on PIP2 is also correctly modelled in dag.tiers. The most noticeable feature missing from both DAGs is the pathway linking Raf to Akt through Mek and Erk.

The approach used in Sachs et al. (2005) yields much better results. Instead of including the interventions in the network as an additional node, they used a modified BDe score (labelled "mbde" in **bnlearn**) incorporating the effects of the interventions into the score components associated with each node (Cooper and Yoo, 1999). When we are controlling the value of a node experimentally, its value is not determined by the other nodes in the BN, but by the experimenter's intervention. Accordingly, mbde disregards the effects of the parents on a controlled node for those observations that are subject to interventions (on that node) while otherwise behaving as the standard bde for other observations.

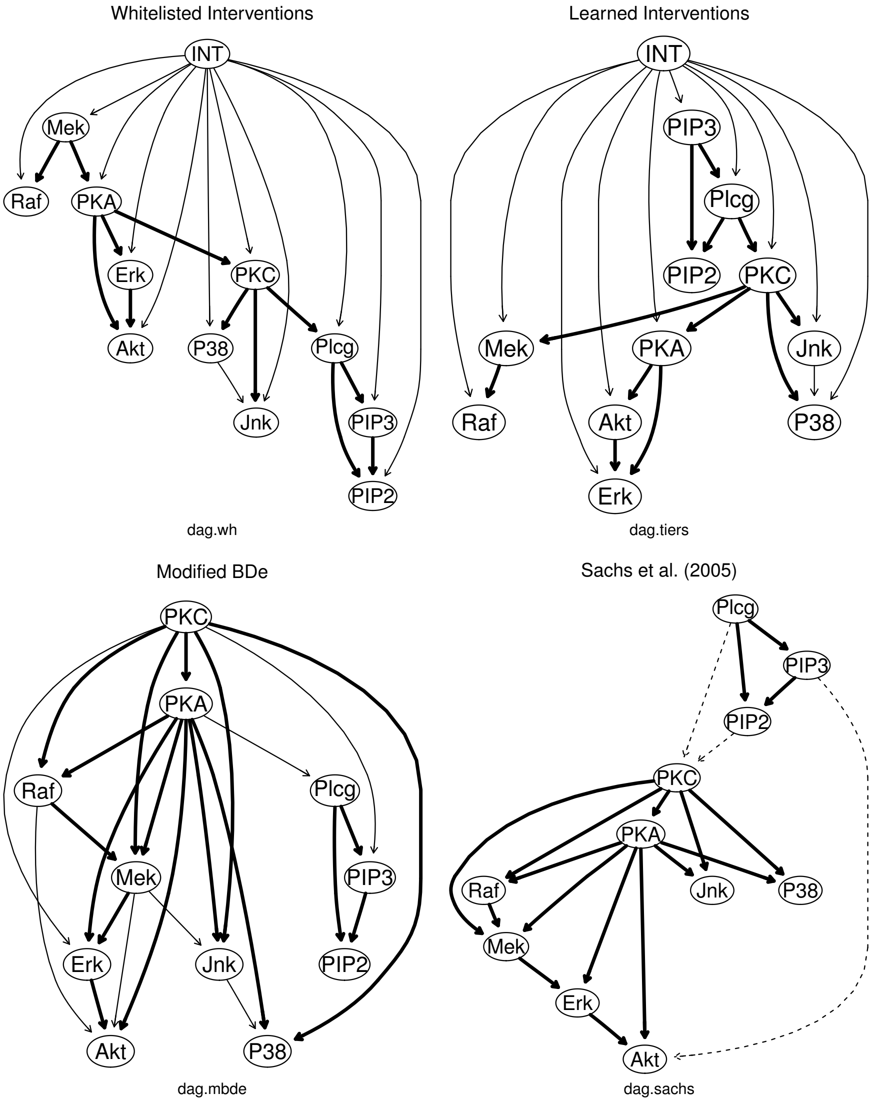
Since the value of INT identifies which node is subject to either a stimulatory cue or an inhibitory intervention for each observation, we can easily construct a named list of which observations are manipulated for each node.

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

We can then pass this list to tabu as an additional argument for mbde. In addition, we can combine the use of mbde with model averaging and random starting points as discussed in Section 6.1.3. To improve the stability of the averaged network, we generate the set of the starting networks for the tabu searches using the algorithm from Melançon et al. (2001), which is not limited to connected networks as is that from Ide and Cozman (2002). In addition, we actually use only one generated network every 100 to obtain a more diverse set.

```
> 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 = INT,
+       iss = 1, start = net, tabu = 50)
+ })
> intscore <- custom.strength(netlist, nodes = nodes,
+                 cpdag = FALSE)
```

Note that we have set cpdag = FALSE in custom.strength, so that the DAGs being averaged are not transformed into CPDAGs before computing arc strengths and direction probabilities. The reason is that, unlike the BDe



Whitelisted Interventions

dag.wh

Learned Interventions

dag.tiers

Modified BDe

dag.mbde

Sachs et al. (2005)

dag.sachs

Figure 6.6
DAGs learned from `isachs`. The first two networks (`dag.wh` on the top left, `dag.tiers` on the top right) have been learned by including `INT` and adding arcs to model stimulatory cues and inhibitory interventions. The third BN (`dag.mbde`, on the bottom left) has been learned with model averaging and the `mbde` score; arcs plotted with a thicker line width make up the validated BN (bottom right) from Sachs et al. (2005).

score we used in Section 6.1.3, `mbde` is not score equivalent as a consequence of incorporating the interventions. In fact, information about the interventions is a form of prior information, and makes it possible to identify arc directions even when the arc would be score equivalent from the data alone.

Averaging the DAGs with `averaged.network` and the threshold from Section 6.1.4, we are finally able to correctly identify all the arcs in the network from Sachs et al. (2005).

```
> dag.mbde <- averaged.network(intscore)
> unlist(compare(dag.sachs, dag.mbde))
tp fp fn
17  7  0
```

As we can see from Figure 6.6, `dag.mbde` is much closer to the validated network from Sachs et al. (2005) than any of the other BNs learned in this section. All the arcs from the validated network are correctly learned; in the output above we have 17 true positives and 0 false negatives, and the original network contains 17 arcs. The three arcs that were missing in Sachs et al. (2005) are missing in `dag.mbde` as well. The arcs from `dag.mbde` that are not present in the validated network were identified in Sachs et al. (2005) and discarded due to their comparatively low strength; this may imply that the simulated annealing algorithm used in Sachs et al. (2005) performs better on this data set than tabu search.

6.1.6 Querying the Network

In their paper, Sachs et al. (2005) used the validated BN to substantiate two claims:

1. a direct perturbation of `Erk` should influence `Akt`;
2. a direct perturbation of `Erk` should not influence `PKA`.

The probability distributions of `Erk`, `Akt` and `PKA` were then compared with the results of two ad-hoc experiments to confirm the validity and the direction of the inferred causal influences.

Given the size of the BN, we can perform the queries corresponding to the two claims above using any of the exact and approximate inference algorithms introduced in Section 4.6.2. First, we need to create a `bn.fit` object for the validated network structure from Sachs et al. (2005).

```
> isachs <- isachs[, 1:11]
> for (i in names(isachs))
+   levels(isachs[, i]) = c("LOW", "AVG", "HIGH")
> fitted <- bn.fit(dag.sachs, isachs, method = "bayes")
```

The `INT` variable, which codifies the intervention applied to each observation, is not needed for inference and is therefore dropped from the data set. Furthermore, we rename the expression levels of each protein to make both the subsequent R code and its output more readable.

Subsequently, we can perform the two queries using the junction tree algorithm provided by the **gRain** package, which we have previously explored in Section 1.6.2.1. The results are shown in Figure 6.7.

```
> library(gRain)
> jtree <- compile(as.grain(fitted))
```

We can introduce the direct perturbation of `Erk` required by both queries by calling `setEvidence` as follows. In causal terms, this would be an ideal intervention.

```
> jlow <- setEvidence(jtree, nodes = "Erk", states  = "LOW")
```

As we can see from the code below, the marginal distribution of `Akt` changes depending on whether we take the evidence (intervention) into account or not.

```
> querygrain(jtree, nodes = "Akt")$Akt
Akt
   LOW    AVG   HIGH
0.6089 0.3104 0.0807
> querygrain(jlow, nodes = "Akt")$Akt
Akt
   LOW    AVG   HIGH
0.6652 0.3333 0.0015
```

The slight inhibition of `Akt` induced by the inhibition of `Erk` agrees with both the direction of the arc linking the two nodes and the additional experiments performed by Sachs et al. (2005). In causal terms, the fact that changes in `Erk` affect `Akt` supports the existence of a causal link from the former to the latter.

As far as `PKA` is concerned, both the validated network and the additional experimental evidence support the existence of a causal link from `PKA` to `Erk`. Therefore, interventions to `Erk` cannot affect `PKA`. In a causal setting, interventions on `Erk` would block all biological influences from other proteins, which amounts to removing all the parents of `Erk` from the DAG.

```
> causal.sachs <- drop.arc(dag.sachs, "PKA", "Erk")
> causal.sachs <- drop.arc(causal.sachs, "Mek", "Erk")
> cfitted <- bn.fit(causal.sachs, isachs, method = "bayes")
> cjtree <- compile(as.grain(cfitted))
> cjlow <- setEvidence(cjtree, nodes = "Erk", states  = "LOW")
```

After building the junction tree for this new DAG, called `causal.sachs`, we can perform the same query on both `cjtree` and `cjlow` as we did above.

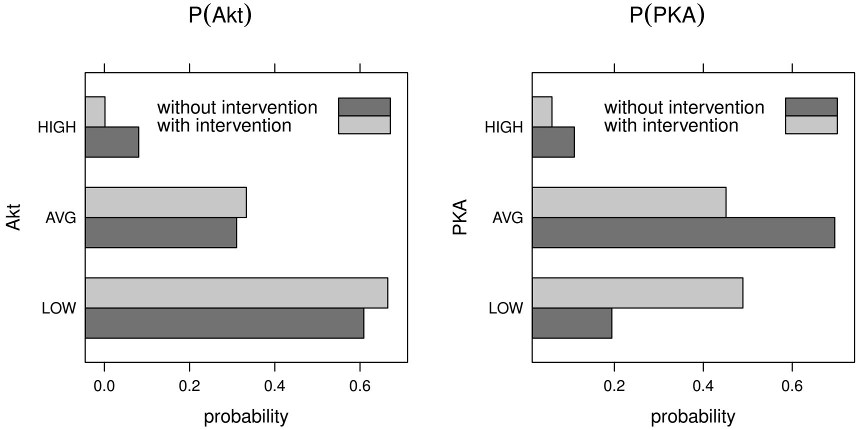


Figure 6.7
Probability distributions of `Akt` and `PKA` before and after inhibiting `Erk`, considered in a non-causal setting.

```
> querygrain(cjtree, nodes = "PKA")$PKA
PKA
  LOW   AVG  HIGH
0.194 0.696 0.110
> querygrain(cjlow, nodes = "PKA")$PKA
PKA
  LOW   AVG  HIGH
0.194 0.696 0.110
```

Indeed, `PKA` has exactly the same distribution in both cases. However, knowledge of the expression level of `Erk` may still alter our expectations on `PKA` if we treat it as evidence instead of an ideal intervention. In practice this implies the use of the original junction trees `jtree` and `jlow`, as opposed to the modified `cjtree` and `cjlow` we used for the previous query.

```
> querygrain(jtree, nodes = "PKA")$PKA
PKA
  LOW   AVG  HIGH
0.194 0.696 0.110
> querygrain(jlow, nodes = "PKA")$PKA
PKA
   LOW    AVG   HIGH
0.4891 0.4512 0.0597
```

All the queries illustrated above can be easily changed to maximum a posteriori queries by finding the largest element (with `which.max`) in the distribution of the target node.

```
> names(which.max(querygrain(jlow, nodes = c("PKA"))$PKA))
[1] "LOW"
```

Clearly, such a simple approach is possible because of the nature of the evidence and the small number of nodes we are exploring. When many nodes are explored simultaneously, inference on their joint conditional distribution quickly becomes very difficult and computationally expensive. In these high-dimensional settings, algorithms specifically designed for MAP queries and ad-hoc approaches are preferable.

6.2 Predicting the Body Composition[1]

The *human body composition* is the distribution of the three components that form body weight: *bone*, *fat* and *lean*. They identify, respectively, the mineral content, the fat and the remaining mass of the body (mainly muscles). In a detailed analysis, body composition is measured separately across *trunk*, *legs*, and *arms*, and it is an important diagnostic tool since ratios of these masses can reveal regional physiological disorders. One of the most common ways to measure these masses is the dual-energy x-ray absorptiometry (DXA; Centers for Disease Control and Prevention, 2010), which unfortunately is time-consuming and very expensive. Therefore, there is an interest in alternative protocols that can be used to the same effect.

In the following we will try, in a very simple way, to predict body composition from related quantities that are much cheaper and easier to measure: *age*, *height*, *weight* and *waist circumference*. To that end, we will use a sample of 100 white men collected from the NHANES project (Centers for Disease Control and Prevention, 2004) which includes simultaneous measurements for the variables above. This data set is available in the **rbmn** package under the name `boco`.

```
> library(rbmn)
> data(boco)
> round(head(boco), 1)
   A   H     W   C   TF   LF  AF   TL   LL   AL  TB  LB  AB
1 83 182  92.6 117 17.1  8.9 3.0 31.2 18.5  6.6 0.6 1.1 0.5
2 68 169  74.7  93  8.3  5.4 2.0 28.0 16.2  7.5 0.7 1.0 0.5
```

[1] This section was written with the help of Simiao Tian (MIAJ, INRA, 78352 Jouy-en-Josas, France).

label	definition	unit
TB	Trunk Bone	kg
TF	Trunk Fat	kg
TL	Trunk Lean	kg
LB	Leg Bone	kg
LF	Leg Fat	kg
LL	Leg Lean	kg
AB	Arm Bone	kg
AF	Arm Fat	kg
AL	Arm Lean	kg
A	Age	year
W	Weight	kg
H	Height	cm
C	Waist Circumference	cm
B	Body Mass Index	kg/m^2

Table 6.1
The nine variables we are interested in predicting along with the possible covariates, their labels and their units of measurement.

```
3 28 182 112.2 112 17.7 11.3 3.1 36.7 24.5 10.1 0.8 1.1 0.5
4 41 171  82.6  96 10.6  6.5 1.8 29.2 19.6  7.8 0.8 1.1 0.5
5 85 169  71.1 102 10.9  4.7 1.9 26.2 14.5  5.8 0.6 1.1 0.4
6 29 176  88.4  96 11.2  7.5 2.7 31.4 19.8  8.3 0.7 1.0 0.4
> boco$B <- boco$W / boco$H^2 * 10^4
> dim(boco)
[1] 100  14
> n <- nrow(boco)
> vr <- colnames(boco)[5:13]
> co <- c("A", "H", "W", "C", "B")
```

First, we computed the body mass index (B) for each individual and added it to boco. It is a very popular score, normalising the weight by the height; intuitively, a weight of 100kg has very different health implications for a person 160cm tall than for another person 185cm tall. Therefore, we now have a set of 5 covariates and 9 variables of interest for a sample size of $n = 100$. Their labels and definitions are given in Table 6.1.

6.2.1 Aim of the Study

A standard statistical approach used for prediction is the multivariate multiple linear regression. Denoting the (response) variables of interest by $\mathbf{Y}$ and the covariates (the explanatory variables) by $\mathbf{X}$, the model has the following form:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\Theta} + \mathbf{E} \tag{6.6}$$

$$V\left(vec\left(\mathbf{E}\right)\right) = \mathbf{I}_n \otimes \boldsymbol{\Sigma} \tag{6.7}$$

where:

- $\mathbf{Y}$ is an $n \times p$ matrix;
- $\mathbf{X}$ is an $n \times (q+1)$ matrix (including a $\mathbf{1}$ column for the intercept as well as the covariates);
- $\boldsymbol{\Theta}$ is a $(q+1) \times p$ matrix for the regression coefficients;
- $\mathbf{E}$ is an $n \times p$ matrix for the error terms;
- $\boldsymbol{\Sigma}$ is a $p \times p$ covariance matrix for the error terms;
- n is the number of observations;
- p is the number of variables; and
- q is the number of covariates.

The number of parameters is $p(q+1)$ for the regression coefficients and $p(p+1)/2$ for the covariance matrix of the error terms. When p and q are large, n must be larger still to provide enough predictive power. Of course, it is well known that covariances are more difficult to estimate from the data than mean values. Using BNs can be an effective way to dramatically reduce the number of parameters in the model and thus the required sample size. More precisely, a GBN defined over p variables and q covariates, with n_c arcs from the covariates to the variables and n_v arcs between the variables, will have only $2p + n_c + n_v$ parameters, and a formulation that is consistent with Equations (6.6) and (6.7).

Indeed, Equation (6.7) can be interpreted as a saturated GBN model (*e.g.*, all possible arcs are present in the DAG) in which the roles of variables and covariates are emphasised. Therefore, we can reasonably expect better predictions from a sparse GBN when the sample size is too small for a standard multivariate multiple regression approach.

With the aim of predicting the nine variables with the five covariates in Table 6.1, we will show an example of how we can learn such a GBN. It could be argued that `W` and `H` make `B` redundant, but their relationship is not linear and therefore they convey different information in a linear model.

6.2.2 Designing the Predictive Approach

6.2.2.1 Assessing the Quality of a Predictor

First we will split the data into a training set (`dtr`) and a validation set (`dva`). The training set will be used to learn or estimate the models; the validation set will be used to select those models that perform the prediction of new individuals well, thus reducing the risk of overfitting (Hastie et al., 2009). Here we will use a randomly generated split into sets of equal size, each comprising 50 individuals.

```
> str <- sort(sample(n, round(n/2)))
> dtr <- boco[str, ]
> dva <- boco[-str, ]
```

We also have to deal with the bias-variance trade-off of a prediction. The discrepancy between the observed value and the predicted mean will capture the bias; the standard deviation will be given by that of the predictor. For each variable, the *standard error of prediction* (SEP) will be obtained by the classic formula

$$\text{SEP} = \sqrt{\text{bias}^2 + \text{standard deviation}^2}. \tag{6.8}$$

Another important consideration when choosing a model for prediction is its interpretability by experts in the field. When their statistical properties are comparable, models which elucidate the phenomenon under investigation are usually preferable to empirical, black-box ones. In the context of BNs, models with not too many arcs, and arcs with a physiological interpretation for body composition will be more appealing than the saturated model in Equations (6.6) and (6.7). In this respect, we hope that complex relationships between the variables will disappear after the incorporation of good covariates. In other words: complex marginal dependencies but simple conditional dependencies, and in the best case conditional independence.

6.2.2.2 The Saturated BN

In order to have a term of comparison for the models we will fit in the following, we now consider the multiple regression equivalent of a saturated BN (with all possible arcs between variables and/or covariates).

```
> satua <- lm(cbind(TF, LF, AF, TL, LL, AL, TB, LB, AB) ~
+                 A + H + W + C + B, data = dtr)
> r.dof <- anova(satua)["Residuals", "Df"]
> satup <- predict(satua, newdata = dva)
> satabias <- abs(dva[, vr] - satup)
> satstdev <- outer(rep(1, nrow(dtr)),
+                 sqrt(colSums(residuals(satua)^2)/r.dof), "*")
> satsep <- sqrt(satabias^2 + satstdev^2)
> satgsco <- cbind(colMeans(satabias), colMeans(satstdev),
+                   colMeans(satsep))
> colnames(satgsco) <- c("|Bias|", "Sd.Dev", "SEP")
> round(satgsco, 2)
   |Bias| Sd.Dev  SEP
TF   1.34   1.67 2.29
LF   1.17   1.56 2.04
AF   0.41   0.42 0.63
TL   1.28   1.60 2.19
LL   0.98   1.30 1.73
AL   0.54   0.70 0.94
TB   0.09   0.10 0.14
LB   0.11   0.12 0.17
AB   0.05   0.05 0.07
> satsupe <- colSums(satgsco)
> round(satsupe, 2)
|Bias| Sd.Dev    SEP
  5.97   7.51  10.19
```

The output above shows a cumulated SEP of 10.19 over the 9 variables, with a bias of 5.97 and a standard deviation of 7.51. These three values do not satisfy Equation (6.8), because we are summing SEPs for all predicted individuals and variables, and the definition of SEP is not additive. The saturated GBN has 36 arcs between the variables and 45 arcs from the covariates to the variables.

6.2.2.3 Convenient BNs

The training set `dtr` can also be used to learn a GBN with one of the algorithms available in **bnlearn**, as below.

```
> library(bnlearn)
> dag1 <- hc(dtr)
> paste(substr(modelstring(dag1), 1, 40), "...", sep = "")
[1] "[A][TB][LB|TB][AB|TB:LB][H|AB][W|AB][B|H..."
```

However, this first attempt at learning a GBN does not produce a DAG that is convenient to use for prediction. We would prefer to have covariates as ancestors of the response variables, to be able to immediately find their conditional probability distribution. This is not the case here since the variable `AB` is the parent of the covariate `H`. As we have seen in Chapter 4, we can certainly use conditioning other than that implied by the topological order; the simple structure of the BN would still help in prediction. But this would require complex computations and we will not proceed in this way, preferring BNs where all covariates are ancestors of all the variables we are trying to predict.

Blacklists and whitelists provide a simple way to enforce these constraints on the network structure when learning the GBN. The strongest constraint is to force every covariate to be the parent of every response variable; that means to include 5×9 arcs in the DAG.

```
> wl1 <- cbind(from = rep(co, each = 9), to = rep(vr, 5))
> dag2 <- hc(dtr, whitelist = wl1)
> paste(substr(modelstring(dag2), 1, 40), "...", sep = "")
[1] "[A][H][W|H][B|H:W][C|A:H:B][LB|A:H:W:C:B..."
```

As expected, the covariates are in the first positions in the topological ordering of the DAG. To relax the constraints on the search, we can switch from the whitelist `wl1` to a blacklist preventing only unwanted arcs. In other words, variables cannot be the parents of covariates but covariates are not necessarily the parents of all the variables.

```
> bl1 <- wl1[, 2:1]
> dag3 <- hc(dtr, blacklist = bl1)
> paste(substr(modelstring(dag3), 1, 40), "...", sep = "")
[1] "[A][H][W|H][B|H:W][C|A:H:B][TF|C][AF|TF:..."
> all.equal(dag2, dag3)
[1] "Different number of directed/undirected arcs"
```

The resulting DAG is similar for the covariates but different for the response variables. In fact, every combination of the two lists is possible, and any overlap between the whitelist and the blacklist is handled properly by `hc`.

```
> iwl <- 1:15
> wl2 <- wl1[iwl, ]
> bl2 <- bl1[-iwl, ]
> dag4 <- hc(dtr, whitelist = wl2, blacklist = bl2)
> paste(substr(modelstring(dag4), 1, 40), "...", sep = "")
[1] "[A][H][W|H][B|H:W][C|A:H:B][TF|A:H:C][TL..."
```

It is important to note that we are not much interested in the distribution of the covariates since they will be assumed known for the purposes of prediction. Clearly, interactions between covariates should be allowed to some extent for the GBN to be correctly specified and to limit the bias in the regression coefficients associated with the covariates in the response variables' conditional distributions. However, such interactions do not directly affect predictions. We are just interested in an efficient and simple form of the conditional distribution

$$Y_1, Y_2, \ldots, Y_p \mid X_1, X_2, \ldots, X_q.$$

6.2.3 Looking for Candidate BNs

With these considerations in mind, we can implement a search strategy for good predictive BNs. We will use the SEP score from Equation (6.8) and its two components (bias and standard deviation) as measures of the BN's performance on the test set `dva`. As a first step, we estimate the parameters of the model from the training set `dtr` using the `dag2` object we created in the previous section.

```
> bn2 <- bn.fit(dag2, data = dtr)
```

Then we obtain the conditional distribution for each individual in the test set with **rbmn**, and we use them to compute bias and standard deviation. We store them in two data frames, `bias` and `stde`.

```
> library(rbmn)
> mn2 <- gema2mn(nbn2gema(bnfit2nbn(bn2)))
> bias <- stde <- dva[, vr]
> for (ind in 1:nrow(dva)) {
+   mni <- condi4joint(mn2, par = vr, pour = co,
+                          unlist(dva[ind, co]))
+   bias[ind, vr] <- dva[ind, vr] - mni$mu[vr]
+   stde[ind, vr] <- sqrt(diag(mni$gamma)[vr])
+ }#FOR
> sep <- sqrt(bias^2 + stde^2)
```

A global score can be obtained summing over all the observations in the validation data set.

```
> gscores <- cbind(colMeans(abs(bias)), colMeans(stde),
+                    colMeans(sep))
> colnames(gscores) <- c("|Bias|", "Sd.Dev", "SEP")
> round(gscores, 2)
   |Bias| Sd.Dev  SEP
TF   1.34   1.70 2.32
LF   1.17   1.57 2.06
AF   0.41   0.42 0.63
TL   1.28   1.83 2.37
LL   0.98   1.34 1.77
AL   0.54   0.79 1.00
TB   0.09   0.10 0.14
LB   0.11   0.12 0.17
AB   0.05   0.05 0.07
> superf <- colSums(gscores)
> round(superf, 2)
|Bias| Sd.Dev    SEP
  5.97   7.92  10.52
```

These results are encouraging compared to those arising from the saturated GBN $(5.97, 7.51, 10.19)$ if we consider we are using far fewer parameters (84 instead of 105). But as previously mentioned, it is also important to check whether `dag2` is easy to interpret. For this purpose, we can plot `dag2` with a convenient layout for the nodes corresponding to the 3×3 variables and the 5 covariates.

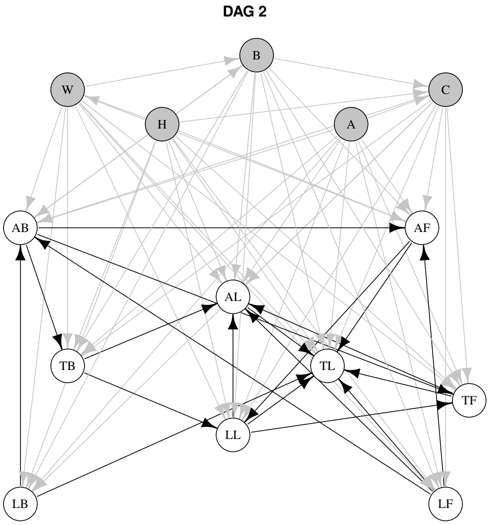


Figure 6.8
Network structure of dag2. Only the arcs we are interested in, those between two variables, are drawn in black; all the others are shown in grey. Nodes corresponding to covariates have a grey background.

```
> library(graph)
> library(igraph)
> load("bc.poco.rda")
> cbind(posi, colo)[c(1:3, 10:11), ]
   x    y    colo
TF "95" "25" "white"
LF "90" "10" "white"
AF "85" "50" "white"
A  "70" "65" "lightgrey"
H  "30" "65" "lightgrey"
> idag2 <- igraph.from.graphNEL(as.graphNEL(dag2))
> nad <- V(idag2)$label <- V(idag2)$name
> edcol <- rep("lightgrey", nrow(arcs(dag2)))
> aa <- which((arcs(dag2)[, 1] %in% vr) &
+              (arcs(dag2)[, 2] %in% vr))
> va <- as.numeric(E(idag2, P = t(arcs(dag2)[aa, ])))
> edcol[va] <- "black"
> plot(idag2, layout = posi[nad, ], main = "DAG 2",
+        edge.color = edcol, vertex.color = colo[nad])
```

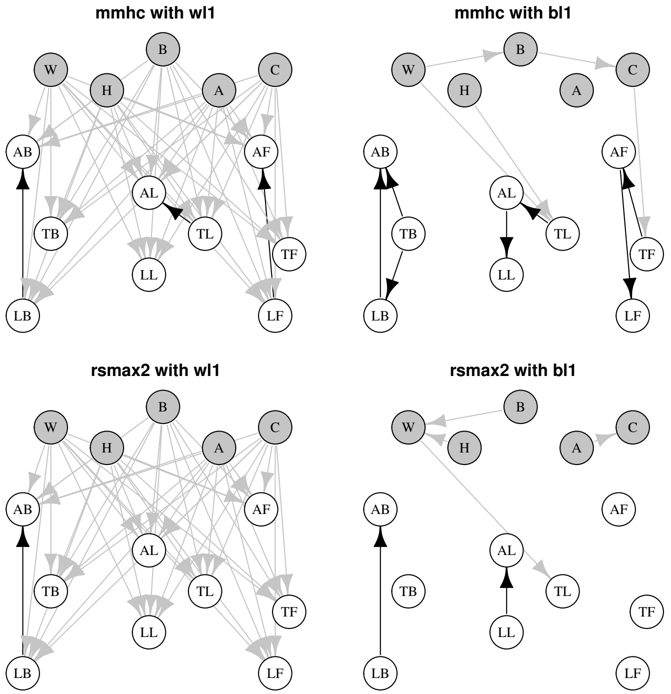


Figure 6.9
DAGs found when combining either `mmhc` (top left and top right) or `rsmax2` (bottom left and bottom right) of **bnlearn** with either the whitelist `wl1` (top left and bottom left) or the blacklist `bl1` (top right and bottom right). The two functions are called with their default options.

The plot produced by the code above is shown in Figure 6.8. No systematic structure is apparent, and a sparser DAG with a comparable SEP score would surely be preferable. Unfortunately, using the blacklist `bl1` instead of the whitelist `wl1` does not improve things significantly. On the other hand, changing structure learning from a score-based to a hybrid algorithm has a dramatic effect, as can be seen in Figure 6.9.

First of all, there are fewer arcs between the variables. In addition, the two-way structure of the regions (trunk, legs, arms) × (lean, fat, bone) is clearly recognisable. The GBN learned with `rsmax2` and the whitelist looks quite promising since only one arc is left between the variables, and it is linking the less physiologically interesting ones (bone mass is not as varied as fat mass). Nevertheless all variables depend on all covariates due to the whitelist, which is not so desirable when aiming at a parsimonious model.

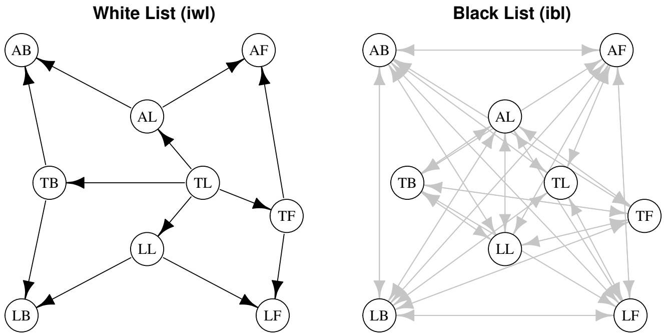


Figure 6.10
Arcs included in the whitelist and in the blacklist used to look for a minimal subset of covariates. The whitelist (iwl, in black, on the left) includes arcs that are interpretable under the current set of assumptions. The blacklist (ibl, in grey, on the right) includes all arcs between two variables which are not in the skeleton of the whitelist.

Therefore, we wonder whether we need to use all the covariates to obtain good predictions. To investigate this further we introduce some restrictions on the arcs linking the variables, either with a whitelist (to see if more arcs are necessary) or with a blacklist (to prevent some arc from appearing); and we still forbid arcs from a variable to a covariate. As the underlying theory, we assume that conditionally to the covariates:

- The lean and the fat remain correlated (arcs between TL, LL, AL and TF, LF, AF).
- The lean and the bone remain correlated (arcs between TL, LL, AL and TB, LB, AB).
- Fat and bone are conditionally independent from the lean (no arcs between TF, LF, AF and TB, LB, AB).
- The trunk weight influences the leg weight since legs carry the body of which the trunk is the most important part (arcs between TB, TF, TL and LB, LF, LL).
- Other appendixes are also related to the trunk (arcs between TB, TF, TL and AB, AF, AL).
- There is no direct arc between LB, LF, LL and AB, AF, AL.

The arcs corresponding to these assumptions and included in the whitelist iwl or in the blacklist ibl are shown in Figure 6.10. The whitelist forces all the arcs with a clear interpretation to be present in the DAG, while allowing

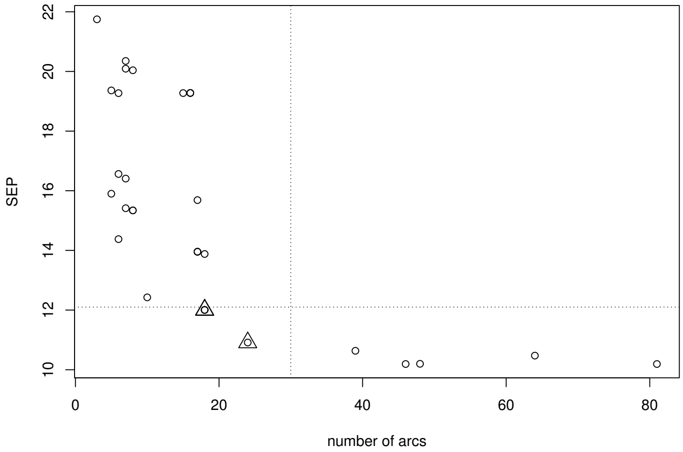


Figure 6.11
Each of the 32 models in Table 6.2 is displayed using the total number of arcs excluding those linking two covariates and the SEP score. The four models selected for a small SEP and a small number of arcs are surrounded with a triangle (in fact three are superimposed).

additional arcs. The blacklist prevents all the arcs that do not have a clear interpretation from being included in the DAG, regardless of their directions, and is designed to complement the whitelist.

Table 6.2 summarises the BNs we created previously and those resulting from a systematic search with 6 subsets of covariates (including the complete set) and with two algorithms (`mmhc` and `rsmax2`, called with their default arguments), using either the whitelist or the blacklist above. `rsmax2` is implemented using Grow-Shrink (`gs`) in the *restrict* phase and hill-climbing (`hc`) in the *maximise phase*. The quality of each of these BNs is assessed with the three scores and the number of arcs either between a covariate and a variable or between two variables.

The correlation between the bias and the standard deviation over all the BNs, with the exception of the saturated one, is very high (0.993) and suggests that both criteria are equivalent to each other and to the global SEP.

To get a synthetic view of the quality of the different models, we plotted their SEP scores against the total number of arcs in Figure 6.11. Four of them are denoted with triangles and appear particularly interesting; they are shown in bold in Table 6.2. Three of them have the same SEP. Perhaps a sensible choice is to retain the GBN with the smallest number of covariates, *i.e.*, `<10>`, displayed in Figure 6.12. The saturated BN provides a good predictive power

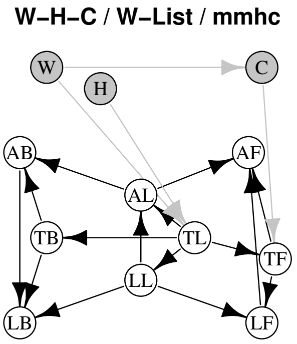


Figure 6.12
DAG of the chosen BN, labelled <10> in Table 6.2. Only three covariates are included, with three arcs pointing towards the variables.

but it is the worst for interpretation, and it is unsurprising to note that it was outperformed by three BNs for the SEP criterion.

The interpretation of the chosen BN is straightforward. Height and weight have a direct influence on the lean mass of the trunk (TL), and in turn that influences the two other lean segments (LL and AL) which are all adjacent to each other. The same is true for the other two components: TF, LF and AF; and TB, LB and AB. Component-regions are directly linked within the component, the trunk being the pivot. Finally, an important correction is provided by the waist circumference (C) on the fat of the trunk (TF), which has a knock-on effect on the two other fat nodes.

The proportion of variance explained for each variable by its parents provides a local quantitative assessment of the BN's quality. To compute it we can use the sum of squares of the corresponding analysis of variance, for instance on the variable TL:

```
> av.tl <- anova(lm(TL ~ H + W, data = dva))
> 1 - av.tl["Residuals", "Sum Sq"] / sum(av.tl[, "Sum Sq"])
[1] 0.904
```

Table 6.3 reports these proportions for all variables. They are quite good, except for TB (0.59) and LF (0.69). Note however that there is some arbitrariness in the choice of the parents because different equivalent DAGs could have been used; we rely on an expert's choice. Furthermore, it is worth underlining that these proportions do not measure the quality of predictions, they only assess the strength of the arcs included in the BN. They coincide only when all parents of the node are covariates, as is the case for TL.

	Algo.	Cova.	BL	WL	|Bias|	SD	SEP	c2v	v2v
dag2	hc	A-H-W-C-B	-	wl1	6.0	7.9	10.5	45	18
dag3	**hc**	**A-H-W-C-B**	**bl1**	-	**6.1**	**8.3**	**10.9**	**8**	16
dag4	hc	A-H-W-C-B	bl2	wl2	5.9	8.0	10.5	20	19
(1)	mmhc	A-H-W-C-B	-	wl1	6.0	7.5	10.2	45	3
(2)	mmhc	A-H-W-C-B	bl1	-	7.0	9.4	12.4	3	7
(3)	rsmax2	A-H-W-C-B	-	wl1	6.0	7.5	10.2	45	1
(4)	rsmax2	A-H-W-C-B	bl1	-	13.2	15.7	21.7	1	2
<1>	mmhc	W-H-B-A-C	ibl+rbl	-	9.2	11.2	15.4	3	4
<2>	**mmhc**	**W-H-B-A-C**	**rbl**	**iwl**	**6.8**	**9.1**	**12.0**	**3**	**15**
<3>	rsmax2	W-H-B-A-C	ibl+rbl	-	9.9	11.9	16.4	3	4
<4>	rsmax2	W-H-B-A-C	rbl	iwl	8.1	10.3	14.0	3	14
<5>	mmhc	W-H-A-C	ibl+rbl	-	9.2	11.1	15.3	4	4
<6>	**mmhc**	**W-H-A-C**	**rbl**	**iwl**	**6.8**	**9.1**	**12.0**	**3**	**15**
<7>	rsmax2	W-H-A-C	ibl+rbl	-	8.7	10.3	14.4	3	3
<8>	rsmax2	W-H-A-C	rbl	iwl	8.1	10.3	14.0	3	14
<9>	mmhc	W-H-C	ibl+rbl	-	9.2	11.1	15.3	4	4
<10>*	**mmhc**	**W-H-C**	**rbl**	**iwl**	**6.8**	**9.1**	**12.0**	**3**	**15**
<11>	rsmax2	W-H-C	ibl+rbl	-	9.6	11.5	15.9	2	3
<12>	rsmax2	W-H-C	rbl	iwl	8.1	10.3	14.0	3	14
<13>	mmhc	B-A-C	ibl+rbl	-	12.5	14.2	20.0	3	5
<14>	mmhc	B-A-C	rbl	iwl	12.0	13.6	19.3	1	15
<15>	rsmax2	B-A-C	ibl+rbl	-	10.0	11.9	16.6	2	4
<16>	rsmax2	B-A-C	rbl	iwl	12.0	13.6	19.3	1	14
<17>	mmhc	B-C	ibl+rbl	-	12.7	14.4	20.4	2	5
<18>	mmhc	B-C	rbl	iwl	12.0	13.6	19.3	1	15
<19>	rsmax2	B-C	ibl+rbl	-	11.7	13.9	19.3	2	4
<20>	rsmax2	B-C	rbl	iwl	12.0	13.6	19.3	1	15
<21>	mmhc	W-H-A	ibl+rbl	-	10.2	12.5	17.1	3	4
<22>	mmhc	W-H-A	rbl	iwl	8.2	12.4	15.7	2	15
<23>	rsmax2	W-H-A	ibl+rbl	-	10.2	13.4	17.8	3	3
<24>	rsmax2	W-H-A	rbl	iwl	8.1	10.3	13.9	4	14
saturated	-	A-H-W-C-B	-	-	6.0	7.5	10.2	45	36

Table 6.2
The 32 BNs we explored, including the learning algorithm (Algo.), the set of covariates (Cova.), the blacklist if any (BL), the whitelist if any (WL). Their performance has been assessed with the bias (|Bias|), the standard deviation (SD), the SEP score, the number of arcs going from a covariate to a response variable (c2v) and the number of arcs going from a response variable to another response variable (v2v). “rbl” is the list of arcs from a variable to a covariate. The four selected models are shown in bold and the model chosen for interpretation is marked with a “*”.

	Parent(s)	Variance Explained
TF	TL + C	0.88
LF	TF + LL	0.69
AF	TF + LF + AL	0.85
TL	W + H	0.90
LL	TL	0.84
AL	TL + LL	0.81
TB	TL	0.59
LB	LL + TB + AB	0.88
AB	AL + TB	0.74

Table 6.3
The proportion of variance explained for each of the nine response variables by the respective parents within the chosen BN.

6.3 Further Reading

There are few books in literature that describe real-world applications of BNs, as we did in this chapter. For the interested reader, we suggest Pourret et al. (2008) and Nagarajan et al. (2013); Chapters 5 and 11 in Korb and Nicholson (2004); and Chapters 6, 7, 8 in Holmes and Jaim (2008). The data analysed therein span several different fields, including medical diagnosis, genetics, education, forensic, finance and industrial systems.

A

Graph Theory

A.1 Graphs, Nodes and Arcs

A graph $G = (\mathbf{V}, A)$ consists of a non-empty set $\mathbf{V}$ of *nodes* or *vertices* and a finite (but possibly empty) set A of pairs of vertices called *arcs*, *links* or *edges*.

Each arc $a = (u, v)$ can be defined either as an ordered or an unordered pair of nodes, which are said to be *connected* by and *incident* on the arc and to be *adjacent* to each other. Here we will restrict ourselves to graphs having zero or one connection between any pair of nodes. Since they are adjacent, u and v are also said to be *neighbours*. If (u, v) is an ordered pair, u is said to be the *tail* of the arc and v the *head*; then the arc is said to be *directed* from u to v, and is usually represented with an arrowhead in v $(u \rightarrow v)$. It is also said that the arc *leaves* or is *outgoing* for u, and that it *enters* or is *incoming* for v. If (u, v) is unordered, u and v are simply said to be incident on the arc without any further distinction. In this case, they are commonly referred to as *undirected arcs* or *edges*, denoted with $e \in E$ and represented with a simple line $(u - v)$.

The characterisation of arcs as directed or undirected induces an equivalent characterisation of the graphs themselves, which are said to be *directed graphs* (denoted with $G = (\mathbf{V}, A)$) if all arcs are directed, *undirected graphs* (denoted with $G = (\mathbf{V}, E)$) if all arcs are undirected and *partially directed* or *mixed graphs* (denoted with $G = (\mathbf{V}, A, E)$) if they contain both arcs and edges.

An example of each class of graphs is shown in Figure A.1. The first graph (on the left) is an undirected graph, in which:

- the node set is $\mathbf{V} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$ and the edge set is $E = \{$ (A − B), (A − C), (A − D), (B − D), (C − E), (D − E) $\}$;
- arcs are undirected, so *i.e.*, A − B and B − A are equivalent and identify the same edge;
- likewise, A is connected to B, B is connected to A, and A and B are adjacent.

The second graph (centre) is a directed graph. Unlike the undirected graph we just considered:

- the directed graph is characterised by the arc set $A = \{$(A → B), (C → A),

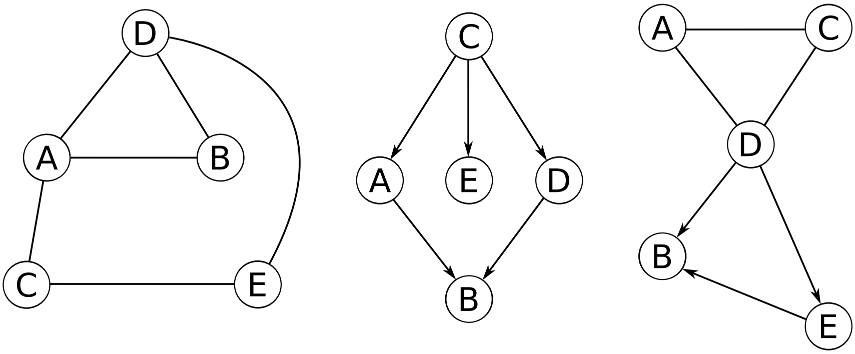


Figure A.1
An undirected graph (left), a directed graph (centre) and a partially directed graph (right).

(D → B), (C → D), (C → E)} instead of an edge set E, even though the node set $\mathbf{V}$ is the same as before;

- arcs are directed, so *i.e.*, A → B and B → A identify different arcs; A → B $\in A$ while B → A $\notin A$. Furthermore, it is not possible for both arcs to be present in the graph because there can be at most one arc between each pair of nodes;

- again, A and B are adjacent, as there is an arc (A → B) from A to B. A → B is an outgoing arc for A (the tail), an incoming arc for B (the head) and an incident arc for both A and B.

The third graph (on the right) is a mixed graph, which is characterised by the combination of an edge set $E = \{(\text{A} - \text{C}), (\text{A} - \text{D}), (\text{C} - \text{D})\}$ and an arc set $A = \{(\text{D} \to \text{E}), (\text{D} \to \text{B}), (\text{E} \to \text{B})\}$.

An undirected graph can always be constructed from a directed or partially directed one by substituting all the directed arcs with undirected ones; such a graph is called the *skeleton* or the *underlying undirected graph* of the original graph.

A.2 The Structure of a Graph

The pattern with which the arcs appear in a graph is referred to as either the *structure* of the graph or the *configuration* of the arcs. In the context of this book it is assumed that the vertices u and v incident on each arc are distinct (if $u = v$ the arc is called a *loop*) and that there is at most one arc between them (so that (u, v) uniquely identifies an arc). The simplest structure is the *empty graph*, that is the graph with no arcs; at the other end of the spectrum are *saturated graphs*, in which each node is connected to every other node. Graphs between these two extremes are said to be *sparse* if they have few arcs compared to the number of nodes, and *dense* otherwise. While the distinction between these two classes of graphs is rather vague, a graph is usually considered sparse if $O(|E| + |A|) = O(|\mathbf{V}|)$.

The structure of a graph determines its properties. Some of the most important deal with *paths*, sequences of arcs or edges *connecting* two nodes, called *end-vertices* or *end-nodes*. Paths are directed for directed graphs and undirected for undirected graphs, and are denoted with the sequence of vertices $(v_1, v_2, \ldots, v_n)$ incident on those arcs. The arcs connecting the vertices $v_1, v_2, \ldots, v_n$ are assumed to be unique, so that a path passes through each arc only once. In directed graphs it is also assumed that all the arcs in a path follow the same direction, and we say that a path *leads from* v_1 (the tail of the first arc in the path) *to* v_n (the head of the last arc in the path). In undirected and mixed graphs (and in general when referring to a graph regardless of which class it belongs to), arcs in a path can point in either direction or be undirected. Paths in which $v_1 = v_n$ are called *cycles*, and are treated with particular care in BN theory.

The structure of a directed graph defines a partial ordering of the nodes if the graph is *acyclic*, that is, if it does not contain any cycle or loop. In that case the graph is called a *directed acyclic graph* (DAG). This ordering is called an *acyclic* or *topological ordering* and is induced by the direction of the arcs. It must satisfy the following property:

$$\{\exists \text{ a path from } v_i \text{ to } v_j\} \Rightarrow \{v_i \prec v_j\} \tag{A.1}$$

so when a node v_i precedes v_j there can be no path from v_j to v_i. According to this definition the first nodes are the *root nodes*, which have no incoming arcs, and the last are the *leaf nodes*, which have no outgoing arcs. Furthermore, if there is a path leading from v_i to v_j, v_i precedes v_j in the sequence of the ordered nodes. In this case v_i is called an *ancestor* of v_j and v_j is called a *descendant* of v_i. If the path is composed by a single arc by analogy x_i is a *parent* of v_j and v_j is a *child* of v_i.

Consider, for instance, node A in the DAG shown in Figure A.2. Its neighbourhood is the union of the parents and children; adjacent nodes necessarily fall into one of these two categories. Its parents are also ancestors, as they necessarily precede A in the topological ordering. Likewise, children are also descendants. Two examples of topological ordering induced by the graph structure are

$$(\{\mathrm{F}, \mathrm{G}, \mathrm{H}\}, \{\mathrm{C}, \mathrm{B}\}, \{\mathrm{A}\}, \{\mathrm{D}, \mathrm{E}\}, \{\mathrm{L}, \mathrm{K}\}) \tag{A.2}$$

and

$$(\{\mathrm{F}\}, \{\mathrm{B}\}, \{\mathrm{G}\}, \{\mathrm{C}\}, \{\mathrm{A}, \mathrm{H}\}, \{\mathrm{E}\}, \{\mathrm{L}\}, \{\mathrm{D}\}, \{\mathrm{K}\}). \tag{A.3}$$

Indeed, the nodes are only *partially ordered*; for example, no ordering can be

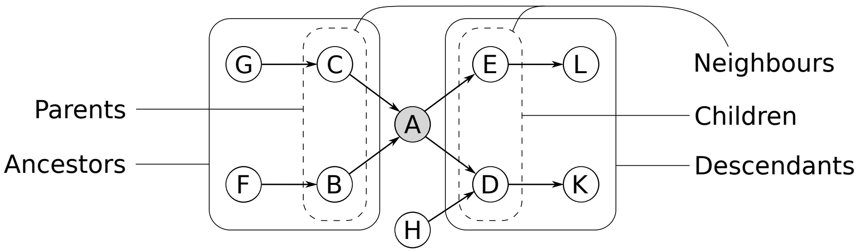


Figure A.2
Parents, children, ancestors, descendants and neighbours of node A in a directed graph.

established among root nodes or leaf nodes. In the examples above, the nodes enclosed in each set of curly brackets can be permuted in any possible way and still satisfy Equation (A.1) for the DAG in Figure A.2. As a result, in practice the topological ordering of a DAG is defined over a set of unordered sets of nodes, denoted with $V_i = \{v_{i_1}, \ldots, v_{i_k}\}$, defining a partition of $\mathbf{V}$.

A.3 Further Reading

For a broader coverage of the properties of directed and mixed graphs, we refer the reader to the monograph by Bang-Jensen and Gutin (2009), which at the time of this writing is the most complete reference on the subject. For undirected graphs, we refer to the classic book of Diestel (2005).

B

Probability Distributions

B.1 General Features

A *probability distribution* is a function that assigns a *probability* to each measurable subset of a set of *events*. It is associated with a *random variable*, here denoted as X.

A *discrete probability distribution* can assume only a finite or countably infinite number of values $\mathbf{U}$, such as

$$\{A, B, C, D\} \qquad \text{or} \qquad \{[0,5], (5,7], (8,10]\} \qquad \text{or} \qquad n \in \mathbb{N}, \tag{B.1}$$

and is characterised by a probability function $\Pr(\cdot)$ that satisfies

$$\sum_{u \in \mathbf{U}} \Pr(X = u) = 1 \qquad \text{and} \qquad \Pr(X = u) \in [0,1] \qquad \text{for all } u. \tag{B.2}$$

A *continuous probability distribution* must assume an infinite number of values $\mathbf{U}$, typically $\mathbb{R}$ or an interval of real numbers, and is characterised by a *density function* $\mathrm{f}(\cdot)$ that satisfies

$$\int_{\mathbf{U}} \mathrm{f}(X = u)du = 1 \qquad \text{and} \qquad \mathrm{f}(X = u) \geqslant 0 \qquad \text{for all } u. \tag{B.3}$$

$\Pr(u)$ and $\mathrm{f}(u)$ are often used as shorthand notations when it is clear which variable we are referring to, along with more precise notations such as $\Pr_X(u)$ and $\mathrm{f}_X(u)$. In addition, sometimes the distinction between discrete and continuous distributions is unimportant and we use $\Pr(\cdot)$ instead of $\mathrm{f}(\cdot)$. Note that $\Pr(u) \leqslant 1$, but $\mathrm{f}(u)$ has no such restriction and can be an arbitrarily large positive number. Furthermore, note that points have null measure, and thus zero probability, for all continuous distributions; intervals can have positive probability. We say that X *follows* or *is distributed as* a certain probability distribution, and we denote it with $X \sim \Pr(\cdot)$ or $X \sim \mathrm{f}(\cdot)$.

Summing $\Pr(\cdot)$ over $\mathbf{U}$ or some subset of it produces many quantities that are extremely useful to study distributions when $\mathbf{U}$ admits an ordering. For instance, this is the case for discrete random variables defined over $\mathbb{N}$ and continuous ones. Some examples are:

- The *cumulative distribution function* (CDF),

$$F(u^*) = \sum_{u \leqslant u^*} \Pr(u) \qquad \text{or} \qquad F(u^*) = \int_{-\infty}^{u^*} \Pr(u) du, \tag{B.4}$$

  which is the probability of getting value no greater than u^*. The quantile function is just the inverse of $F(\cdot)$:

$$Q(p) = u \Longleftrightarrow F(u) = p. \tag{B.5}$$

  $F(\cdot)$ is usually estimated from an observed sample $x_1, \ldots, x_n$ with the *empirical cumulative distribution function* (ECDF),

$$\widehat{F}(u^*) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(x_i <= u^*), \tag{B.6}$$

  the proportion of the observations that have a value no greater than u^*.

- The *mean*, *expected value* or *expectation*,

$$\mathrm{E}(X) = \sum_{u \in \mathbf{U}} u \Pr(u) \qquad \text{or} \qquad \mathrm{E}(X) = \int_{\mathbf{U}} u \Pr(u) du, \tag{B.7}$$

  usually estimated with the *empirical mean* $\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$. Note that for some distributions $\mathrm{E}(X)$ may not be finite.

- The *variance*,

$$\mathrm{VAR}(X) = \mathrm{E}\left[(X - \mathrm{E}(X))^2\right] = \mathrm{E}(X^2) - \mathrm{E}(X)^2 \tag{B.8}$$

  which measures how much the distribution of X is spread around $\mathrm{E}(X)$. Its positive square root is called the *standard deviation* of X. The variance is often estimated as $\frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^2$. Again, for some distributions $\mathrm{VAR}(X)$ may not be finite.

Both mean and variance are easy to compute for *linear transformations* of X, that is, when adding and multiplying by some constants $a, b \in \mathbb{R}$:

$$\mathrm{E}(aX + b) = a\,\mathrm{E}(X) + b \qquad \text{and} \qquad \mathrm{VAR}(aX + b) = a^2\,\mathrm{VAR}(X). \tag{B.9}$$

B.2 Marginal and Conditional Distributions

All the definitions above implicitly assume that X takes values in a one-dimensional space $\mathbf{U}$ such as $\mathbb{N}$ or $\mathbb{R}$; in that case X is called a *univariate* *random variable*. On the other hand, a *multivariate random variable* $\mathbf{X}$ takes values in a multi-dimensional space such as $\mathbb{R}^k, k \geqslant 2$. It is also called a *random vector* because it can be characterised by the *joint distribution* of a vector of univariate variables,

$$\mathbf{X} = [X_1 \; X_2 \; \cdots \; X_k], \tag{B.10}$$

one for each dimension of $\mathbf{X}$. Therefore, integrals and summations can be used as before but have to be defined over all the k dimensions. For example, the mean of $\mathbf{X}$ is a vector of length k,

$$\mathrm{E}(\mathbf{X}) = [\mathrm{E}(X_1) \; \mathrm{E}(X_2) \; \cdots \; \mathrm{E}(X_k)], \tag{B.11}$$

and the variance is replaced by a $k \times k$ *covariance matrix*

$$\mathrm{COV}(\mathbf{X}) = \begin{bmatrix} \mathrm{VAR}(X_1) & \mathrm{COV}(X_1, X_2) & \cdots & \mathrm{COV}(X_1, X_k) \\ \mathrm{COV}(X_2, X_1) & \mathrm{VAR}(X_2) & \cdots & \mathrm{COV}(X_2, X_k) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{COV}(X_k, X_1) & \mathrm{COV}(X_k, X_2) & \cdots & \mathrm{VAR}(X_k) \end{bmatrix} \tag{B.12}$$

where $\mathrm{COV}(X_i, X_j) = \mathrm{E}(X_i X_j) - \mathrm{E}(X_i)\,\mathrm{E}(X_j)$ is the *covariance* between X_i and X_j; note that $\mathrm{COV}(X_i, X_i) = \mathrm{VAR}(X_i)$ and that $\mathrm{COV}(X_i, X_j) = \mathrm{COV}(X_j, X_i)$. The empirical estimate of $\mathrm{COV}(X_i, X_j)$ is $\frac{1}{n} \sum_{l=1}^{n} (x_{li} - \overline{x}_i)(x_{lj} - \overline{x}_j)$, where x_{li} is the lth observation for X_i and x_{lj} is the lth observation for X_j. A related quantity which is easier to interpret is *correlation*, defined as

$$\mathrm{COR}(X_i, X_j) = \frac{\mathrm{COV}(X_i, X_j)}{\sqrt{\mathrm{VAR}(X_i)}\sqrt{\mathrm{VAR}(X_j)}} \tag{B.13}$$

and estimated using the empirical estimates of variance and covariance.

The univariate linear transformation in Equation (B.9) can be reformulated using a $h \times 1$ vector $\mathbf{b}$ and a $h \times k$ matrix A:

$$\mathrm{E}(A\mathbf{X} + \mathbf{b}) = A\,\mathrm{E}(\mathbf{X}) + \mathbf{b} \quad \text{and} \quad \mathrm{COV}(A\mathbf{X} + \mathbf{b}) = A\,\mathrm{COV}(\mathbf{X})A^T. \tag{B.14}$$

The distribution of each X_i in $\mathbf{X}$ can be considered without taking into account its relationship with other variables; in this case it is called the *marginal distribution* of X_i. We can derive it from the distribution of $\mathbf{X}$ by summing or integrating over all the possible values $\mathbf{U_j}, j \neq i$ of all $X_j, j \neq i$:

$$\Pr(X_i) = \sum_{u_1 \in \mathbf{U_1}} \cdots \sum_{u_{i-1} \in \mathbf{U_{i-1}}} \sum_{u_{i+1} \in \mathbf{U_{i+1}}} \cdots \sum_{u_k \in \mathbf{U_k}} \Pr(\mathbf{X}), \tag{B.15}$$

$$\Pr(X_i) = \int_{\mathbf{U_1}} \cdots \int_{\mathbf{U_{i-1}}} \int_{\mathbf{U_{i+1}}} \cdots \int_{\mathbf{U_k}} \Pr(\mathbf{X}) d\mathbf{X}. \tag{B.16}$$

Extending these definitions to obtain the marginal distribution of more than one variable is trivial; summation (or integration) is carried out as above, just on all the other variables in $\mathbf{X}$.

Another distribution of interest for X_i is its *conditional distribution*, that is, the distribution of X_i when the values of some other variables $X_{j_1}, \ldots, X_{j_m}$ in $\mathbf{X}$ are fixed to specific values. It can be derived as follows:

$$\Pr(X_i = u_i \mid X_{j_1} = u_{j_1}, \ldots, X_{j_m} = u_{j_m}) = \\ \frac{\Pr(X_i = u_i, X_{j_1} = u_{j_1}, \ldots, X_{j_m} = u_{j_m})}{\Pr(X_{j_1} = u_{j_1}, \ldots, X_{j_m} = u_{j_m})} \quad \text{(B.17)}$$

Again, extending this definition to include more than one variable just requires adding the corresponding terms in the probability and density functions.

B.3 Discrete Distributions

Discrete distributions in common use for BNs are the *binomial* and the *multinomial* distributions and we will focus on them. Nevertheless, as shown in Chapter 3, any distribution can be used in the general case so we will also mention others that are important in practical situations.

B.3.1 Binomial Distribution

The binomial distribution models the number of successes in a sequence of n independent experiments with two possible outcomes; each experiment yields a positive outcome (often called a *success*) with identical probability p. It is usually denoted with $Bi(n, p)$ and has probability function

$$\Pr(X = x) = \binom{n}{x} p^x (1-p)^{(n-x)}, \quad n \in \mathbb{N}, x \in \{0, \ldots n\}, p \in [0,1]. \quad \text{(B.18)}$$

When $p = 0$ (or $p = 1$) the distribution is degenerate in the sense that X can take only the value 0 (or n). Expectation and variance are easily calculated:

$$\mathrm{E}(X) = np \qquad \text{and} \qquad \mathrm{VAR}(X) = np(1-p). \quad \text{(B.19)}$$

So, for n fixed, the variance is maximal when $p = \frac{1}{2}$ and tends to zero when p tends to 0 or 1.

The maximum likelihood estimate of p is simply $\hat{p} = \frac{r}{n}$, the number r of successes over the number n of experiments. Such an estimate is problematic when $r = 0$ or $r = n$, as discussed in Section 1.4.

B.3.2 Multinomial Distribution

The multinomial distribution is the multivariate extension of the binomial, which is required when there are three or more possible outcomes instead of two. Strictly speaking, the binomial is the multivariate distribution of the frequencies of the two outcomes; but as the frequency of one outcome is uniquely determined from the other as $(n - X)$, it is redundant to make it explicit in the notation. Along with the binomial, it's called a *categorical* distribution if the outcomes do not have any explicit ordering.

Given a sequence of n independent trials each having identical probabilities $\mathbf{p} = (p_1, \ldots, p_k)$ for k possible outcomes, the vector of the associated counts $\mathbf{X} = (X_1, \ldots, X_k)$ is said to follow a multinomial distribution and it is denoted as $Mu(n, \mathbf{p})$. The probability function is

$$\Pr(\mathbf{X} = \mathbf{x}) = \frac{n!}{x_1! x_2! \cdots x_k!} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}, \quad n \in \mathbb{N}, \sum_i x_i = n, \sum_i p_i = 1. \tag{B.20}$$

It is important to note that even though each X_i is a random variable following a $Bi(n, p_i)$, $X_1, \ldots, X_k$ are linked by the simple fact to sum to n. As a consequence, they are negatively correlated ($\mathrm{COR}(X_i, X_j) < 0$) and the covariance matrix is not full rank since $\mathrm{VAR}(\sum_i X_i) = 0$. Furthermore,

$$\mathrm{E}(\mathbf{X}) = n\mathbf{p} \qquad \text{and} \qquad \mathrm{VAR}(\mathbf{X}) = n\left(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T\right), \tag{B.21}$$

where $\mathrm{diag}(\mathbf{p})$ denotes the diagonal matrix with vector $\mathbf{p}$ on its diagonal.

B.3.3 Other Common Distributions

B.3.3.1 Bernoulli Distribution

The *Bernoulli distribution*, $Ber(p)$ is a particular case of $Bi(n, p)$ when $n = 1$.

B.3.3.2 Poisson Distribution

The *Poisson* distribution can be derived from the binomial distribution. Consider a rare animal living in a particular region. If we partition that region into N land plots, when N is large enough the probability that two or more animals are present in any plot is negligible. While the number of animals can be modelled as a $Bi(N, p)$ where the probability p depends on the size of the plots, it is logical to assume that in fact $p = \frac{\lambda}{N}$.

It occurs that when $N \to \infty$,

$$Bi\left(N, \frac{\lambda}{N}\right) \to Pois(\lambda), \tag{B.22}$$

where $Pois(\lambda)$ is a Poisson distribution, and λ is in fact the density of presence in the region. The probability function of $Pois(\lambda)$ is

$$\Pr(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}, \qquad x \in \mathbb{N}, \lambda \geqslant 0, \tag{B.23}$$

and both mean and variance are equal to λ:

$$\mathrm{E}(X) = \lambda \qquad \text{and} \qquad \mathrm{VAR}(X) = \lambda. \tag{B.24}$$

B.4 Continuous Distributions

The *normal* or *Gaussian distribution* plays a central role among continuous distributions. Like the binomial distribution, it can be conveniently extended into a multivariate distribution. Some indications about the estimation of the parameters of the normal distribution are provided in Section B.2. In addition, we also consider the *beta* distribution (and its multivariate version) for its close relationship with the binomial (multinomial) discrete distribution.

B.4.1 Normal Distribution

A normal or Gaussian distribution has density

$$\Pr(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}, \quad x,\mu \in \mathbb{R}, \sigma^2 > 0. \tag{B.25}$$

It is denoted with $N(\mu,\sigma^2)$. Some simple calculations show that the two parameters μ and σ^2 have a very direct interpretation, since

$$\mathrm{E}(X) = \mu \qquad \text{and} \qquad \mathrm{VAR}(X) = \sigma^2. \tag{B.26}$$

When $\mu = 0$ and $\sigma^2 = 1$, the associated random variable follows a *standard* normal distribution, and it is straightforward to check that $\frac{X-\mu}{\sigma} \sim N(0,1)$. It is also worth noting that even if X is defined over the whole $\mathbb{R}$ and any value between $-\infty$ and $+\infty$ has a strictly positive density,

$$\Pr\left(X \notin [\mu - 4\sigma, \mu + 4\sigma]\right) < 5.10^{-5}. \tag{B.27}$$

B.4.2 Multivariate Normal Distribution

The multivariate extension of the normal distribution is called multivariate normal, multinormal or multivariate Gaussian distribution. Its density function is

$$\Pr(\mathbf{x};\boldsymbol{\mu},\Sigma) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \quad \mathbf{x},\boldsymbol{\mu} \in \mathbb{R}^k, \tag{B.28}$$

where $\boldsymbol{\mu}$ is a $k \times 1$ mean vector and Σ is a $k \times k$ positive-semidefinite covariance matrix. The distribution is denoted with $N_k(\boldsymbol{\mu},\Sigma)$.

A multivariate normal random variable has two very convenient properties:

1. An affine transformation of a normal vector is still a normal vector. More precisely, supposing that $\mathbf{b}$ is a vector of size h and A an $h \times k$ matrix:

$$\mathbf{X} \sim N_k\left(\boldsymbol{\mu},\Sigma\right) \implies \quad A\mathbf{X} + \mathbf{b} \sim N_h\left(\mathbf{b} + A\boldsymbol{\mu}, A\Sigma A^T\right) \tag{B.29}$$

2. The conditional distribution for a subset A of components of $\mathbf{X}$ given the other components B of a multivariate normal is again a multivariate normal distribution with

$$\begin{aligned}\boldsymbol{\mu}_{A|B} &= \boldsymbol{\mu}_A + \Sigma_{AB}\Sigma_{BB}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_B) \\ \Sigma_{A|B} &= \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}\end{aligned} \tag{B.30}$$

where

$$\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B\end{bmatrix} \qquad \text{and} \qquad \Sigma = \begin{bmatrix}\Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB}\end{bmatrix}. \tag{B.31}$$

Note that Equation (B.30) implicitly assumes that Σ_{BB} is full rank; that is, no redundancy occurs in the conditioning variables which are linearly independent. Theoretically, this is not an insurmountable problem since redundant variables can be removed without changing the nature of the problem. However, from a practical point of view it is often a difficult task since the assessment of the rank of a matrix (even a non-negative one) is a non-trivial numerical problem.

B.4.3 Other Common Distributions

Two important distributions related to the univariate normal distribution are *Student's t* and the *Chi-square* (χ^2) distribution. *Beta* and *Dirichlet* distributions have been included, because they are conjugate distributions (see Section B.5) of the binomial and the multinomial.

B.4.3.1 Chi-square Distribution

The χ^2 distribution can be viewed as the sum of the squares of ν independent standard normal random variables. ν is then the only parameter, called its *degrees of freedom*; the distribution is denoted with χ^2_ν. Accordingly, the sum of two independent χ^2 random variables with ν_1 and ν_2 degrees of freedom follows a χ^2 distribution with $\nu_1 + \nu_2$ degrees of freedom. The expectation and variance are :

$$\mathrm{E}(X) = \nu \qquad \text{and} \qquad \mathrm{VAR}(X) = 2\nu. \tag{B.32}$$

This distribution can be generalised by introducing a non-centrality parameter induced by non-centred generative normal variables of unit variance. More precisely, let

$$X = \sum_{i=1}^{\nu} U_i^2 \text{ where the } U_i \text{ are independent } N(\mu_i, 1); \tag{B.33}$$

then $X \sim nc\chi^2(\lambda)$ with $\lambda = \sum_{i=1}^{\nu}\left(\mu_i^2\right)$. Equation (B.32) generalises with

$$\mathrm{E}(X) = \nu + \lambda, \tag{B.34}$$

$$\mathrm{VAR}(X) = 2(\nu + 2\lambda), \tag{B.35}$$

$$\mathrm{E}\left([X - E(X)]^3\right) = 8(\nu + 3\lambda). \tag{B.36}$$

It can also be shown that the χ^2 distribution is a particular case of the *gamma* distribution, allowing the ν parameter to take non-integer but positive values.

B.4.3.2 Student's t Distribution

The need for the Student's t distribution arises when standardising a normal random variable with an estimate of its variance. It can be introduced as the ratio of a standard normal variable with the square root of an independent χ^2_ν variable. Then, it has only one parameter, the degrees of freedom of the χ^2 variable, $\nu \in \mathbb{N}^+$. Its density function is bell-shaped and symmetric like the normal's, and differs mainly in the thickness of the tails. For high values of ν Student's t is well approximated by a $N(0,1)$ distribution, while low values result in fatter tails and assign more probability to extreme values. When $\nu > 2$, the expectation and variance are:

$$\mathrm{E}(X) = 0 \qquad \text{and} \qquad \mathrm{VAR}(X) = \frac{\nu}{\nu - 2}. \tag{B.37}$$

As for the χ^2 distribution, a second *non-centrality* parameter can be considered to generalise the distribution.

B.4.3.3 Beta Distribution

The beta distribution, denoted with $Beta(a, b)$, is a common choice for a random variable restricted to the $[0, 1]$ interval. Its density function is

$$\Pr(x; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}, \qquad a, b > 0, x \in [0,1], \tag{B.38}$$

which simplifies to a binomial if $a, b \in \mathbb{N}$ because $\Gamma(n) = (n-1)!$ for $n \in \mathbb{N}$. In that case, a plays the role of x, b plays the role of $n - x$ and x plays the role of p in Equation (B.18). This is why it is the conjugate distribution (see Section B.5) in the beta-binomial model in Bayesian statistics. The expectation and variance are:

$$\mathrm{E}(X) = \frac{a}{a+b} \qquad \text{and} \qquad \mathrm{VAR}(X) = \frac{ab}{(a+b)^2(a+b+1)}. \tag{B.39}$$

This distribution is quite flexible when varying the parameters. It can easily be transposed to any finite non point interval $[a, b]$ by an affine transformation of the form

$$a + (b-a)X \qquad \text{where} \qquad a, b \in \mathbb{R}. \tag{B.40}$$

B.4.3.4 Dirichlet Distribution

The Dirichlet distribution is a multivariate generalisation of the beta distribution in the same way as the multinomial is a generalisation of the binomial. For instance, a Dirichlet with two components is just a beta distribution, the two components summing to one. The distribution is often denoted by $Dir(\boldsymbol{a})$, $\boldsymbol{a} = (a_1, a_2, \ldots, a_k)$, and its density function is

$$\Pr(X = \mathbf{x}) = \frac{\Gamma(\sum_i a_i)}{\Gamma(a_1)\Gamma(a_2)\cdots\Gamma(a_k)} x_1^{a_1} x_2^{a_2} \cdots x_k^{a_k}, \qquad \sum_i x_i = 1, x_i \in [0,1], a_1, \ldots, a_k > 0. \tag{B.41}$$

As was the case for the beta with the binomial, the similarity with the multinomial distribution is apparent: x_i plays the role of the p_i and a_i plays the role of the x_i in Equation (B.20). Denoting with A the sum of the parameters $\sum_i a_i$, the expectation and variance are:

$$\mathrm{E}(X) = \frac{1}{A}\boldsymbol{a} \quad \text{and} \quad \mathrm{VAR}(X) = \frac{1}{A(1+A)}\left(\mathrm{diag}(\boldsymbol{a} - \frac{1}{A}\boldsymbol{a}\boldsymbol{a}^T\right). \tag{B.42}$$

From a Bayesian point of view, the Dirichlet is the conjugate distribution (see Section B.5) for the parameter $\mathbf{p}$ of a multinomial distribution.

B.5 Conjugate Distributions

Several times in the above, we have mentioned that two distributions are conjugate. Here we will try to convey the fundamental idea of conjugacy in a few words. The concept originates in the framework of Bayesian statistics.

Consider two random variables A and B whose joint probability is defined by the marginal and conditional distributions,

$$\Pr(A, B) = \Pr(A)\Pr(B \mid A). \tag{B.43}$$

When the conditional distribution $\Pr(A \mid B)$ belongs to the same family as $\Pr(A)$, then we say that $\Pr(A)$ is the *conjugate* of $\Pr(B \mid A)$. A simple example is the beta-binomial pair:

- let $p \sim Beta(a, b)$,
- and let $(r \mid p) \sim Bi(n, p)$,
- then $(p \mid r) \sim Beta(a + r, b + n - r)$.

The main advantage of conjugacy is to simplify to the extreme the computation of conditional distributions.

B.6 Further Reading

Literature contains many books on probability theory with varying levels of complexity. An introductory book focusing on commonly used distributions and their fundamental properties is Ross (2012). More advanced books, which cover probability theory from a more theoretical perspective, are DeGroot and Scherivsh (2012); Ash (2000); Feller (1968). A thorough derivation of the basis of probability theory as an off-shoot of measure theory is provided in Loève (1977). Furthermore, an encyclopedia about probability distributions is the series of Johnson's and Kotz's books (Kotz et al., 2000; Johnson et al., 1994, 1995, 1997, 2005).

C

A Note about Bayesian Networks

The relationship between *Bayesian networks* and *Bayesian statistics* is often unclear because of how similar their names are and the common qualifier of *Bayesian*. Even though they are connected in many ways, it is important to underline the differences between the two.

C.1 Bayesian Networks and Bayesian Statistics

The basic difference originates in the term *statistics*. A statistical procedure consists in summarising the information comprised in a data set, or more generally in performing some inference within the framework of some probabilistic model. A Bayesian network is just a complex, principled way of proposing a probabilistic model on a set of variables, without necessarily involving data.

The confusion originates by the following points:

- It is very convenient to present the Bayesian statistics as a BN of this kind: $(Parameters) \mapsto (Data)$. The marginal distribution of the multivariate node $(Parameters)$ is the prior; the conditional distribution of the multivariate node $(Data)$ is the likelihood; then the posterior distribution is just the conditional distribution of the multivariate node $(Parameters)$ for the observed $(Data)$. In essence, a Bayesian statistical procedure can be described as a BN.

- When querying a BN as in Section 4.6.1, by fixing some nodes of the BN and updating the local probability distributions in this new context, we are in fact applying a Bayesian statistical approach, if we accept to assimilate the fixed node(s) to observation(s).

- The probabilistic model used in a statistical approach (whether Bayesian or not) can be defined through a BN. That is apparent when using JAGS in Chapter 3 and Section 5.2.

- A BN is not always known in advance and many learning algorithms have been proposed to estimate them from data. This has been detailed in various parts of the book (Sections 1.5, 2.5 and 4.5).

But it is important to be clear in the fact that the probabilistic model used in a statistical approach can be defined without the help of BNs; and also that the learning of a BN from a data set can be in a Bayesian context or not.

Glossary

Here we have collected some of the technical terms used within the book. For those terms referring to graph theory, Appendix A provides a short but comprehensive introduction to the field.

acyclic graph: An acyclic graph is a graph without any cycle.

adjacent nodes: Two nodes are said to be adjacent when they are linked by an arc or an edge.

ancestor: An ancestor of a node is a node that precedes it within a directed path and therefore in the topological ordering of the graph.

arc: An arc is a directed link between two nodes, usually assumed to be distinct. Nodes linked by an arc have a direct parent-child relationship: the node on the tail of the arc is the parent node, the node on the head of the arc is the child node. A node can have no, one or several parents; a node can have no, one or several children. Sometimes the expression "undirected arcs" is used to refer to edges, *i.e.*, undirected links.

Bayes, Thomas: Bayes was an English Presbyterian minister (1701-1761). He wrote some notes about deducing causes from effects, published after his death by a friend of his under the title "*An Essay towards solving a Problem in the Doctrine of Chances*". They contain the original formulation of the conditional probability theorem, which is known as *Bayes' theorem*. This important result was independently published in a better mathematical form by Pierre Simon Laplace.

Bayesian networks (BNs): A Bayesian network defines a joint probability distribution over a set of variables and the corresponding local univariate distributions. The DAG associated to the BN determines whether each of them is marginal or conditional on other variables. Each node is associated to one variable in the BN. Root nodes are given marginal distributions; other nodes have distributions conditional only on the values of the respective parents. A formal link between the conditional independencies in the BN and a DAG is given in Definition 4.1. The term *Bayesian network* comes from the recursive use of Bayes' theorem to decompose the joint distribution into the individual distributions of the nodes and other distributions of interest, following the dependence structure given by the DAG.

child node: See the definition for *arc*.

clique: In an undirected graph, a clique is a subset of nodes such that every two nodes in the subset are adjacent. A clique is said to be maximal when including another node in the clique means it is no longer a clique.

conditional distribution: The conditional probability distribution of a random variable A with respect to another random variable B is its probability distribution when B is known or restricted, particularly to a given value b. Generally, this distribution depends on b and it is denoted by $\Pr(A \mid B = b)$. When A and B are independent, $\Pr(A \mid B = b) = \Pr(A)$ for any b, that is, the conditional distribution is identical to the marginal distribution and it does not depend on b.

configuration: The configuration of the arcs or edges of a graph is synonymous with the structure of a graph.

connected: Two nodes are said to be connected when there exists a path linking them; when the path is of length one (*e.g.*, a single arc or edge) they are adjacent.

connected graph: A connected graph is an undirected graph in which every pair of nodes is connected by a path; or a directed graph whose skeleton is connected (*i.e.*, disregarding the direction of the arcs and substituting them with edges).

convergent connection: A convergent connection is one of the three fundamental connections between three nodes (see Figure 1.3 on page 22). Two nodes are parents of the third: $A \rightarrow B \leftarrow C$.

CPDAG: Acronym for Completed Partially Directed Acyclic Graph. The CPDAG of a (partially) directed acyclic graph is the partially directed graph built on the same set of nodes, keeping the same v-structures and the same skeleton, completed with the compelled arcs. Two DAGs having the same CPDAG are equivalent in the sense that they result in BNs describing the same probability distribution.

cycle: A cycle is a path having identical end-nodes, that is, starting from and ending on the same node. Note that it is possible to have cycles in both directed and undirected graphs.

DAGs: Acronym for Directed Acyclic Graph. It is a directed graph with no cycles. Sometimes, there is some confusion between BNs and DAGs; a DAG expresses only the conditional independence structure of a BN. Nevertheless, the DAG representation is very useful to discuss the construction or the interpretation of a BN.

d-separation: Two subsets of variables, say $\mathbf{A}$ and $\mathbf{B}$, are said to be d-separated in a BN with respect to a third, say $\mathbf{C}$, when they are graphically separated according to the conditions in Definition 4.2. Following Definition

4.3, this also means that any variable in **A** is independent from any variable in **B** conditional on the variables in **C**.

descendant: A descendant of a node is a second node following it in a directed path.

directed edge: Synonymous with arc.

directed graph: A directed graph is a graph where all links are arcs (*i.e.*, have a direction).

divergent connection: A divergent connection is one of the three fundamental connections between three nodes (see Figure 1.3 on page 22). A parent with two children: $A \leftarrow B \rightarrow C$.

edge: An edge is an undirected link between two nodes. Sometimes the expression "directed edge" is used to refer to arcs.

empty graph: An empty graph is a graph with no edges or arcs at all, the node set is commonly assumed to be non-empty.

end-node: An end-node is either the first or the last node of a given path.

graph: A graph consists in a non-empty set of nodes where each pair of nodes is linked by zero or one link. Links can be arcs or edges; and then the graph is said to be directed (only arcs), undirected (only edges) or partially directed (both arcs and edges).

independence: In a probabilistic framework two random variables are said to be independent if and only if the conditional probability distribution of each of them with respect to the other is identical to the marginal distribution. It is a symmetric property. At an intuitive level, this means that knowing the value of one variable does not modify our knowledge about the distribution of the other.

joint distribution: The joint probability distribution of a set of random variables provides the probability of any combination of events over them. If the variables are independent the joint distribution simplifies into the product of the marginal distributions.

leaf node: A leaf node is a node in a DAG without any child.

link: An arc or an edge connecting two nodes.

marginal distribution: The marginal probability distribution of a random variable is its probability distribution, defined without taking into account other random variables that may be related. The knowledge of the marginal distributions of two variables is not sufficient to deduce their joint distribution without further assumptions, such as that they are independent. On the contrary, it is always possible to deduce marginal distributions from the joint distribution. Marginal distributions can also be derived for subsets of random variables.

mixed graphs: Synonymous with partially directed graphs.

neighbour-nodes: The neighbours of a node are all the nodes that are adjacent to it.

nodes: The nodes, together with the arcs, are the fundamental components of DAGs; and in the context of BNs they also refer to the random variables in the model.

parent node: See the definition for *arc*.

partially directed: See the definition of *graph*.

path: A path is a set of successive links. It is usually described by an ordered sequence of linked nodes. When the path comprises some arcs, all of them must follow the same direction; then it is a *directed path*.

probability distribution: The probability distribution of a random variable is a function defining the probability of each possible event described by the random variable. A probability is a real number in $[0, 1]$. A probability of zero means that the event is impossible; a probability of one means that the event is certain. Even though probability distributions have different mathematical forms for discrete and continuous variables, they interact in similar ways. When the random variable involves several components, the *joint distribution* refers to events describing all the components; a *marginal distribution* refers to events describing a subset (usually of size one) of components; a *conditional distribution* refers to events describing a subset of components when a second, disjoint subset is fixed to some conditioning event.

random variable: A random variable is a variable whose value follows a probability distribution. Random variables can take very different forms, such as discrete and continuous, or univariate and multivariate.

root node: A root node is a node in a DAG without any parent.

saturated graphs: A graph in which all nodes are adjacent to each others.

serial connection: A serial connection is one of the three fundamental connections between three nodes (see Figure 1.3 on page 22). Grandparent, parent, child: $A \rightarrow B \rightarrow C$.

skeleton: The skeleton of a (partially) directed graph is its underlying undirected graph, *i.e.*, the undirected graph obtained by disregarding all arc directions.

structure: The structure of a BN is the set of its conditional independencies; they are represented by its DAG.

topological order: An ordering on the nodes of a DAG is said to be topological when it is consistent with every directed path of the DAG. Often a topological order is not unique, and a DAG may admit more than one ordering.

tree: A tree is an undirected graph in which every pair of nodes is connected with exactly *one* path, or a directed path in which all nodes have exactly one parent except a single *root node.*

undirected graph: A graph comprising only undirected links.

v-structure: A v-structure is a convergent connection in which the two parents are not linked. Note that several v-structures can be centred on the same node (*i.e.*, have the same child node).

vertex/vertices: Synonymous for node(s).

Solutions

Exercises of Chapter 1

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?**

1. Parents and children of each node are as follows.

   - Age (`A`) has no parent, and Education (`E`) is its only child.
   - Sex (`S`) has no parent, and Education (`E`) is its only child.
   - Education (`E`) has two parents, Age (`A`) and Sex (`S`), and two children, Occupation (`O`) and Residence (`R`).
   - Occupation (`O`) has one parent, Education (`E`), and one child, Travel (`T`).
   - Residence (`R`) has one parent, Education (`E`), and one child, Travel (`T`).
   - Travel (`T`) has two parents, Occupation (`O`) and Residence (`R`), and no child.

2. The fundamental connections in the DAG are as follows.

   - $\mathtt{A} \rightarrow \mathtt{E} \leftarrow \mathtt{S}$ (convergent).
   - $\mathtt{A} \rightarrow \mathtt{E} \rightarrow \mathtt{O}$ (serial).
   - $\mathtt{A} \rightarrow \mathtt{E} \rightarrow \mathtt{R}$ (serial).
   - $\mathtt{S} \rightarrow \mathtt{E} \rightarrow \mathtt{O}$ (serial).
   - $\mathtt{S} \rightarrow \mathtt{E} \rightarrow \mathtt{R}$ (serial).
   - $\mathtt{O} \leftarrow \mathtt{E} \rightarrow \mathtt{R}$ (divergent).

- E → O → T (serial).
- E → R → T (serial).
- O → T ← R (convergent).

3. Adding A → O does not introduce any cycles; the graph is still a DAG and a valid BN. On the other hand, adding T → E introduces the following cycles: T → E → R → T and T → E → O → T. Therefore, the resulting graph is not acyclic and cannot be part of a valid BN.

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?**

1. The number of parents' configurations is 2 × 3 for E, 2 for R, 2 for O and 4 for T. A and S have no parent.
2. The numbers of free parameters is 2 for A, 1 for S, 6 × 1 for E, 2 × 1 for O, 2 × 1 for R and 4 × 2 for T. Indeed, 1 must be removed from the number of probabilities since their sum is known to be one.
3. The number of free parameters of the global distribution is the sum of the numbers of free parameters of the conditional probability tables, that is, 21 parameters.
4. After adding E → T, Equation (1.1) reads

$$\begin{aligned}&\Pr(\mathtt{A},\mathtt{S},\mathtt{E},\mathtt{O},\mathtt{R},\mathtt{T}) = \\ &\qquad \Pr(\mathtt{A})\Pr(\mathtt{S})\Pr(\mathtt{E}\mid\mathtt{A},\mathtt{S})\Pr(\mathtt{O}\mid\mathtt{E})\Pr(\mathtt{R}\mid\mathtt{E})\Pr(\mathtt{T}\mid\mathtt{E},\mathtt{O},\mathtt{R}).\end{aligned}$$

   The number of free parameters of the local distribution of T increases to 2 × 8 due to the additional parents' configurations; all the other local distributions are unchanged.

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?**

1. The easiest way is to use Equation (1.1) and `model2network` as follows.

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

2.
```
> nodes(dag)
[1] "A" "E" "O" "R" "S" "T"
> arcs(dag)
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
> par <- sapply(nodes(dag), parents, x = dag)
> chld <- sapply(nodes(dag), children, x = dag)
```

3.
```
> modelstring(dag)
[1] "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

4.
```
> survey <- read.table("../chap1/survey.txt",
+                header = TRUE)
> fitted <- bn.fit(dag, survey, method = "bayes")
```

5.
```
> dag2 <- drop.arc(dag, from = "E", to = "O")
```

6. The conditional probability table of O now has only two cells and one dimension, because O has no parent in dag2.

```
> fitted2 <- bn.fit(dag2, survey, method = "bayes")
> dim(coef(fitted$O))

[1] 2 2

> dim(coef(fitted2$O))

[1] 2
```

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?**

1. Conditioning does not have a big effect on the distribution of O.

```
> library(gRain)
> junction <- compile(as.grain(bn.mle))
> querygrain(junction, nodes = "O")$O

O
  emp  self
0.966 0.034

> jage <- setEvidence(junction, "A", states = "young")
> querygrain(jage, nodes = "O")$O

O
   emp   self
0.9644 0.0356

> jage <- setEvidence(junction, "A", states = "adult")
> querygrain(jage, nodes = "O")$O

O
   emp   self
0.9636 0.0364

> jage <- setEvidence(junction, "A", states = "old")
> querygrain(jage, nodes = "O")$O
```

```
O
   emp   self
0.9739 0.0261
```

2. 10^3 simulations are enough for likelihood weighting, 10^4 for logic sampling.

```
> set.seed(123)
> cpquery(bn.mle, event = (O == "emp"),
+   evidence = list(A = "young"), method = "lw",
+   n = 10^3)
[1] 0.96
> cpquery(bn.mle, event = (O == "emp"),
+   evidence = (A == "young"), method = "ls",
+   n = 10^4)
[1] 0.969
```

3.
```
> dag <- bn.net(bn.mle)
```

4.
```
> sapply(nodes(dag), function(z) dsep(dag, "A", "O", z))
    A     E     O     R     S     T
 TRUE  TRUE  TRUE FALSE FALSE FALSE
```

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

```
> rejection.sampling <- function(bn, nsim, event.node,
+     event.value, evidence.node, evidence.value) {
+
+   sims <- rbn(bn, nsim)
+   m1 <- sims[sims[, evidence.node] == evidence.value, ]
+   m2 <- m1[m1[, event.node] == event.value, ]
+   return(nrow(m2)/nrow(m1))
+
+ }#REJECTION.SAMPLING
> rejection.sampling(bn.mle, nsim = 10^4, event.node = "O",
+   event.value = "emp", evidence.node = "A",
+   evidence.value = "young")
[1] 0.966
```

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.**

1.

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

2.

```
> vs <- vstructs(dag, arcs = TRUE)
> hl <- list(nodes = unique(as.character(vs)), arcs = vs)
> graphviz.plot(dag, highlight = hl)
```

3.

```
> hl <- matrix(c("A", "E", "E", "O"), nc = 2,
+         byrow = TRUE)
> graphviz.plot(dag, highlight = list(arcs = hl))
```

4.

```
> bn.fit.barchart(bn$E)
```

5.

```
> library(gRain)
> junction <- compile(as.grain(bn))
> jmale <- setEvidence(junction, "S", states = "M")
> jfemale <- setEvidence(junction, "S", states = "F")
> library(lattice)
> library(gridExtra)
> p1 <- barchart(querygrain(jmale, nodes = "E")$E,
+        main = "Male", xlim = c(0, 1))
> p2 <- barchart(querygrain(jfemale, nodes = "E")$E,
+        main = "Female", xlim = c(0, 1))
> grid.arrange(p1, p2, ncol = 2)
```

Exercises of Chapter 2

2.1 Prove that Equation (2.2) implies Equation (2.3).

Using Bayes' theorem twice, we obtain that

$$\begin{aligned}\mathrm{f}(\mathtt{G}=g \mid \mathtt{C}=c) &= \frac{\mathrm{f}(\mathtt{G}=g, \mathtt{C}=c)}{\mathrm{f}(\mathtt{C}=c)} \\ &= \frac{\mathrm{f}(\mathtt{C}=c \mid \mathtt{G}=g)}{\mathrm{f}(\mathtt{C}=c)}\,\mathrm{f}(\mathtt{G}=g).\end{aligned}$$

If $\mathrm{f}(\mathtt{G}=g \mid \mathtt{C}=c) = \mathrm{f}(\mathtt{G}=g)$ it would imply that $\mathrm{f}(\mathtt{C}=c \mid \mathtt{G}=g) = \mathrm{f}(\mathtt{C}=c)$ which contradicts the first step in the proof.

2.2 Within the context of the DAG shown in Figure 2.1, prove that Equation (2.5) is true using Equation (2.6).

From Equation (2.6), we can obtain $\mathrm{f}(\mathtt{V},\mathtt{W},\mathtt{N})$ integrating out the other variables:

$$\begin{aligned}\mathrm{f}(V,W,N) &= \int_G\int_E\int_C \mathrm{f}(\mathtt{G},\mathtt{E},\mathtt{V},\mathtt{N},\mathtt{W},\mathtt{C}) \\ &= \mathrm{f}(\mathtt{V})\,\mathrm{f}(\mathtt{N}\mid\mathtt{V})\,\mathrm{f}(\mathtt{W}\mid\mathtt{V})\times \\ &\quad \left(\int_G\int_E \mathrm{f}(\mathtt{G})\,\mathrm{f}(\mathtt{E})\,\mathrm{f}(V\mid\mathtt{G},\mathtt{E})\right)\left(\int_C \mathrm{f}(\mathtt{C}\mid\mathtt{N},\mathtt{W})\right) \\ &= \mathrm{f}(\mathtt{V})\,\mathrm{f}(\mathtt{N}\mid\mathtt{V})\,\mathrm{f}(\mathtt{W}\mid\mathtt{V})\end{aligned}$$

and from $\mathrm{f}(V,W,N)$ we can obtain the joint distribution of W and N conditional to V

$$\begin{aligned}\mathrm{f}(\mathtt{W},\mathtt{N}\mid\mathtt{V}) &= \frac{\mathrm{f}(\mathtt{V},\mathtt{W},\mathtt{N})}{\mathrm{f}(\mathtt{V})} \\ &= \mathrm{f}(\mathtt{N}\mid\mathtt{V})\,\mathrm{f}(\mathtt{W}\mid\mathtt{V})\end{aligned}$$

characteristic of the conditional independence.

2.3 Compute the marginal variance of the two nodes with two parents from the local distributions proposed in Table 2.1. Why is it much more complicated for C than for V?

V and C are the two nodes for which we have to compute the variance. For the sake of clarity, we will first replace the numeric values of the coefficient with arbitrary constants k_0, k_1, k_2 and k_3. From the first three equations of Table 2.1, V can be written as

$$\mathtt{V} = k_0 + k_1\mathtt{G} + k_2\mathtt{E} + k_3\epsilon_V$$

where G, E and ϵ_V are independent random variables with variances 10^2, 10^2 and 5^2, respectively, so we can easily find that

$$\begin{aligned}\mathrm{VAR}(\mathtt{V}) &= k_1^2\,\mathrm{VAR}(\mathtt{G}) + k_2^2\,\mathrm{VAR}(\mathtt{E}) + k_3^2\,\mathrm{VAR}(\epsilon_V) \\ &= \left(\frac{1}{2}\right)^2 10^2 + \left(\sqrt{\frac{1}{2}}\right)^2 10^2 + 5^2 \\ &= 10.\end{aligned}$$

For C, things are a little more complicated because

$$\texttt{C} = k_1\texttt{N} + k_2\texttt{W} + k_3\epsilon_C$$

and the variables N and W are not independent, as they share V as a common ancestor. Therefore we have to first compute COV (N, V):

$$\begin{aligned} \mathrm{COV}\left(\texttt{N},\texttt{V}\right) &= \mathrm{COV}\left(h_1\texttt{V}, h_2\texttt{V}\right) \\ &= h_1h_2\,\mathrm{VAR}\left(\texttt{V}\right) \\ &= 0.1 \times 0.7 \times 10^2 \end{aligned}$$

and finally

$$\begin{aligned} \mathrm{VAR}(\texttt{C}) &= k_1^2\,\mathrm{VAR}\left(\texttt{N}\right) + k_2^2\,\mathrm{VAR}\left(\texttt{W}\right) + k_3^2\,\mathrm{VAR}\left(\epsilon_C\right) + 2k_1k_2\,\mathrm{COV}\left(\texttt{N},\texttt{W}\right) \\ &= \left(0.3^2 + 0.7^2 + 0.3 \times 0.7 \times 0.14\right)10^2 + 6.25^2 \\ &= \left(10.00012\right)^2. \end{aligned}$$

2.4 Write an R script using only the rnorm and cbind functions to create a 100×6 matrix of 100 observations simulated from the BN defined in Table 2.1. Compare the result with those produced by a call to cpdist function.

First, it is sensible to initialise the pseudo-random generator and parameterise the desired number of simulations.

```
> set.seed(12345)
> ns <- 100
```

Then the following seven commands answer the first question.

```
> sG <- rnorm(ns, 50, 10)
> sE <- rnorm(ns, 50, 10)
> sV <- rnorm(ns, -10.35534 + 0.5 * sG + 0.70711 * sE, 5)
> sN <- rnorm(ns, 45 + 0.1 * sV, 9.949874)
> sW <- rnorm(ns, 15 + 0.7 * sV, 7.141428)
> sC <- rnorm(ns, 0.3 * sN + 0.7 * sW, 6.25)
> simu1 <- cbind(sG, sE, sV, sN, sW, sC)
```

To perform the equivalent simulation with cpdist, we first have to define the BN:

```
> library(bnlearn)
> dag.bnlearn <- model2network("[G][E][V|G:E][N|V][W|V][C|N:W]")
> disE <- list(coef = c("(Intercept)" = 50), sd = 10)
> disG <- list(coef = c("(Intercept)" = 50), sd = 10)
> disV <- list(coef = c("(Intercept)" = -10.35534,
+                 E = 0.70711, G = 0.5), sd = 5)
> disN <- list(coef = c("(Intercept)" = 45, V = 0.1),
+               sd = 9.949874)
> disW <- list(coef = c("(Intercept)" = 15, V = 0.7),
+               sd = 7.141428)
> disC <- list(coef = c("(Intercept)" = 0, N = 0.3, W = 0.7),
+               sd = 6.25)
> dis.list <- list(E = disE, G = disG, V = disV, N = disN,
+                   W = disW, C = disC)
> gbn.bnlearn <- custom.fit(dag.bnlearn, dist = dis.list)
> simu2 <- cpdist(gbn.bnlearn, nodes = nodes(gbn.bnlearn),
+             evidence = TRUE)
```

The comparison can be made in different ways; here we will just extract the medians for all nodes.

```
> su1 <- summary(simu1)
> su2 <- summary(simu2)
> cbind(simu1 = su1[3, ], simu2 = su2[3, ])
        simu1            simu2
     sG "Median :54.8  " "Median :49.3  "
     sE "Median :50.3  " "Median :49.4  "
     sV "Median :49.9  " "Median :49.9  "
     sN "Median :53.6  " "Median :49.2  "
     sW "Median :49.3  " "Median :49.0  "
     sC "Median :50.7  " "Median :49.1  "
```

Indeed even if all results are around the expected value of 50, there are differences; but they are comparable within each column and between different columns. They are the consequence of the small size of the simulation, making the same calculation with `ns` $= 10^4$ gives a more precise result, even if still approximate.

2.5 Imagine two ways other than changing the size of the points (as in Section 2.7.2) to introduce a third variable in the plot.

This is not an easy question. Many approaches have been proposed and the user has to choose that best suited to the data. Here, denoting with $(\mathtt{A}, \mathtt{B}, \mathtt{C})$ the three variables we want to explore, we illustrate three possible choices.

- The simplest choice is perhaps to make three scatter plots of two variables each, dividing the plot into four quarters.

  1. Making `plot(A, B)` into the top right quarter,
  2. making `plot(A, C)` into the bottom right quarter (A on the same scale as in the previous plot),
  3. making `plot(B, C)` into the bottom left quarter (C on the same scale as in the previous plot).

  The top left quarter is left empty.

- Another choice is to associate a different symbol or shape to each point depending on the values of the three variables. A possible example is the figure shown in Hartigan (1975, page 39), in which each point is associated to a rectangular box having the values of the three variables (assumed to be positive) as dimensions. Some points will be represented with tall boxes, others with flat boxes; the depth is also important.

- Finally, we can create a true 3D plot and rotate it interactively to assess the distance between the points. Suitable functions are available in R, for instance in the **rgl** package.

2.6 Can GBNs be extended to log-normal distributions? If so how, if not, why?

Of course this is possible and very easy! Just take the logarithm of the initial variables and apply a GBN to the transformed variables. There is no need to do that for all variables; we can transform only some of them. Note that it is necessary that all possible values of the variables to transform are positive. From a practical point of view, a constant can be added to satisfy this constraint, giving access to a third parameter for

$$A \rightarrow \log\left(A+k\right) \sim N\left(\mu, \sigma^2\right). \tag{C.1}$$

In general any univariate or multivariate transformation can be applied to a set of variables before attempting a model based on a GBN.

2.7 How can we generalise GBNs as defined in Section 2.3 in order to make each node's variance depend on the node's parents?

Indeed a strong restriction of GBNs is that the conditional variance of a node with respect to its parents must be a constant. There is no difficulty in proposing a generalisation. For instance, if `A` is the unique parent of `B`, we could assume that

$$\mathrm{VAR}(B \mid A) = (A - \mathrm{E}(A))^2 \times \sigma_B^2.$$

This is not usually done because most of BN's methodological developments are very general and do not discuss specific, limited cases of ad-hoc node parameterisations such as most instances of hybrid BNs.

2.8 From the first three lines of Table 2.1, prove that the joint distribution of `E`, `G` and `V` is trivariate normal.

To prove it, we will use the following result: the logarithm of the density of a multivariate normal distribution is, up to an additive constant:

$$\mathrm{f}(\mathbf{x}) \propto -\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^T \Sigma^{-2}\left(\mathbf{x}-\boldsymbol{\mu}\right)$$

where $\mathbf{x}$ stands for the value of the random vector, $\boldsymbol{\mu}$ is its expectation and Σ its covariance matrix.

To simplify the notation, we will first transform the three variables to give them a zero marginal expectation and a unity marginal variance.

$$\begin{aligned}
\widetilde{\mathtt{G}} &= \frac{\mathtt{G} - \mathrm{E}(\mathtt{G})}{\sqrt{\mathrm{VAR}(\mathtt{G})}} = \frac{\mathtt{G} - 50}{10} \\
\widetilde{\mathtt{E}} &= \frac{\mathtt{E} - \mathrm{E}(\mathtt{E})}{\sqrt{\mathrm{VAR}(\mathtt{E})}} = \frac{\mathtt{E} - 50}{10} \\
\widetilde{\mathtt{V}} &= \frac{\mathtt{V} - \mathrm{E}(\mathtt{V})}{\sqrt{\mathrm{VAR}(\mathtt{V})}} = \frac{\mathtt{V} - 50}{10}
\end{aligned}$$

The resulting normalised variables are

$$\begin{aligned}
\widetilde{\mathtt{G}} &\sim N(0,1), \\
\widetilde{\mathtt{E}} &\sim N(0,1), \\
\widetilde{\mathtt{V}} \mid \widetilde{\mathtt{G}}, \widetilde{\mathtt{E}} &\sim N\left(\frac{1}{2}\widetilde{\mathtt{G}} + \sqrt{\frac{1}{2}}\widetilde{\mathtt{E}}, \left(\frac{1}{2}\right)^2\right).
\end{aligned}$$

We are now able to compute the joint density of the three transformed variables

$$\begin{aligned}
\mathrm{f}\left(\widetilde{\mathtt{G}} = g, \widetilde{\mathtt{E}} = e, \widetilde{\mathtt{V}} = v\right) &\propto \mathrm{f}\left(\widetilde{\mathtt{G}} = g\right) + \mathrm{f}\left(\widetilde{\mathtt{E}} = e\right) + \mathrm{f}\left(\widetilde{\mathtt{V}} = v \mid \widetilde{\mathtt{G}} = g, \widetilde{\mathtt{E}} = e\right) \\
&= -\frac{g^2}{2} - \frac{e^2}{2} - 2\left(v - \frac{1}{2}g - \sqrt{\frac{1}{2}}e\right)^2 \\
&= -\begin{bmatrix} g \\ e \\ v \end{bmatrix}^T \begin{bmatrix} 1 & \frac{\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{3}{2} & -\sqrt{2} \\ -1 & -\sqrt{2} & 2 \end{bmatrix} \begin{bmatrix} g \\ e \\ v \end{bmatrix} \\
&= -\frac{1}{2}\begin{bmatrix} g \\ e \\ v \end{bmatrix}^T \begin{bmatrix} 1 & 0 & \frac{1}{2} \\ 0 & 1 & \sqrt{\frac{1}{2}} \\ \frac{1}{2} & \sqrt{\frac{1}{2}} & 1 \end{bmatrix}^{-1} \begin{bmatrix} g \\ e \\ v \end{bmatrix}.
\end{aligned}$$

Now

$$\mathrm{VAR}\left(\begin{bmatrix} \widetilde{\mathtt{G}} \\ \widetilde{\mathtt{E}} \\ \widetilde{\mathtt{V}} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & \frac{1}{2} \\ 0 & 1 & \sqrt{\frac{1}{2}} \\ \frac{1}{2} & \sqrt{\frac{1}{2}} & 1 \end{bmatrix}$$

which completes the proof.

Exercises of Chapter 3

3.1 Explain why it is logical to get a three-step function for the discretised approach in Figure 3.2.

For those observations for which the diameter is less than 6.16, the value of the discretised transform of D is always the lower interval, and the probability of predicting a particular supplier is constant on this interval. The same argument applies to the second and third intervals. The result is the step function shown in Figure 3.2. The level of the steps fits well with the continuous function with respect to the interval limits. More intervals would produce a better fit but the resulting variable would be more difficult to interpret.

3.2 Starting from the BUGS model in Section 3.1.1, write another BUGS model for the discretised model proposed in Section 3.1.2. The functions required for this task are described in the JAGS manual.

Here is such a model:

```
model {
  csup ~ dcat(sp);
  ddiam ~ dcat(dsk[, csup]);
}
```

Compared to the continuous version illustrated in Chapter 3, we just replaced the normal distribution with a categorical distribution. The probability vector associated to each supplier is in a column of matrix `dsk`. By the way, this shows that matrices can be used when defining BUGS models, and arrays as well.

3.3 Let $d = 6.0, 6.1, 6.2, 6.4$**.**

1. **Using the BUGS model proposed in Section 3.1.1, write the R script to estimate $P\left(\texttt{S} = \texttt{s2} \mid \texttt{D} = \texttt{d}\right)$ for the continuous approach demonstrated in the same section.**
2. **Using the BUGS model obtained in Exercise 3.2, write the R script to estimate $P\left(\texttt{S} = \texttt{s2} \mid \texttt{D} = \texttt{d}\right)$ for the discretised approach suggested in Section 3.1.2.**

And check the results with Figure 3.2.

1. We just have to use the proposed R commands of Section 3.1.1 wrapping them in a loop for the four values of d.

   ```
   > library(rjags)
   > sp <- c(0.5, 0.5)
   > mu <- c(6.1, 6.25)
   > sigma <- 0.05
   ```

```
> set.seed(98765)
> diameters <- c(6.0, 6.1, 6.2, 6.4)
> probas <- rep(NA, length(diameters))
> names(probas) <- diameters
> for (ii in seq(length(diameters))) {
+   jags.data <- list(sp = sp, mu = mu, sigma = sigma,
+                     cdiam = diameters[ii])
+   model1 <- jags.model(file = "../chap3/inclu.sc.jam",
+                           data = jags.data, quiet = TRUE)
+   update(model1, n.iter = 10000)
+   simu <- coda.samples(model = model1,
+              variable.names = "csup", n.iter = 20000,
+              thin = 20)[[1]]
+   probas[ii] <- sum(simu == 1)/length(simu)
+ }#FOR
> probas

    6   6.1   6.2   6.4
1.000 0.989 0.176 0.000
```

2. Almost the same script can be used for the discretised case, but we can simplify the calculation since the values of d are only in the first and third intervals.

```
> library(rjags)
> sp <- c(0.5, 0.5)
> dsk <- matrix(c(0.88493, 0.07914, 0.03593,
+                 0.03593, 0.07914, 0.88493), 3)
> set.seed(98765)
> diameters <- c(6.0, 6.1, 6.2, 6.4)
> ddiam <- c(1, 3)
> probas <- rep(NA, length(diameters))
> names(probas) <- diameters
> for (ii in seq(length(ddiam))) {
+   jags.data <- list(sp = sp, dsk = dsk,
+                     ddiam = ddiam[ii])
+   model2 <- jags.model(file = "exo3.2.jam",
+                           data = jags.data, quiet = TRUE)
+   update(model2, n.iter = 10000)
+   simu <- coda.samples(model = model2,
+              variable.names = "csup", n.iter = 20000,
+              thin = 20)[[1]]
+   probas[(1:2) + 2 * (ii - 1)] <-
+                              sum(simu == 1)/length(simu)
+ }#FOR
> probas
```

```
    6   6.1   6.2   6.4
0.956 0.956 0.033 0.033
```

Indeed the estimated probabilities are the same as in Figure 3.2.

3.4 In Section 3.1.1, the probability that the supplier is `s1` knowing that the diameter is 6.2 was estimated to be 0.1824 which is not identical to the value obtained with JAGS.

1. **Explain why the calculation with the R function `dnorm` is right and why the value 0.1824 is correct. Can you explain why the JAGS result is not exact? Propose a way to improve it.**
2. **Would this value be different if we modify the marginal distribution for the two suppliers?**

1. `dnorm` is based on closed form formulas while JAGS calculations are produced by simulations, and are always approximations; just changing the seed of the pseudo-random generator changes the result. Simulations obtained with JAGS can give arbitrarily precise results by increasing the number of iterations…but the required number of iterations can be very large.
2. The result will be different since the marginal distributions are part of the calculation of the conditional probability resulting from Bayes' formula. This is underlined in the caption of Figure 3.1.

3.5 Revisiting the discretisation in Section 3.1.2, compute the conditional probability tables for `D` | `S` and `S` | `D` when the interval boundaries are set to (6.10, 6.18) instead of (6.16, 6.19).

Compared to the results presented in Section 3.1.2, what is your conclusion?

To get `D` | `S`, you just have to use the following R code.

```
> limits <- c(6.10, 6.18)
> dsd <- matrix(c(diff(c(0, pnorm(limits, mu[1], sigma), 1)),
+                 diff(c(0, pnorm(limits, mu[2], sigma), 1))),
+               3, 2)
> dimnames(dsd) <- list(D = c("thin", "average", "thick"),
+                       S = c("s1", "s2"))
> dsd
         S
D             s1      s2
  thin    0.5000 0.00135
  average 0.4452 0.07941
  thick   0.0548 0.91924
```

To get S | D, we can apply Bayes' theorem.

```
> jointd <- dsd/2
> mardd <- rowSums(jointd)
> dds <- t(jointd / mardd)
> dds
    D
S       thin average  thick
  s1 0.99731   0.849 0.0563
  s2 0.00269   0.151 0.9437
```

It seems that the `average` category of diameters is not very useful because it behaves the same as the lower category, giving a high probability to supplier `s1`. This could be illustrated with a plot similar to Figure 3.1.

Exercises of Chapter 4

4.1 Consider the `survey` data set from Chapter 1.

1. **Learn a BN with the IAMB algorithm and the asymptotic mutual information test.**
2. **Learn a second BN with IAMB but using only the first 100 observations of the data set. Is there a significant loss of information in the resulting BN compared to the BN learned from the whole data set?**
3. **Repeat the structure learning in the previous point with IAMB and the Monte Carlo and sequential Monte Carlo mutual information tests. How do the resulting networks compare with the BN learned with the asymptotic test? Is the increased execution time justified?**

1.
```
> survey <- read.table("../chap1/survey.txt",
+              header = TRUE)
> dag <- iamb(survey, test = "mi")
```

2.
```
> dag100 <- iamb(survey[1:100, ], test = "mi")
> nrow(directed.arcs(dag))
[1] 0
> nrow(undirected.arcs(dag))
[1] 8
```

```
> nrow(directed.arcs(dag100))
[1] 0
> nrow(undirected.arcs(dag100))
[1] 2
```

While both DAGs are very different from that in Figure 1.1, `dag100` has only a single arc; not enough information is present in the first 100 observations to learn the correct structure. In both cases all arcs are undirected. After assigning directions with `cextend`, we can see that `dag100` has a much lower score than `dag`, which confirms that `dag100` is not as good a fit for the data as `dag`.

```
> score(cextend(dag), survey, type = "bic")
[1] -1999.259
> score(cextend(dag100), survey, type = "bic")
[1] -2008.116
```

3. The BIC score computed from the first 100 observations does not increase when using Monte Carlo tests, and the DAGs we learn still have just a single arc. There is no apparent benefit over the corresponding asymptotic test.

```
> dag100.mc <- iamb(survey[1:100, ], test = "mc-mi")
> narcs(dag100.mc)
[1] 1
> dag100.smc <- iamb(survey[1:100, ], test = "smc-mi")
> narcs(dag100.smc)
[1] 1
> score(cextend(dag100.mc), survey, type = "bic")
[1] -2008.116
> score(cextend(dag100.smc), survey, type = "bic")
[1] -2008.116
```

4.2 Consider again the `survey` data set from Chapter 1.

1. **Learn a BN using Bayesian posteriors for both structure and parameter learning, in both cases with `iss = 5`.**
2. **Repeat structure learning with `hc` and 3 random restarts and with `tabu`. How do the BNs differ? Is there any evidence of numerical or convergence problems?**

3. **Use increasingly large subsets of the `survey` data to check empirically that BIC and BDe are asymptotically equivalent.**

1.
```
> dag <- hc(survey, score = "bde", iss = 5)
> bn <- bn.fit(dag, survey, method = "bayes", iss = 5)
```

2.
```
> dag.hc3 <-  hc(survey, score = "bde", iss = 5,
+                  restart = 3)
> dag.tabu <-  tabu(survey, score = "bde", iss = 5)
> modelstring(dag.hc3)

[1] "[R][E|R][T|R][A|E][O|E][S|E]"

> modelstring(dag.tabu)

[1] "[O][S][E|O:S][A|E][R|E][T|R]"
```

The two DAGs are quite different; from the model strings above, `hc` seems to learn a structure that is closer to that in Figure 1.1. The BIC scores of `dag.hc3` and `dag.tabu` support the conclusion that `hc` with random restarts is a better fit for the data.

```
> score(dag.hc3, survey)

[1] -1998.432

> score(dag.tabu, survey)

[1] -1999.733
```

Using the `debug` option to explore the learning process we can confirm that no numerical problem is apparent, because all the DAGs learned from the random restarts fit the data reasonably well.

4.3 Consider the `marks` data set from Section 4.7.

1. **Create a `bn` object describing the graph in the bottom right panel of Figure 4.5 and call it `mdag`.**
2. **Construct the skeleton, the CPDAG and the moral graph of `mdag`.**
3. **Discretise the `marks` data using `"interval"` discretisation with 2, 3 and 4 intervals.**
4. **Perform structure learning with `hc` on each of the discretised data sets; how do the resulting DAGs differ?**

1.
```
> mdag <-
+   model2network(paste("[ANL][MECH][LAT|ANL:MECH]",
+           "[VECT|LAT][ALG|LAT][STAT|LAT]", sep = ""))
```

2.
```
> mdag.sk <- skeleton(mdag)
> mdag.cpdag <- cpdag(mdag)
> mdag.moral <- moral(mdag)
```

3.
```
> data(marks)
> dmarks2 <- discretize(marks, "interval", breaks = 2)
> dmarks3 <- discretize(marks, "interval", breaks = 3)
> dmarks4 <- discretize(marks, "interval", breaks = 4)
```

4.
```
> dag2 <- hc(dmarks2)
> dag3 <- hc(dmarks3)
> dag4 <- hc(dmarks4)
> modelstring(dag2)

[1] "[MECH][VECT|MECH][ALG|VECT][ANL|ALG][STAT|ALG]"

> modelstring(dag3)

[1] "[MECH][ALG|MECH][ANL|ALG][STAT|ALG][VECT|ANL]"

> modelstring(dag4)

[1] "[MECH][VECT][ALG][ANL|ALG][STAT|ANL]"
```

From the output above, we can deduce that using 4 intervals for discretising the data breaks all the dependencies between the variables; dag4 has only two arcs. Using 2 or 3 intervals results in DAGs with 4 arcs, which is closer to the 6 arcs of the true structure. However, the DAGs are still quite different from the latter, suggesting that a noticeable amount of information is lost in the discretisation.

Exercises of Chapter 5

5.1 One essential task in any analysis is to import and export the R objects describing models from different packages. This is all the more true in the case of BN modelling, as no package implements all of structure learning, parameter learning and inference.

1. **Create the `dag.bnlearn` object from Section 5.1.1.**

2. **Export it to deal.**
3. **Import the result back into bnlearn.**
4. **Export `dag.bnlearn` to catnet and import it back in bnlearn.**
5. **Perform parameter learning using the discretised `dmarks` and `dag.bnlearn` and export it to a DSC file, which can be read in Hugin and GeNIe.**

1.
```
> library(bnlearn)
> dag.bnlearn <- model2network(
+   paste("[ANL][MECH][LAT|ANL:MECH]",
+         "[VECT|LAT][ALG|LAT][STAT|LAT]", sep = ""))
```

2.
```
> library(deal)
> data(marks)
> latent <- factor(c(rep("A", 44), "B",
+                    rep("A", 7), rep("B", 36)))
> marks$LAT <- latent
> ord <- bnlearn::node.ordering(dag.bnlearn)
> bn <- deal::network(marks[, ord])
> bn <- deal::as.network(
+               bnlearn::modelstring(dag.bnlearn), bn)
> all.equal(deal::modelstring(bn),
+           bnlearn::modelstring(dag.bnlearn))

[1] TRUE
```

3.
```
> dag2 <- bnlearn::model2network(deal::modelstring(bn))
```

4.
```
> library(catnet)
> nodes <- bnlearn::nodes(dag.bnlearn)
> edges <- sapply(bnlearn::nodes(dag.bnlearn),
+            bnlearn::children, x = dag.bnlearn)
> dag.catnet <- cnCatnetFromEdges(nodes, edges)
> dag2 <- empty.graph(nodes)
> arcs(dag2) <- cnMatEdges(dag.catnet)
> all.equal(dag.bnlearn, dag2)

[1] TRUE
```

5.
```
> dmarks <- discretize(marks[, 1:6], breaks = 2,
+            method = "interval")
> dmarks$LAT <- marks$LAT
> bn <- bn.fit(dag.bnlearn, dmarks)
> write.dsc(bn, file = "dmarks.dsc")
```

5.2 Learn a GBN from the marks data (without the LAT variable) using pcalg and a custom test that defines dependence as significant if the corresponding partial correlation is greater than 0.50.

```
> library(pcalg)
> customCItest = function(x, y, S, suffStat) {
+   pcor <- cov2cor(solve(suffStat$C[c(x, y, S), c(x, y, S)]))
+   pcor[1, 2]
+ }#CUSTOMCITEST
> suffStat <- list(C = cor(marks), n = nrow(marks))
> pc.fit <- pc(suffStat, indepTest = customCItest,
+              p = ncol(marks), alpha = 0.50)
```

5.3 Reproduce the example of structure learning from Section 5.1.1 using deal, but set the imaginary sample size to 20. How does the resulting network change?

```
> library(deal)
> latent <- factor(c(rep("A", 44), "B",
+                    rep("A", 7), rep("B", 36)))
> marks$LAT <- latent
> net <- network(marks)
> prior <- jointprior(net, N = 20)
Imaginary sample size: 20
> net <- learn(net, marks, prior)$nw
> best <- autosearch(net, marks, prior)$nw
> best
##  6 ( 1 discrete+ 5 ) nodes;score= -1750.773;relscore= 0.0178
1       MECH    continuous()    2       3
2       VECT    continuous()    3       4       6
3       ALG     continuous()    4       6
4       ANL     continuous()    6
5       STAT    continuous()    1       3       4       6
6       LAT     discrete(2)
```

The network learned with imaginary sample size 20 has 12 arcs, compared to the 9 arcs of the network learned in Section 5.1.1. This is a known effect of choosing a uniform prior and giving it a weight on the same order of magnitude of the observed sample size: the resulting graphs are not sparse, and typically have many more arcs than nodes.

Bibliography

Agresti, A. (2013). *Categorical Data Analysis.* Wiley, 3rd edition.

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., and Xenofon, X. D. (2010). Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I: Algorithms and Empirical Evaluation. *Journal of Machine Learning Research*, 11:171–234.

Andersen, S., Olesen, K., Jensen, F., and Jensen, F. (1989). HUGIN – a Shell for Building Bayesian Belief Universes for Expert Systems. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1080–1085.

Anderson, T. W. (2003). *An Introduction to Multivariate Statistical Analysis.* Wiley, 3rd edition.

Andreassen, S., Jensen, F., Andersen, S., Falck, B., Kjærulff, U., Woldbye, M., Sørensen, A., Rosenfalck, A., and Jensen, F. (1989). MUNIN – An Expert EMG Assistant. In Desmedt, J. E., editor, *Computer-Aided Electromyography and Expert Systems*, pages 255–277. Elsevier.

Ash, R. B. (2000). *Probability and Measure Theory.* Academic Press, 2nd edition.

Balov, N. (2013). *mugnet: Mixture of Gaussian Bayesian Network Model.* R package version 1.01.3.

Balov, N. and Salzman, P. (2013). *catnet: Categorical Bayesian Network Inference.* R package version 1.14.2.

Bang-Jensen, J. and Gutin, G. (2009). *Digraphs: Theory, Algorithms and Applications.* Springer, 2nd edition.

Bøttcher, S. G. and Dethlefsen, C. (2003). deal: A Package for Learning Bayesian Networks. *Journal of Statistical Software*, 8(20):1–40.

Bouckaert, R. R. (1995). *Bayesian Belief Networks: From Construction to Inference.* PhD thesis, Utrecht University, The Netherlands.

Bühlmann, P., Kalisch, M., and Maathuis, M. H. (2010). Variable Selection in High-Dimensional Linear Models: Partially Faithful Distributions and the PC-Simple Algorithm. *Biometrika*, 97(2):261–278.

Castillo, E., Gutiérrez, J. M., and Hadi, A. S. (1997). *Expert Systems and Probabilistic Network Models.* Springer.

Centers for Disease Control and Prevention (2004). *The 1999-2004 Dual Energy X-ray Absorptiometry (DXA) Multiple Imputation Data Files and Technical Documentation.* National Center for Health Statistics, Hyattsville, MD (USA).

Centers for Disease Control and Prevention (2010). *National Health and Nutrition Examination Survey: Body Composition Procedures Manual.* National Center for Health Statistics, Hyattsville, MD (USA).

Cheng, J. and Druzdzel, M. J. (2000). AIS-BN: An Adaptive Importance Sampling Algorithm for Evidential Reasoning in Large Bayesian Networks. *Journal of Artificial Intelligence Research*, 13:155–188.

Cooper, G. F. and Yoo, C. (1999). Causal Discovery from a Mixture of Experimental and Observational Data. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 116–125. Morgan Kaufmann.

Crawley, M. J. (2013). *The R Book.* Wiley, 2nd edition.

DeGroot, M. H. and Scherivsh, M. J. (2012). *Probability and Statistcs.* Prentice Hall, 4th edition.

Denis, J.-B. (2013). *rbmn: Handling Linear Gaussian Bayesian Networks.* R package version 0.9-2.

Diestel, R. (2005). *Graph Theory.* Springer, 3rd edition.

Edwards, D. I. (2000). *Introduction to Graphical Modelling.* Springer, 2nd edition.

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications.* Wiley, 3rd edition.

Friedman, N. and Koller, D. (2003). Being Bayesian about Bayesian Network Structure: A Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning*, 50(1–2):95–126.

Friedman, N., Linial, M., Nachman, I., and Pe'er, D. (2000). Using Bayesian Network to Analyze Expression Data. *Journal of Computational Biology*, 7:601–620.

Friedman, N., Pe'er, D., and Nachman, I. (1999). Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In *Proceedings of 15th Conference on Uncertainty in Artificial Intelligence*, pages 206–221. Morgan Kaufmann.

Geiger, D. and Heckerman, D. (1994). Learning Gaussian Networks. Technical report, Microsoft Research, Redmond, Washington. Available as Technical Report MSR-TR-94-10.

Goeman, J. J. (2012). *penalized R Package.* R package version 0.9-42.

Hartemink, A. J. (2001). *Principled Computational Methods for the Validation and Discovery of Genetic Regulatory Networks.* PhD thesis, School of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Hartigan, J. A. (1975). *Clustering Algorithms.* Wiley.

Hartley, S. W. and Sebastiani, P. (2013). PleioGRiP: Genetic Risk Prediction with Pleiotropy. *Bioinformatics*, 29(8):1086–1088.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2nd edition.

Hausser, J. and Strimmer, K. (2009). Entropy Inference and the James-Stein Estimator, with Application to Nonlinear Gene Association Networks. *Journal of Machine Learning Research*, 10:1469–1484.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243. Available as Technical Report MSR-TR-94-09.

Højsgaard, S. (2013). *gRain: Graphical Independence Networks.* R package version 1.2-2.

Højsgaard, S., Dethlefsen, C., and Bowsher, C. (2013). *gRbase: A Package for Graphical Modelling in R.* R package version 1.6-12.

Højsgaard, S., Edwards, D., and Lauritzen, S. (2012). *Graphical Models with R.* Use R! series. Springer.

Holmes, D. E. and Jaim, L. C. (2008). *Innovations in Bayesian Networks: Theory and Applications.* Springer.

Ide, J. S. and Cozman, F. G. (2002). Random Generation of Bayesian Networks. In *SBIA '02: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence*, pages 366–375. Springer-Verlag.

Johnson, N. L., Kemp, A. W., and Kotz, S. (2005). *Univariate Discrete Distributions.* Wiley, 3rd edition.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, volume 1. Wiley, 2nd edition.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1995). *Continuous Univariate Distributions*, volume 2. Wiley, 2nd edition.

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions.* Wiley.

Kalisch, M. and Bühlmann, P. (2007). Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. *Journal of Machine Learning Research*, 8:613–636.

Kalisch, M. and Bühlmann, P. (2008). Robustification of the PC-Algorithm for Directed Acyclic Graphs. *Journal of Computational and Graphical Statistics*, 17(4):773–789.

Kalisch, M., Mächler, M., Colombo, D., Maathuis, M. H., and Bühlmann, P. (2012). Causal Inference Using Graphical Models with the R Package pcalg. *Journal of Statistical Software*, 47(11):1–26.

Kenett, R. S., Perruca, G., and Salini, S. (2012). *Modern Analysis of Customer Surveys: With Applications Using R*, chapter 11. Wiley.

Kjærluff, U. B. and Madsen, A. L. (2008). *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis.* Springer.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques.* MIT Press.

Korb, K. B. and Nicholson, A. E. (2004). *Bayesian Artificial Intelligence.* Chapman and Hall.

Koski, T. and Noble, J. M. (2009). *Bayesian Networks: An Introduction.* Wiley.

Kotz, S., Balakrishnan, N., and Johnson, N. L. (2000). *Continuous Multivariate Distributions.* Wiley, 2nd edition.

Kulinskaya, E., Morgenthaler, S., and Staudte, R. G. (2008). *Meta Analysis: A Guide to Calibrating and Combining Statistical Evidence.* Wiley.

Kullback, S. (1968). *Information Theory and Statistics.* Dover Publications.

Larrañaga, P., Sierra, B., Gallego, M. J., Michelena, M. J., and Picaza, J. M. (1997). Learning Bayesian Networks by Genetic Algorithms: A Case Study in the Prediction of Survival in Malignant Skin Melanoma. In *Proceedings of the 6th Conference on Artificial Intelligence in Medicine in Europe (AIME'97)*, pages 261–272. Springer.

Lauritzen, S. (1996). *Graphical Models.* Oxford University Press.

Loève, M. (1977). *Probability Theory.* Springer-Verlag, 4th edition.

Lunn, D., Jackson, C., Best, N., Thomas, A., and Spiegelhalter, D. (2012). *The BUGS Book: A Practical Introduction to Bayesian Analysis.* Chapman & Hall.

Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The BUGS project: Evolution, Critique and Future Directions (with discussion). *Statistics in Medicine*, 28(5):3049–3067.

Lunn, D., Thomas, A., Best, N., and Spiegelhalter, D. (2000). WinBUGS – A Bayesian modelling framework: Concepts, Structure, and Extensibility. *Statistics and Computing*, 10(4):325–337.

Mardia, K. V., Kent, J. T., and Bibby, J. M. (1979). *Multivariate Analysis*. Academic Press.

Margaritis, D. (2003). *Learning Bayesian Network Model Structure from Data*. PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. Available as Technical Report CMU-CS-03-153.

Melançon, G., Dutour, I., and Bousquet-Mélou, M. (2001). Random Generation of Directed Acyclic Graphs. *Electronic Notes in Discrete Mathematics*, 10:202–207.

Morota, G., Valente, B. D., Rosa, G. J. M., Weigel, K. A., and Gianola, D. (2012). An Assessment of Linkage Disequilibrium in Holstein Cattle Using a Bayesian Network. *Journal of Animal Breeding and Genetics*, 129(6):474–487.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nagarajan, R., Scutari, M., and Lèbre, S. (2013). *Bayesian Networks in R with Applications in Systems Biology*. Use R! series. Springer.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice Hall.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (2009). *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition.

Plummer, M. (2003). JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, pages 1–10.

Plummer, M. (2012). *JAGS Version 3.3.0 User Manual*.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6(1):7–11.

Pourret, O., Naim, P., and Marcot, B. (2008). *Bayesian Networks: A Practical Guide to Applications*. Wiley.

Robert, C. P. and Casella, G. (2009). *Introducing Monte Carlo Methods with R*. Springer.

Ross, S. (2012). *A First Course in Probability*. Prentice Hall, 9th edition.

Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D. A., and Nolan, G. P. (2005). Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data. *Science*, 308(5721):523–529.

Schäfer, J., Opgen-Rhein, R., Zuber, V., Ahdesmäki, M., Silva, A. P. D., and Strimmer., K. (2013). *corpcor: Efficient Estimation of Covariance and (Partial) Correlation*. R package version 1.6.6.

Scutari, M. (2010). Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 35(3):1–22.

Scutari, M. and Brogini, A. (2012). Bayesian Network Structure Learning with Permutation Tests. *Communications in Statistics – Theory and Methods*, 41(16–17):3233–3243. Special Issue "Statistics for Complex Problems: Permutation Testing Methods and Related Topics". Proceedings of the Conference "Statistics for Complex Problems: the Multivariate Permutation Approach and Related Topics", Padova, June 14–15, 2010.

Scutari, M. and Nagarajan, R. (2013). On Identifying Significant Edges in Graphical Models of Molecular Networks. *Artificial Intelligence in Medicine*, 57(3):207–217. Special Issue containing the Proceedings of the Workshop "Probabilistic Problem Solving in Biomedicine" of the 13th Artificial Intelligence in Medicine (AIME) Conference, Bled (Slovenia), July 2, 2011.

Shäfer, J. and Strimmer, K. (2005). A Shrinkage Approach to Large-Scale Covariance Matrix Estimation and Implications for Functional Genomics. *Statistical Applications in Genetics and Molecular Biology*, 4:32.

Spector, P. (2009). *Data Manipulation with R*. Springer-Verlag.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation, Prediction, and Search*. MIT Press.

Stan Development Team (2013). *Stan: A C++ Library for Probability and Sampling, Version 2.1*.

Sturtz, S., Ligges, U., and Gelman, A. (2005). R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software*, 12(3):1–16.

Thomas, A., O'Hara, B., Ligges, U., and Sturtz, S. (2006). Making BUGS Open. *R News*, 6(1):12–17.

Tsamardinos, I., Aliferis, C. F., and Statnikov, A. (2003). Algorithms for Large Scale Markov Blanket Discovery. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference*, pages 376–381. AAAI Press.

Tsamardinos, I. and Borboudakis, G. (2010). Permutation Testing Improves Bayesian Network Learning. In Balcázar, J., Bonchi, F., Gionis, A., and Sebag, M., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 322–337. Springer.

Tsamardinos, I., Brown, L. E., and Aliferis, C. F. (2006). The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *Machine Learning*, 65(1):31–78.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S.* Springer, 4th edition.

Verma, T. S. and Pearl, J. (1991). Equivalence and Synthesis of Causal Models. *Uncertainty in Artificial Intelligence*, 6:255–268.

Villa-Vialaneix, N., Liaubet, L., Laurent, T., Cherel, P., Gamot, A., and San-Cristobal, M. (2013). The Structure of a Gene Co-Expression Network Reveals Biological Functions Underlying eQTLs. *PLOS ONE*, 8(4):e60045.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics.* Wiley.

Yaramakala, S. and Margaritis, D. (2005). Speculative Markov Blanket Discovery for Optimal Feature Selection. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 809–812. IEEE Computer Society.

Yu, J., Smith, V. A., Wang, P. P., Hartemink, A. J., and Jarvis, E. D. (2004). Advances to Bayesian Network Inference for Generating Causal Networks from Observational Biological Data. *Bioinformatics*, 20(18):3594–3603.

Yuan, C. and Druzdzel, M. J. (2003). An Importance Sampling Algorithm Based on Evidence Pre-Propagation. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, pages 624–631.

Zou, M. and Conzen, S. D. (2005). A New Dynamic Bayesian Network (DBN) Approach for Identifying Gene Regulatory Networks from Time Course Microarray Data. *Bioinformatics*, 21(1):71–79.

Statistics

Bayesian Networks: With Examples in R introduces Bayesian networks using a hands-on approach. Simple yet meaningful examples in R illustrate each step of the modeling process. The examples start from the simplest notions and gradually increase in complexity. The authors also distinguish the probabilistic models from their estimation with data sets.

The first three chapters explain the whole process of Bayesian network modeling, from structure learning to parameter learning to inference. These chapters cover discrete Bayesian, Gaussian Bayesian, and hybrid networks, including arbitrary random variables.

The book then gives a concise but rigorous treatment of the fundamentals of Bayesian networks and offers an introduction to causal Bayesian networks. It also presents an overview of R and other software packages appropriate for Bayesian networks. The final chapter evaluates two real-world examples: a landmark causal protein signaling network paper and graphical modeling approaches for predicting the composition of different body parts.

Covering theoretical and practical aspects of Bayesian networks, this book provides you with an introductory overview of the field. It gives you a clear, practical understanding of the general approach and steps involved.

About the Authors

Marco Scutari is a research associate in statistical genetics at the Genetics Institute, University College London. His research focuses on the theory of Bayesian networks and their applications to biological data.

Jean-Baptiste Denis is a senior scientist in the Applied Mathematics and Computer Science Department at the French National Institute for Agricultural Research. His research interests are Bayesian approaches to statistics and networks, especially applications to microbiological food safety.

K22427

ISBN: 978-1-4822-2558-7

90000

9 781482 225587

CRC Press
Taylor & Francis Group
an informa business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK